

COBOL for AIX



プログラミング・ガイド

バージョン 3.1

COBOL for AIX



プログラミング・ガイド

バージョン 3.1

お願い

本書および本書で紹介する製品をご使用になる前に、831 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM COBOL for AIX のバージョン 3.1 (プログラム番号 5724-V62)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。製品のレベルに合った正しい版をご使用ください。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC23-8566-00
COBOL for AIX
Programming Guide
Version 3.1

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1版第1刷 2009.3

© Copyright International Business Machines Corporation 1995, 2009.

目次

表	xiii
前書き	xv
本書について	xv
本書の使い方	xv
略語	xv
構文図の読み方	xvi
例の示し方	xviii
関連情報	xviii
改訂の要約	xviii
バージョン 3.1 (2009 年 1 月)	xviii
COBOL for AIX のアクセシビリティ	xxi
アクセシビリティ機能	xxi
キーボード・ナビゲーション	xxii
インターフェース情報	xxii
アクセシビリティ関連情報	xxii
IBM およびアクセシビリティ	xxii

第 1 部 プログラムのコーディング . . . 1

第 1 章 プログラムの構造 5

プログラムの識別	5
プログラムを再帰的として識別する	6
収容プログラムによってプログラムに呼び出し可能のマークを付ける	6
プログラムを初期状態に設定する	6
ソース・リストのヘッダーの変更	7
コンピューター環境の記述	7
例: FILE-CONTROL 段落	8
照合シーケンスの指定	8
シンボリック文字を定義する	10
ユーザー定義のクラスを定義する	10
オペレーティング・システムに対してファイルを識別する	10
データの記述	12
入出力操作でのデータの使用	12
WORKING-STORAGE と LOCAL-STORAGE の比較	14
別のプログラムからのデータの使用	16
データの処理	17
PROCEDURE DIVISION 内でロジックが分割される方法	18
宣言	22

第 2 章 データの使用 23

変数、構造、リテラル、および定数の使用	23
変数の使用	23
データ項目とグループ項目の使用	24
リテラルの使用	26
定数の使用	26

形象定数の使用	27
データ項目への値の割り当て	27
例: データ項目の初期化	28
構造の初期化 (INITIALIZE)	31
基本データ項目への値の割り当て (MOVE)	33
グループ・データ項目への値の割り当て (MOVE)	34
算術結果の割り当て (MOVE または COMPUTE)	35
画面またはファイルからの入力への割り当て (ACCEPT)	35
画面上またはファイル内での値の表示 (DISPLAY)	37
組み込み関数の使用 (組み込み関数)	37
テーブル (配列) とポインターの使用	38

第 3 章 数値および算術演算 41

数値データの定義	41
数値データの表示	43
数値データの保管方法の制御	44
数値データの形式	45
外部 10 進数 (DISPLAY および NATIONAL) 項目	46
外部浮動小数点 (DISPLAY および NATIONAL) 項目	46
2 進数 (COMP) 項目	47
固有 2 進数 (COMP-5) 項目	47
バック 10 進数 (COMP-3) 項目	48
内部浮動小数点 (COMP-1 および COMP-2) 項目	48
例: 数値データおよび内部表現	49
データ形式の変換	51
変換および精度	52
ゾーンおよびバック 10 進数データのサイン表記	53
非互換データの検査 (数値のクラス・テスト)	54
算術の実行	54
COMPUTE およびその他の算術ステートメントの使用	54
算術式の使用	55
数字組み込み関数の使用	56
例: 数字組み込み関数	57
固定小数点演算と浮動小数点演算の対比	60
浮動小数点計算	60
固定小数点計算	61
算術比較 (比較条件)	61
例: 固定小数点計算および浮動小数点計算	61
通貨記号の使用	62
例: 複数の通貨符号	63

第 4 章 テーブルの処理 65

テーブルの定義 (OCCURS)	65
テーブルのネスト	67
例: 添え字付け	68
例: 指標付け	68
テーブル内の項目の参照	69

添え字付け	69
索引付け	70
テーブルに値を入れる方法	71
テーブルの動的なロード	72
テーブルの初期化 (INITIALIZE)	72
テーブルの定義時の値の割り当て (VALUE)	73
例: PERFORM と添え字付け	75
例: PERFORM および索引付け	76
可変長テーブルの作成 (DEPENDING ON)	77
可変長テーブルのロード	79
可変長テーブルへの値の割り当て	80
テーブルの探索	80
逐次探索 (SEARCH)	81
二分探索 (SEARCH ALL)	82
組み込み関数を使用したテーブル項目の処理	83
例: 組み込み関数を使用したテーブルの処理	84

第 5 章 プログラム・アクションの選択と 反復 85

プログラム・アクションの選択	85
アクションの選択項目のコーディング	85
条件式のコーディング	90
プログラム・アクションの繰り返し	94
インラインまたはライン外 PERFORM の選択	94
ループのコーディング	95
テーブルのループ処理	96
複数の段落またはセクションの実行	97

第 6 章 スtringの処理 99

データ項目の結合 (STRING)	99
例: STRING ステートメント	100
データ項目の分割 (UNSTRING)	102
例: UNSTRING ステートメント	103
ヌル終了Stringの取り扱い	105
例: ヌル終了String	106
データ項目のサブStringの参照	106
参照修飾子	108
例: 参照修飾子としての演算式	109
例: 参照修飾子としての組み込み関数	109
データ項目の計算および置換 (INSPECT)	110
例: INSPECT ステートメント	110
データ項目の変換 (組み込み関数)	111
大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)	112
逆順への変換 (REVERSE)	112
数値への変換 (NUMVAL、NUMVAL-C)	113
あるコード・ページから別のコード・ページへ の変換	114
データ項目の評価 (組み込み関数)	114
照合シーケンスに関する単一文字の評価	115
最大または最小データ項目の検出	115
データ項目の長さの検出	118
コンパイルの日付の検出	118

第 7 章 ファイルの処理 121

ファイルの識別	121
---------	-----

VSAM ファイルの識別	122
Encina SFS ファイルの識別	122
STL ファイルの識別	123
RSD ファイルの識別	123
ファイル・システム	124
VSAM ファイル・システム	125
SFS ファイル・システム	125
STL ファイル・システム	127
RSD ファイル・システム	129
2 GB を超える STL ファイルおよび VSAM フ ァイル	130
ファイル・システム要約	131
ファイル編成およびアクセス・モードの指定	132
ファイル編成およびアクセス・モード	132
ファイル・オープン時のエラーからの保護	136
ファイル状況フィールドの設定	136
ファイル構造の詳細記述	137
ファイルの入出カステートメントのコーディング	137
例: COBOL でのファイルのコーディング	138
ファイル位置標識	140
ファイルのオープン	140
ファイルからのレコードの読み取り	143
ファイルへのレコード書き込み時に使用するステ ートメント	144
ファイルへのレコードの追加	145
ファイル内のレコードの置換	145
ファイルからのレコードの削除	146
ファイルの更新に使用する PROCEDURE DIVISION ステートメント	147
COBOL アプリケーションで SFS ファイルにアク セスできるようにする	148
例: SFS ファイルへのアクセス	149
SFS パフォーマンスの向上	151

第 8 章 ファイルのソートおよびマージ 155

ソートおよびマージ・プロセス	156
ソートまたはマージ・ファイルの記述	156
ソートまたはマージへの入力の記述	157
例: SORT 用のソート・ファイルおよび入力ファ イルの記述	157
入力プロシージャのコーディング	158
ソートまたはマージからの出力の記述	159
出力プロシージャのコーディング	160
入出力プロシージャに関する制約事項	160
ソートまたはマージの要求	161
ソートまたはマージ基準の設定	162
代替照合シーケンスの選択	163
例: 入出力プロシージャを使用したソート	163
ソートまたはマージの成否の判断	164
ソートおよびマージ・エラー番号	165
ソートまたはマージ操作の途中停止	168

第 9 章 エラーの処理 169

Stringの結合および分割におけるエラーの処理	169
算術演算でのエラーの処理	170
例: 0 による除算の検査	170

入出力操作でのエラーの処理	171
ファイルの終わり条件 (AT END) の使用	172
ERROR 宣言のコーディング	173
ファイル状況キーの使用	174
ファイル・システム状況コードの使用	175
INVALID KEY 句のコーディング	177
プログラム呼び出し時のエラーの処理	178

第 2 部 各国語環境に合わせたプログラムの対応 181

第 10 章 国際環境でのデータの処理 183

COBOL ステートメントと国別データ	184
組み込み関数と国別データ	187
Unicode および言語文字のエンコード	188
COBOL での国別データ (Unicode) の使用	189
国別データ項目の定義	189
国別リテラルの使用	190
国別文字形象定数の使用	191
国別数値データ項目の定義	192
国別グループ	192
国別グループの使用	194
文字データの保管	196
国別 (Unicode) 表現との間の変換	197
英数字、DBCS、および整数から国別への変換 (MOVE)	198
英数字または DBCS から国別への変換 (NATIONAL-OF)	198
国別の英数字への変換 (DISPLAY-OF)	199
デフォルト・コード・ページのオーバーライド	199
例: 国別データとの間の変換	200
UTF-8 データの処理	201
中国語 GB 18030 データの処理	201
国別 (UTF-16) データの比較	202
2 つのクラス国別オペランドの比較	203
クラス国別オペランドとクラス数値オペランドの比較	204
国別数値オペランドと他の数値オペランドの比較	204
国別と他の文字ストリング・オペランドとの比較	204
国別データ・オペランドと英数字グループ・オペランドの比較	205
DBCS サポート用のコーディング	205
DBCS データの宣言	206
DBCS リテラルの使用	206
有効な DBCS 文字に関するテスト	207
DBCS データを含む英数字データ項目の処理	208

第 11 章 ロケールの設定 209

アクティブ・ロケール	209
文字データのコード・ページの指定	210
環境変数を使用したロケールの指定	211
システム設定からのロケールの決定	212
変換が使用可能なメッセージのタイプ	213
サポートされるロケールおよびコード・ページ	213
ロケール付きの照合シーケンスの制御	216

ロケール付きの英数字照合シーケンスの制御	217
ロケール付きの DBCS 照合シーケンスの制御	218
ロケール付きの国別照合シーケンスの制御	219
照合シーケンスに依存する組み込み関数	219
アクティブ・ロケールおよびコード・ページ値へのアクセス	220
例: コード・ページ ID の取得および変換	221

第 3 部 プログラムのコンパイル、リンク、実行、デバッグ 223

第 12 章 プログラムのコンパイル、リンク、実行 225

環境変数の設定	225
コンパイラ環境変数	226
ランタイム環境変数	228
例: 環境変数の設定とアクセス	234
コンパイル済みプログラム	235
コマンド行からのコンパイル	236
シェル・スクリプトを使用したコンパイル	237
SFS ファイルのためのコンパイル	238
PROCESS (CBL) ステートメントによるコンパイラ・オプションの指定	238
デフォルトのコンパイラ構成の変更	239
ソース・プログラムのエラーの訂正	242
コンパイル・エラー・メッセージの重大度コード	242
コンパイル・エラー・メッセージのリストの生成	243
cob2、cob2_j、および cob2_r オプション	245
コンパイルに適用されるオプション	245
リンクに適用されるオプション	246
コンパイルとリンクの両方に適用されるオプション	247
プログラムのリンク	247
リンカーへのオプションの引き渡し	248
リンカーの入出力ファイル	251
リンク内のエラーの訂正	253
プログラムの実行	254
COBOL for AIX ライブラリー・コンポーネントの再配布	254

第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行 257

オブジェクト指向アプリケーションのコンパイル	257
オブジェクト指向アプリケーションの準備	258
例: COBOL クラス定義のコンパイルとリンク	259
オブジェクト指向アプリケーションの実行	260
main メソッドで始まるオブジェクト指向アプリケーションの実行	260
COBOL プログラムで始まるオブジェクト指向アプリケーションの実行	261
Java SDK サポート	261

第 14 章 コンパイラ・オプション 263

矛盾するコンパイラ・オプション	265
---------------------------	-----

ADATA	266
ADDR	266
ARITH	268
BINARY	269
CALLINT	270
CHAR	271
CICS	273
COLLSEQ	274
COMPILE	276
CURRENCY	277
DATEPROC	278
DIAGTRUNC	279
DYNAM	280
ENTRYINT	280
EXIT	281
文字ストリング形式	282
ユーザー出口作業域	283
出口モジュールのパラメーター・リスト	283
INEXIT の使用	284
LIBEXIT の使用	284
PRTEXIT の使用	286
ADEXIT の使用	286
FLAG	286
FLAGSTD	288
FLOAT	289
LIB	290
LINECOUNT	291
LIST	291
LSTFILE	292
MAP	293
MAXMEM	294
MDECK	295
NCOLLSEQ	296
NSYMBOL	296
NUMBER	297
OPTIMIZE	298
PGMNAME	299
PGMNAME(UPPER)	300
PGMNAME(MIXED)	301
PROFILE	301
QUOTE/APOST	301
SEPOBJ	302
バッチ・コンパイル	302
SEQUENCE	303
SIZE	304
SOSI	305
SOURCE	306
SPACE	307
SPILL	308
SQL	308
SSRANGE	309
TERMINAL	310
TEST	310
THREAD	311
TRUNC	312
TRUNC の例 1	314

TRUNC の例 2	314
VBREF	315
WSCLEAR	315
XREF	316
YEARWINDOW	317
ZWB	318

第 15 章 コンパイラー指示ステートメント 319

第 16 章 ランタイム・オプション 325

CHECK	325
DEBUG	326
ERRCOUNT	326
FILEMODE	327
FILESYS	328
TRAP	328
UPSI	329

第 17 章 デバッグ 331

ソース言語によるデバッグ	331
プログラム・ロジックのトレース	332
入出力エラーの検出および処理	332
データの妥当性検査	333
初期化されていないデータの検出	333
プロシージャーに関する情報の生成	334
コンパイラー・オプションを使用したデバッグ	336
コーディング・エラーの検出	336
行シーケンス問題の検出	337
有効範囲の検査	337
診断するエラーのレベルの選択	338
プログラム・エンティティ定義および参照の検出	340
データ項目のリスト	341
デバッガーの使用	341
リストの入手	342
例: 短縮リスト	343
例: SOURCE および NUMBER 出力	345
例: MAP 出力	346
例: XREF 出力: データ名相互参照	349
例: VBREF コンパイラー出力	352
オフセット情報を含むメッセージによるデバッグ	353
ユーザー出口のデバッグ	353
アセンブラー・ルーチンのデバッグ	354

第 4 部 データベースへのアクセス 355

第 18 章 DB2 環境用のプログラミング 357

DB2 coprocessor	357
SQL ステートメントのコーディング	358
DB2 coprocessor を用いた SQL INCLUDE の使用	359
SQL ステートメントでのバイナリー項目の使用	359
SQL ステートメントの成否の判断	360
コンパイル前の DB2 の起動	360

SQL オプションを使用したコンパイル	360
DB2 サブオプションの分離	361
パッケージ名およびバインド・ファイル名の使用	361

第 19 章 COBOL プログラムの開発 (CICS の場合) 363

CICS のもとで実行する COBOL プログラムのコーディング	364
CICS のもとでのシステム日付の取得	365
CICS での動的呼び出し	366
SFS データへのアクセス	367
CICS での COBOL および C/C++ 間の呼び出し	367
CICS プログラムのコンパイルおよび実行	368
組み込みの CICS 変換プログラム	369
CICS プログラムのデバッグ	369

第 20 章 Open Database Connectivity (ODBC) 371

ODBC と組み込み SQL の比較	371
バックグラウンド	372
ODBC 対応ソフトウェアのインストールおよび構成	372
COBOL からの ODBC 呼び出しのコーディング: 概要	372
ODBC に適したデータ型の使用	372
ODBC 呼び出しにおける引数としてのポインタの受け渡し	373
ODBC 呼び出しにおける関数戻り値へのアクセス	375
ODBC 呼び出しにおけるビットのテスト	375
ODBC API 用の COBOL コピーブックの使用	376
例: ODBC コピーブックを使用したサンプルプログラム	378
例: ODBC プロシージャ用のコピーブック	379
例: ODBC データ定義用のコピーブック	382
COBOL 用に切り捨てまたは省略される ODBC 名	382
ODBC 呼び出しを行うプログラムのコンパイルおよびリンク	384

第 5 部 XML と COBOL の連携 385

第 21 章 XML 入力の処理 387

COBOL での XML パーサー	388
XML 文書へのアクセス	389
XML 文書の構文解析	389
XML を処理するためのプロシージャの作成	391
XML-EVENT	392
XML-CODE	393
XML-TEXT および XML-NTEXT	393
XML テキストの COBOL データ項目への変換	394
例: XML の処理用プログラム	395
XML 文書のエンコード方式	397
XML 入力文書エンコード	398
コード・ページの指定	399

XML マークアップ内の EBCDIC コード・ページ依存文字	400
UTF-8 でエンコードされた XML 文書の構文解析	400
XML PARSE の例外処理	401
XML パーサーによるエラーの処理方法	403
コード・ページの矛盾の処理	404
XML 構文解析の終了	404

第 22 章 XML 出力の生成 407

XML 出力の生成	407
生成される XML 出力のエンコードの制御	411
XML GENERATE 例外の処理	412
例: XML の生成	413
プログラム XGFX	414
プログラム Pretty	415
プログラム XGFX からの出力	417
XML 出力の拡張	418
例: XML 出力の拡張	419
例: エレメントまたは属性名のハイフンを下線に変換する	421

第 6 部 オブジェクト指向プログラムの開発 425

第 23 章 オブジェクト指向プログラムの作成 427

例: 口座	428
サブクラス	429
クラスの定義	430
クラス定義用の CLASS-ID 段落	432
クラス定義用の REPOSITORY 段落	433
クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION	434
例: クラスの定義	435
クラス・インスタンス・メソッドの定義	436
クラス・インスタンス・メソッド定義用の METHOD-ID 段落	437
クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION	438
クラス・インスタンス・メソッド定義用の DATA DIVISION	438
クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION	439
インスタンス・メソッドのオーバーライド	440
インスタンス・メソッドの多重定義	441
属性 (get および set) メソッドのコーディング	442
例: メソッドの定義	443
クライアントの定義	445
クライアント定義用の REPOSITORY 段落	446
クライアント定義用の DATA DIVISION	447
オブジェクト参照の比較および設定	449
メソッドの呼び出し (INVOKE)	450
クラスのインスタンスの作成および初期化	455
クラスのインスタンスの解放	457
例: クライアントの定義	457

サブクラスの定義	458
サブクラス定義用の CLASS-ID 段落	459
サブクラス定義用の REPOSITORY 段落	460
サブクラス・インスタンス・データ定義用の WORKING-STORAGE SECTION	460
サブクラス・インスタンス・メソッドの定義	461
例: サブクラスの定義 (メソッドに関して)	461
ファクトリー・セクションの定義	462
ファクトリー・データ定義用の WORKING-STORAGE SECTION	463
ファクトリー・メソッドの定義	464
例: ファクトリーの定義 (メソッドに関して)	467
プロシージャ指向 COBOL プログラムのラッピング	473
OO アプリケーションの構造化	473
例: java コマンドを使用して実行される COBOL アプリケーション	474

第 24 章 Java メソッドとの通信 477

JNI サービスへのアクセス	477
Java 例外の処理	479
ローカル参照とグローバル参照の管理	480
Java アクセス制御	482
Java とのデータ共有	482
COBOL および Java での相互運用可能なデータ 型のコーディング	483
Java 用の配列およびストリングの宣言	484
Java 配列の取り扱い	485
Java ストリングの取り扱い	488

第 7 部 複雑なアプリケーションを扱う作業 491

第 25 章 プラットフォーム間でのアプリケーションの移植 493

コンパイルするメインフレーム・アプリケーション の取得	493
実行するメインフレーム・アプリケーションの取得: 概要	494
データ表現による違いの修正	494
移植性に影響する環境の違いの修正	497
言語エレメントによる違いの修正	497
メインフレーム上で実行するコードの作成	498
AIX と Windows 間で移植可能なアプリケーション の作成	499

第 26 章 サブプログラムの使用 501

メインプログラム、サブプログラム、および呼び出し	501
メインプログラムまたはサブプログラムの終了と再 入	502
ネストされた COBOL プログラムの呼び出し	503
ネストされたプログラム	503
例: ネストされたプログラムの構造	505
名前の有効範囲	505

ネストなし COBOL プログラムの呼び出し	506
CALL identifier および CALL literal	507
COBOL および C/C++ プログラム間の呼び出し	508
環境の初期設定	509
COBOL と C/C++ 間でのデータの受け渡し	509
スタック・フレームの縮小と実行単位またはプロ セスの終了	510
COBOL および C/C++ のデータ型	511
例: C 関数を呼び出す COBOL プログラム	511
例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プロ グラム	512
例: C プログラムによって呼び出される COBOL プログラム	513
例: コンパイルおよび実行の結果の例	514
例: C++ 関数を呼び出す COBOL プログラム	514
再帰呼び出しの実行	515
戻りコードの受け渡し	516

第 27 章 データの共有 517

データの受け渡し	517
呼び出し側プログラムの中での引数の記述	519
呼び出し先プログラムの中でのパラメーターの記 述	519
OMITTED 引数に関するテスト	520
LINKAGE SECTION のコーディング	520
引数を受け渡すための PROCEDURE DIVISION の コーディング	521
受け渡されるデータのグループ化	522
ヌル終了ストリングの処理	522
ポインターによるチェーン・リストの処理	523
プロシージャ・ポインターと関数ポインターの使 用	526
戻りコード情報の引き渡し	526
RETURN-CODE 特殊レジスタの理解	527
PROCEDURE DIVISION RETURNING . . . の使 用	527
CALL . . . RETURNING の指定	527
EXTERNAL 節によるデータの共有	527
プログラム間でのファイルの共有 (外部ファイル)	528
例: 外部ファイルの使用	529
コマンド行引数の使用	532
例: -host オプションを使用しないコマンド行引 数	532
例: -host オプションを使用したコマンド行引数	533

第 28 章 共用ライブラリーの構築 535

スタティック・リンクおよび共用ライブラリー	535
共用ライブラリーへの参照をリンカーが解決する方 法	536
例: サンプルの共用ライブラリーの作成	537
例: サンプルの共用ライブラリー用の Make ファ イルの作成	540
例: CALL identifier を使用した共用ライブラ リーの呼び出し	540

例: COBOL および C の共用ライブラリーを呼び出す COBOL プログラム. 542

第 29 章 マルチスレッド化のための

COBOL プログラムの準備 545

マルチスレッド化 545

マルチスレッド化による言語エレメントの処理 . . . 547

実行単位の有効範囲を持つエレメントの処理 . . . 547

プログラム呼び出しインスタンスの有効範囲を持つエレメントの処理 548

マルチスレッド化を使用した COBOL 言語エレメントの有効範囲 548

マルチスレッド化サポートのための THREAD の選択 549

マルチスレッド化されたプログラムへの制御権移動 . . 549

マルチスレッド化されたプログラムの終了 550

マルチスレッド化による COBOL 制限の処理 550

例: マルチスレッド環境での COBOL の使用 551

thrcob.c のソース・コード 551

subd.cbl のソース・コード 553

sube.cbl のソース・コード 553

第 30 章 COBOL ランタイム環境の事前初期設定 555

永続的な COBOL 環境の初期設定 555

事前初期設定された COBOL 環境の終了 556

例: COBOL 環境の事前初期設定 557

第 31 章 2 桁年の日付の処理 561

2000 年言語拡張 (MLE) 562

この拡張の原則と目標 563

日付に関連したロジック問題の解決 564

世紀ウィンドウの使用 565

内部ブリッジングの使用 566

完全フィールド拡張への移行 567

年先行型、年単独型、および年末尾型の日付フィールドの使用 570

互換性のある日付 570

例: 年先行型日付フィールドの比較 571

その他の日付形式の使用 572

例: 年の分離 572

リテラルを日付として操作する 573

仮定による世紀ウィンドウ 574

非日付の処理 575

符号条件の使用 576

日付フィールドに対する算術の実行 576

ウィンドウ表示日付フィールドのオーバーフローの考慮 577

評価の順序の指定 578

日付処理の明示的制御 579

DATEVAL の使用 579

UNDATE の使用 580

例: DATEVAL 580

例: UNDATE 580

日付関連診断メッセージの分析および回避 581

日付処理上の問題の回避 583

パック 10 進数フィールドの問題の回避 583

拡張日付フィールドからウィンドウ表示日付フィールドへの移動 584

第 8 部 パフォーマンスおよび生産

性の向上 585

第 32 章 プログラムのチューニング 587

最適なプログラミング・スタイルの使用 588

構造化プログラミングの使用 588

一括表示表現 588

シンボリック定数の使用 589

定数計算のグループ化 589

重複計算のグループ化 589

効率的なデータ型の選択 590

効率的な計算データ項目の選択 590

一貫性のあるデータ型の使用 591

算術式の効率化 591

指数計算の効率化 591

テーブルの効率的処理 592

テーブル参照の最適化 593

コードの最適化 595

最適化 596

パフォーマンスを向上させるコンパイラー機能の選択 597

パフォーマンスに関連するコンパイラー・オプション 597

パフォーマンスの評価 600

第 33 章 コーディングの単純化 601

反復コーディングの除去 601

例: COPY ステートメントの使用 602

日時の取り扱い 603

日時呼び出し可能サービスからのフィードバックの取得 604

日時の呼び出し可能サービスからの条件の処理 604

例: 日付の操作 604

例: 出力用の日付形式 605

フィードバック・トークン 606

ピクチャー文字項およびストリング 607

例: 日時のピクチャー・ストリング 609

世紀ウィンドウ 610

第 9 部 付録 613

付録 A. 移行および互換性 615

オブジェクト・コード 615

オブジェクト指向構文、および Java 5 または Java

6 SDK 615

データ・サイズ制限 616

XML 生成 616

SFS ファイル 616

32 ビット・アプリケーションの 64 ビット・モードへの移行 617

付録 B. ホスト COBOL との違いの要約 619

コンパイラー・オプション 619
データ表現 620
 ゾーン 10 進数データ 620
 パック 10 進データ 620
 浮動小数点データの表示 620
 国別データ 620
 EBCDIC および ASCII データ 620
 データ変換用のコード・ページの決定 621
 DBCS 文字ストリング 621
環境変数 621
ファイル指定 622
言語間通信 (ILC) 622
入出力 623
ランタイム・オプション 623
ソース・コードの行サイズ 623
言語エレメント 623

付録 C. zSeries ホスト・データ形式についての考慮事項 627

CICS アクセス 627
日時呼び出し可能サービス 627
浮動小数点のオーバーフロー例外 627
DB2 628
Java とのインターオペラビリティのためのオブジェクト指向構文 628
分散コンピューティング環境アプリケーション 628
ファイル・データ 628
SORT 628

付録 D. 中間結果および算術精度 631

中間結果用の用語 632
例: 中間結果の計算 633
固定小数点データと中間結果 633
 加算、減算、乗算、および除算 633
 指数 634
 例: 固定小数点の算術での指数 636
 中間結果での切り捨て 636
 バイナリー・データと中間結果 636
固定小数点算術で評価される組み込み関数 637
 整数関数 637
 混合関数 637
浮動小数点データと中間結果 639
 浮動小数点演算で評価される指数 639
 浮動小数点演算で評価される組み込み関数 640
非算術ステートメントの算術式 640

付録 E. 複合 OCCURS DEPENDING ON 643

例: 複合 ODO 643
 長さの計算方法 644
 ODO オブジェクトの値の設定 644
ODO オブジェクト値の変更の影響 644

ODO オブジェクト値を変更する際の指標エラーを防止する 645
エレメントを可変テーブルに追加する際のオーバーレイを防止する 646

付録 F. 日時呼び出し可能サービス 649

CEECBLDY: 日付から COBOL 整数形式への変換 651
CEEDATE: リリアン日付から文字形式への変換 655
CEEDATM: 秒から文字タイム・スタンプへの変換 659
CEEDAYS: 日付からリリアン形式への変換 663
CEEDYWK: リリアン日付からの曜日の計算 666
CEEGMT: 現在のグリニッジ標準時の取得 669
CEEGMTO: グリニッジ標準時から現地時間までのオフセットの取得 671
CEEISEC: 整数から秒への変換 673
CEELOCT: 現在の現地日時の取得 676
CEEQCEN: 世紀ウィンドウの照会 678
CEESCEN: 世紀ウィンドウの設定 679
CEESECI: 秒から整数への変換 681
CEESECS: タイム・スタンプの秒への変換 685
CEEUTC: 協定世界時の取得 689
IGZEDT4: 現在日付の取得 689

付録 G. XML 参照資料 691

継続を許可する XML PARSE 例外 691
継続を許可しない XML PARSE 例外 697
XML 準拠 700
XML GENERATE 例外 702

付録 H. JNI.cpy 705

付録 I. COBOL SYSADATA ファイルの内容 711

SYSADATA ファイルに影響する既存のコンパイラー・オプション 711
SYSADATA レコード・タイプ 712
例: SYSADATA 713
SYSADATA レコード記述 714
共通ヘッダー・セクション 715
ジョブ識別レコード: X'0000' 717
ADATA 識別レコード: X'0001' 717
コンパイル単位の開始/終了レコード: X'0002' 718
オプション・レコード: X'0010' 718
外部シンボル・レコード: X'0020' 729
構文解析ツリー・レコード: X'0024' 730
トークン・レコード: X'0030' 744
ソース・エラー・レコード: X'0032' 758
ソース・レコード: X'0038' 758
COPY REPLACING レコード: X'0039' 759
記号レコード: X'0042' 759
記号相互参照レコード: X'0044' 772
ネストされたプログラム・レコード: X'0046' 773
ライブラリー・レコード: X'0060' 774
統計レコード: X'0090' 775
EVENTS レコード: X'0120' 775

付録 J. ランタイム・メッセージ	781
特記事項	831
商標	833
用語集	835

資料名リスト	871
COBOL for AIX	871
関連資料	871
索引	873

表

1. FILE SECTION 記入項目	13	36. 被比較数のデータ型および照合シーケンスが比較に与える影響	275
2. プログラム内でのデータ項目の割り当て	27	37. 出口モジュールのパラメーター・リスト	283
3. COMP-5 データ項目の値の範囲	48	38. ランタイム・オプション	325
4. 2 進数値項目の内部表現	49	39. コンパイラー・メッセージの重大度レベル	338
5. 固有数値項目の内部表現	50	40. コンパイラー・オプションとリストの対応	342
6. CHAR(EBCDIC) および FLOAT(HEX) が有効であるときの数値項目の内部表現	51	41. MAP 出力で使用される用語および記号	347
7. 算術演算子の評価の順序	56	42. ODBC C タイプおよび対応する COBOL 宣言	373
8. 数字組み込み関数	57	43. ODBC コピーブック	377
9. STL ファイル・システムの戻りコード	127	44. COBOL 用に切り捨てまたは省略される ODBC 名	382
10. STL ファイル・システムのアダプター・オープン・ルーチン戻りコード	129	45. XML パーサーが使用する特殊レジスター	391
11. ファイル・システム要約	131	46. さまざまな EBCDIC CCSID 用特殊文字の 16 進値	400
12. ファイル編成およびアクセス・モード	133	47. ENCODING 句を省略した場合に生成される XML のエンコード	412
13. 順次ファイルに有効な COBOL ステートメント	141	48. クラス定義の構成	431
14. 行順次ファイルに有効な COBOL ステートメント	142	49. インスタンス・メソッド定義の構成	436
15. 索引付きファイルおよび相対ファイルに有効な COBOL ステートメント	142	50. COBOL クライアントの構成	445
16. ファイルへのレコード書き込み時に使用するステートメント	144	51. COBOL クライアントでの引数の合致	451
17. ファイルの更新に使用する PROCEDURE DIVISION ステートメント	147	52. COBOL クライアントでの戻されるデータ項目の合致	454
18. ソートおよびマージ・エラー番号	165	53. ファクトリー定義の構成	463
19. COBOL ステートメントと国別データ	184	54. ファクトリー・メソッド定義の構成	465
20. 組み込み関数と国別文字データ	187	55. ローカルおよびグローバル参照の JNI サービス	482
21. グループ・セマンティクスで処理される国別グループ項目	196	56. COBOL および Java で相互運用可能なデータ型	483
22. エンコード方式と英数字、DBCS、および国別データのサイズ	196	57. COBOL および Java で相互運用可能な配列およびストリング	484
23. サポートされるロケールおよびコード・ページ	214	58. COBOL および Java で相互運用可能でない配列型	485
24. 照合シーケンスに依存する組み込み関数	220	59. JNI 配列サービス	485
25. TZ 環境パラメーター変数	234	60. jstring 参照と国別データ間の変換サービス	488
26. cob2 コマンドからの出力	237	61. jstring 参照と UTF-8 データ間の変換サービス	489
27. シェル・スクリプト内のコンパイラー・オプション構文の例	238	62. ASCII 文字と EBCDIC 文字との対比	495
28. スタンザ属性	241	63. ASCII での比較と EBCDIC での比較の対比	495
29. コンパイル・エラー・メッセージの重大度コード	243	64. IEEE と 16 進数の対比	496
30. 共通リンカー・オプション	249	65. COBOL および C/C++ のデータ型	511
31. リンカーで想定されるデフォルトのファイル名	253	66. CALL ステートメントでデータを渡す方法	518
32. クラス定義のコンパイルおよびリンクのコマンド	258	67. マルチスレッド化を使用した COBOL 言語エレメントの有効範囲	548
33. JVM をカスタマイズするための Java コマンド・オプション	261	68. 2000 年問題のソリューションの利点および欠点	564
34. コンパイラー・オプション	263	69. パフォーマンスに関連するコンパイラー・オプション	598
35. 相互に排他的なコンパイラー・オプション	265	70. パフォーマンス調整のワークシート	600
		71. ピクチャー文字項およびストリング	607
		72. 日本元号	609

73. 日時のピクチャー・ストリングの例	609	99. SYSADATA オプション・レコード	718
74. Enterprise COBOL for z/OS と COBOL for AIX 間の言語の違い	624	100. SYSADATA 外部シンボル・レコード	729
75. 最大浮動小数点値	627	101. SYSADATA 構文解析ツリー・レコード	730
76. 日時呼び出し可能サービス	649	102. SYSADATA トークン・レコード	744
77. 日時組み込み関数	650	103. SYSADATA ソース・エラー・レコード	758
78. CEECBLDY のシンボリック条件	652	104. SYSADATA ソース・レコード	758
79. CEEDATE のシンボリック条件	656	105. SYSADATA COPY REPLACING レコード	759
80. CEEDATM のシンボリック条件	659	106. SYSADATA 記号レコード	759
81. CEEDAYS のシンボリック条件	664	107. SYSADATA 記号相互参照レコード	772
82. CEEDYWK のシンボリック条件	667	108. SYSADATA ネストされたプログラム・レコー ド	773
83. CEEGMT のシンボリック条件	669	109. SYSADATA ライブラリー・レコード	774
84. CEEGMTO のシンボリック条件	672	110. SYSADATA 統計レコード	775
85. CEEISEC のシンボリック条件	674	111. SYSADATA EVENTS TIMESTAMP レコー ドのレイアウト	776
86. CEEOCT のシンボリック条件	677	112. SYSADATA EVENTS PROCESSOR レコー ドのレイアウト	776
87. CEEQCEN のシンボリック条件	678	113. SYSADATA EVENTS FILE END レコードの レイアウト	776
88. CEESCEN のシンボリック条件	680	114. SYSADATA EVENTS PROGRAM レコー ドのレイアウト	777
89. CEESECI のシンボリック条件	683	115. SYSADATA EVENTS FILE ID レコードのレ イアウト	777
90. CEESECS のシンボリック条件	686	116. SYSADATA EVENTS ERROR レコードのレ イアウト	778
91. 続行可能な XML PARSE 例外	691	117. ランタイム・メッセージ	781
92. 継続を許可しない XML PARSE 例外	697		
93. XML GENERATE 例外	702		
94. SYSADATA レコード・タイプ	712		
95. SYSADATA 共通ヘッダー・セクション	715		
96. SYSADATA ジョブ識別レコード	717		
97. ADATA 識別レコード	718		
98. SYSADATA コンパイル単位の開始終了レコー ド	718		

前書き

本書について

IBM® COBOL for AIX®, IBM の AIX 用 COBOL コンパイラおよびランタイムをご利用いただき、ありがとうございます。

ホスト用の COBOL とワークステーション用の COBOL には多少の違いがあります。COBOL for AIX と Enterprise COBOL for z/OS® の言語およびシステムの違いについては、619 ページの『付録 B. ホスト COBOL との違いの要約』を参照してください。

本書の使い方

本書は、IBM COBOL for AIX プログラムの作成、コンパイル、リンク・エディット、および実行に役立ちます。さらに、オブジェクト指向クラスおよびメソッドの定義、メソッドの呼び出し、およびプログラムでのオブジェクトの参照を行うのに役立ちます。

本書は、アプリケーション・プログラムの開発の経験と、COBOL に関する多少の知識を前提としています。本書では、COBOL 言語の定義ではなく、プログラミングの目的に合った COBOL の使用に重点を置いています。COBOL 構文の詳細については、「COBOL for AIX 言語解説書」を参照してください。

また、本書では AIX の知識がある方を対象としています。AIX について詳しくは、お使いのオペレーティング・システムのマニュアルを参照してください。

略語

本書では、短縮形で使用される用語があります。最も頻繁に使用される製品名の省略形を以下にリストします。

使用される用語	長形式
CICS®	IBM TXSeries®
COBOL for AIX	IBM COBOL for AIX
DB2®	Database 2™
RSD	レコード順次区切りファイル・システム
STL	標準言語ファイル・システム

これらの略語のほかに、本書では「標準 COBOL 85」という用語が使用されています。この用語は、以下の規格の組み合わせを示します。

- ISO 1989:1985、プログラム言語 - COBOL
- ISO/IEC 1989/AMD1:1992、プログラム言語 - COBOL - 組み込み関数モジュール

- ISO/IEC 1989/AMD2:1994、プログラム言語 - COBOL - COBOL 用修正および説明改訂
- ANSI INCITS 23-1985、プログラム言語 - COBOL
- ANSI INCITS 23a-1989、プログラム言語 - COBOL 用組み込み関数モジュール
- ANSI INCITS 23b-1993、プログラム言語 - COBOL 用修正と改訂

ISO 規格は、米国標準規格と一致しています。

その他の用語（一般に理解されていない場合）は、初出時にイタリック体で示され、巻末の用語集にリストされています。

構文図の読み方

本書中の構文図を読むには、以下の説明を参照してください。

- 構文図は、左から右、上から下へと線をたどって読んでください。

>>--- 記号は、構文図の開始を示します。

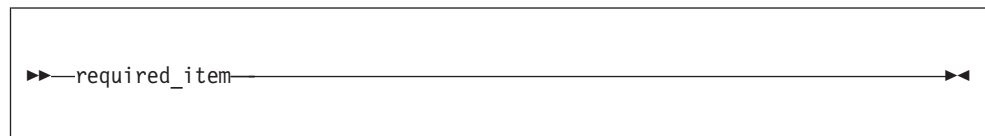
---> 記号は、構文図が次の行に続くことを示します。

>--- 記号は、構文図が前の行から続いていることを示します。

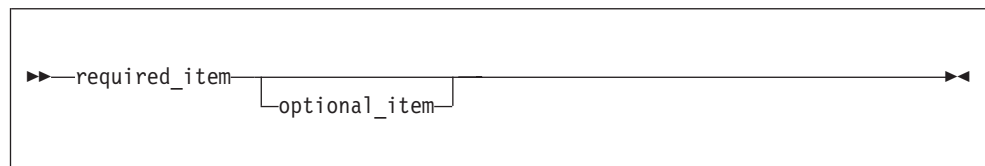
---<< 記号は、構文図の終わりを示します。

完全なステートメントではない構文単位の図は、>--- 記号で始まり、---> 記号で終わります。

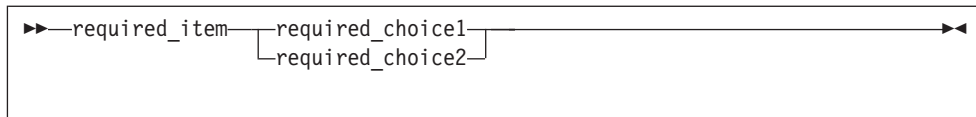
- 必須項目は、水平線（幹線）と同じ高さに示されます。



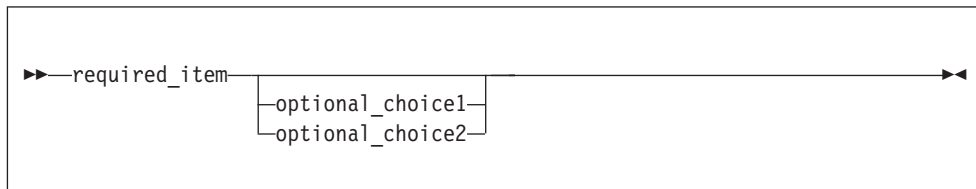
- オプション項目は、幹線の下側に示されます。



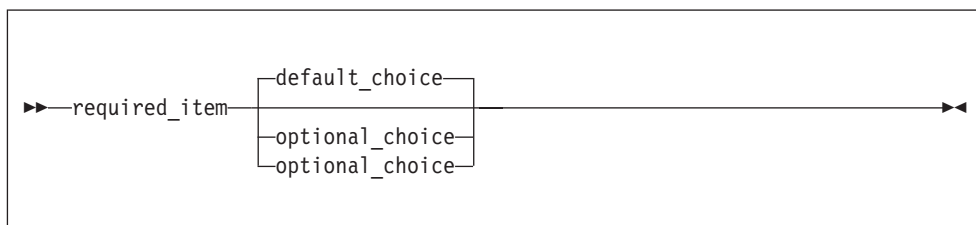
- 複数の項目から選択できる場合には、それらの項目は縦方向に重ねて示されます。それらの項目のうち 1 つを選択しなければならない場合には、それらの項目のうち 1 つが幹線上に示されます。



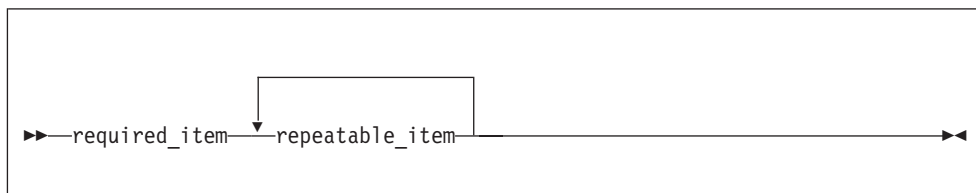
項目の選択が任意である場合には、それらの項目全体が幹線より下に置かれます。



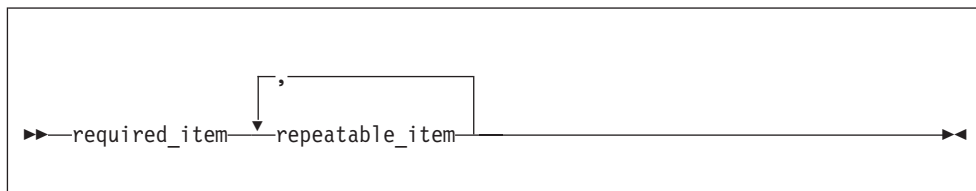
項目のうちの 1 つがデフォルトであれば、それが幹線より上に示され、残りの選択項目は幹線より下に示されます。



- 幹線の上を左に戻する矢印は、その項目を繰り返して指定できることを示しています。



反復矢印にコンマが含まれている場合は、繰り返す項目をコンマで区切って指定する必要があります。



- キーワードは大文字で示されています (例えば、FROM)。キーワードは、示されているとおりに入力しなければなりません。変数は日本語または小文字で示されます (例えば *column-name*)。それらの変数は、ユーザーが指定する名前や値を表します。
- 句読記号、括弧、算術演算子などの記号が示されている場合には、これらも構文の一部として入力しなければなりません。

例の示し方

本書では、コーディング技法を説明するために、サンプル COBOL ステートメント、プログラムの一部分、および短いプログラムの例が多く示されています。プログラム・コードの例は、小文字、大文字、または大/小文字混合で書かれており、これらのいずれでもプログラムを作成できることを示しています。

例を説明テキストと明確に区別する場合は、例はモノスペース・フォントで示されます。

テキスト内の COBOL キーワードとコンパイラー・オプションは、通常は、SMALL UPPERCASE (小さい英大文字フォント) で示されます。プログラム変数名などのようなその他の用語は、明確にするために、イタリック・フォント で示される場合があります。

関連情報

この「プログラミング・ガイド」に記載されている情報は、オンラインの IBM AIX コンパイラーのインフォメーション・センター (publib.boulder.ibm.com/infocenter/comphelp/index.jsp) でもご覧いただけます。インフォメーション・センターには、COBOL 言語に関する参照情報とデバッガーを使用するための情報が、すべて揃っています。また、オンラインでも本書内でも、リソース・リストをチェックすることができます。

改訂の要約

ここでは、バージョン 3 で COBOL for AIX に加えられた主要な変更を示します。本書で解説されている変更には、読者の便宜のため、参照ページが記載されています。COBOL for AIX バージョン 2 に関する変更 (技術的な変更と用語に対する編集上の変更の両方) については、PDF 版では左マージンに縦線 (|) でマークが付けられています。

バージョン 3.1 (2009 年 1 月)

- 現在、64 ビット・アプリケーションを作成できます。64 ビットの活用をサポートする機能が追加されました。
 - 新規コンパイラー・オプション ADDR を使用すれば、32 ビット・オブジェクト・プログラムと 64 ビット・オブジェクト・プログラムのどちらを生成するのかを指示できます (『266 ページの『ADDR』』)。

- COBOL データ項目サイズのいくつかの限界値が大幅に引き上げられました。例えば、LOCAL-STORAGE、WORKING-STORAGE、および LINKAGE の各セクションの最大サイズは、ADDR(64) が有効の場合は実質的に無制限 (マシンのアドレッシング容量まで) となりました。

こうしたサポートにより、大量データに関連したプログラミングが容易になります。例として次のようなものがあります。

- DB2 BLOB および CLOB データ・タイプ を使用する DB2/COBOL アプリケーション
- 大きな XML 文書を構文解析または生成する COBOL XML アプリケーション

増やされたコンパイラ限界値の詳細については、コンパイラ限界値 (「COBOL for AIX 言語解説書」) を参照してください。

- 2 GB より大きいファイルのサポート: バージョン 3 のプログラムによって作成された STL ファイルと VSAM ファイルはデフォルトで、最大サイズ 2 GB の制限がなくなった新規の大容量形式メタデータ表現 (内部制御情報) を持ちます。これらのファイルは、バージョン 3 のランタイム・ライブラリーで実行されるバージョン 2 のプログラムでも処理できます (『130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』』)。
- 新規ランタイム・オプション FILEMODE を使用すれば、新たに作成する STL ファイルと VSAM ファイルが新規の大容量形式メタデータ表現を持つのか、または新たに作成する STL ファイルと VSAM ファイルが小容量形式メタデータ表現を持ち、バージョン 2 のプログラムによって作成された STL ファイルおよび VSAM ファイルと互換性のある 2 GB 最大サイズが使用されるのかを指定できます (『327 ページの『FILEMODE』』)。
- 国別 (Unicode UTF-16) データのサポートが拡張されました。いくつかの種類のデータ項目がさらに暗黙的または明示的に USAGE NATIONAL として記述できるようになりました。
 - 外部 10 進数 (国別 10 進数) 項目 (41 ページの『数値データの定義』)
 - 外部浮動小数点 (国別浮動小数点) 項目 (45 ページの『数値データの形式』)
 - 数字編集項目 (43 ページの『数値データの表示』)
 - 国別編集項目 (189 ページの『COBOL での国別データ (Unicode) の使用』)
 - GROUP-USAGE NATIONAL 節によってサポートされるグループ (国別グループ) 項目 (194 ページの『国別グループの使用』)
- 数多くの COBOL 言語エレメントが、新しい種類の UTF-16 データをサポートするか、または国別データの処理を新たにサポートします。
 - USAGE NATIONAL (国別 10 進数および国別浮動小数点) が指定された数値データは、算術演算で、また数値オペランドをサポートする任意の言語構造体で使用できます (45 ページの『数値データの形式』)。
 - USAGE NATIONAL が指定された編集データは、移動に関連した編集操作および編集解除操作を含め、既存の編集タイプと同じ言語構造体でサポートされます (43 ページの『数値データの表示』 および 189 ページの『COBOL での国別データ (Unicode) の使用』)。
 - すべての国別データを含むグループ項目は、GROUP-USAGE NATIONAL 節で定義できます。これにより、グループは、ほとんどの言語構造体で基本項目として

動作するようになります。このサポートにより、STRING、UNSTRING、および INSPECT などのステートメントでの国別グループの使用が容易になります (192 ページの『国別グループ』)。

- XML GENERATE ステートメントは、国別グループを受信データ項目としてサポートし、国別編集、USAGE NATIONAL の数字編集、国別 10 進数、国別浮動小数点、および国別グループの各項目を送信データ項目としてサポートします (407 ページの『XML 出力の生成』)。
- NUMVAL および NUMVAL-C 組み込み関数は、国別リテラルまたは国別データ項目を引数として取ることができます (113 ページの『数値への変換 (NUMVAL、NUMVAL-C)』)。

これらの新規の国別データ機能を使用することにより、Unicode のみをすべてのアプリケーション・データで使用する COBOL プログラムの作成が、実用的なものになりました。

- XML GENERATE ステートメントが強化されました (407 ページの『第 22 章 XML 出力の生成』)。
 - NAMESPACE 句を使用して名前空間を、NAMESPACE-PREFIX 句を使用して各エレメントに適用される名前空間接頭部を指定できます。
 - ENCODING 句を使用して、生成された文書のエンコードを指定できます (411 ページの『生成される XML 出力のエンコードの制御』)。
 - XML 文書が、UTF-16 や、さまざまな ASCII コード・ページ/EBCDIC コード・ページに加えて UTF-8 でも生成できるようになりました (397 ページの『XML 文書のエンコード方式』)。
 - WITH ATTRIBUTES 句を使用すると、適格な基本項目が、生成された XML 内で子エレメントとしてではなく属性として表されます。
 - WITH XML-DECLARATION 句を使用すると、XML 宣言が生成されます。
- XML PARSE ステートメントを使用するとき、UTF-16 や、さまざまな ASCII コード・ページ/EBCDIC コード・ページでエンコードされた文書 (397 ページの『XML 文書のエンコード方式』) だけでなく、UTF-8 でエンコードされた文書 (400 ページの『UTF-8 でエンコードされた XML 文書の構文解析』) も構文解析できるようになりました。
- 現在、コンパイラー・リストは、コピーブックの取得元となるライブラリー名とファイル名に対してそのコピーブックを相互参照するようになっています (『350 ページの『例: XREF 出力: COPY/BASIS 相互参照』』)。
- 統合 DB2 コプロセッサ (SQL コンパイラー・オプション) の使用時に、DB2 V9 の新機能がサポートされるようになりました。
- 統合 DB2 コプロセッサの使用時に、COBOL-DB2 アプリケーションに対するユーザビリティ機能拡張が使用可能になりました。例えば、LOCAL-STORAGE SECTION または WORKING-STORAGE SECTION を明示的にコーディングする必要がなくなりました。
- 個別の変換ステップなしで、CICS ステートメントを含むプログラムおよびコピーブックをコンパイルできるようになりました (『369 ページの『組み込みの CICS 変換プログラム』』)。

- 新規コンパイラー・オプション CICS を使用すれば、統合 CICS 変換プログラムを利用したり、CICS オプションを指定したりできます (『273 ページの『CICS』』)。
- デバッグが機能拡張され、Debugger for AIX をサポートするようになりました。
- 新規の呼び出し可能サービス iwzGetSortErrno を使用すれば、ソート操作やマージ操作が行われるたびに、ソートやマージのエラー番号を取得できます (『164 ページの『ソートまたはマージの成否の判断』』)。
- REDEFINES 節は拡張されて、レベル 01 ではないデータ項目の場合、再定義するデータ項目よりも項目のサブジェクトを大きくすることができるようになりました。
- 新規コンパイラー・オプション MDECK により、ライブラリー処理ステートメントからの出力がファイルに書き込まれます (295 ページの『MDECK』)。
- クラス国別のデータ項目用の VALUE 節のリテラルは、英数字にすることができます (74 ページの『グループ・レベルでのテーブルの初期化』)。

このリリースでは、以下の用語の変更も行われています。

- 英数字グループ という用語が導入されて、国別グループ以外のグループを特に指しています。
- グループ という用語は、明らかに英数字グループのみまたは国別グループのみを指しているコンテキストで使用されている場合を除き、英数字グループと国別グループの両方を意味します。
- 外部 10 進数 という用語は、ゾーン 10 進数項目と国別 10 進数項目の両方を指しています。
- 表示浮動小数点 という用語が導入されて、USAGE DISPLAY を持つ外部浮動小数点項目を指すようになっています。
- 外部浮動小数点 という用語は、表示浮動小数点項目と国別浮動小数点項目の両方を指します。

COBOL for AIX のアクセシビリティ

アクセシビリティ機能は、運動障害や視覚障害などの障害を持つユーザーが情報技術製品を快適に使用できるようにサポートします。

アクセシビリティ機能

COBOL for AIX の主要なアクセシビリティ機能:

- スクリーン・リーダーで一般的に使用されるインターフェース
- AT[®] および端末エミュレーション・ソフトウェアを使用する、直接接続またはネットワーク接続されたリモートの Windows[®] または Linux[®] システムを介した、アクセス可能コマンド行インターフェース

AIX コンパイラーのインフォメーション・センターとその関連資料もアクセシビリティに対応しています。インフォメーション・センターのアクセシビリティ機能については、IBM AIX コンパイラーのインフォメーション・センター (www.ibm.com/software/awdtools/cobol/aix/library) を参照してください。

キーボード・ナビゲーション

AIX インターフェースへのアクセスについては、IBM AIX コンパイラーのインフォメーション・センター (www.ibm.com/software/awdtools/cobol/aix/library) を参照してください。

インターフェース情報

支援技術製品は、AIX のユーザー・インターフェースと連動します。特定のガイダンス情報については、AIX インターフェースへのアクセスに使用する支援技術製品の資料を参照してください。

アクセシビリティ関連情報

IBM AIX Compiler Information Center (www.ibm.com/software/awdtools/cobol/aix/library) で提供される英語 XHTML フォーマットのこの文書は、スクリーン・リーダーを使用する視覚障害者の方がご利用になれます。

スクリーン・リーダーが構文図やソース・コード例を正確に読み上げたり、ピリオドやコンマのピクチャー記号を含むテキストを正確に読み上げたりできるようにするには、スクリーン・リーダーがすべての句読文字を読み上げるように設定する必要があります。

IBM およびアクセシビリティ

IBM がアクセシビリティに関して行っている内容について詳しくは、「IBM Human Ability and Accessibility Center」(www.ibm.com/able) を参照してください。

第 1 部 プログラムのコーディング

第 1 章 プログラムの構造	5
プログラムの識別	5
プログラムを再帰的として識別する	6
収容プログラムによってプログラムに呼び出し可能のマークを付ける	6
プログラムを初期状態に設定する	6
ソース・リストのヘッダーの変更	7
コンピューター環境の記述	7
例: FILE-CONTROL 段落	8
照合シーケンスの指定	8
例: 照合シーケンスの指定	9
シンボリック文字を定義する	10
ユーザー定義のクラスを定義する	10
オペレーティング・システムに対してファイルを識別する	10
実行時の入出力ファイルの変更	11
データの記述	12
入出力操作でのデータの使用	12
FILE SECTION 記入項目	13
WORKING-STORAGE と LOCAL-STORAGE の比較	14
例: ストレージ・セクション	15
別のプログラムからのデータの使用	16
別々にコンパイルされたプログラムでのデータの共用	16
ネストされたプログラムでのデータの共用	17
再帰的またはマルチスレッド化されたプログラムでのデータの共用	17
データの処理	17
PROCEDURE DIVISION 内でロジックが分割される方法	18
命令ステートメント	19
条件ステートメント	19
コンパイラ指示ステートメント	21
範囲終了符号	21
宣言	22
第 2 章 データの使用	23
変数、構造、リテラル、および定数の使用	23
変数の使用	23
データ項目とグループ項目の使用	24
リテラルの使用	26
定数の使用	26
形象定数の使用	27
データ項目への値の割り当て	27
例: データ項目の初期化	28
構造の初期化 (INITIALIZE)	31
基本データ項目への値の割り当て (MOVE)	33
グループ・データ項目への値の割り当て (MOVE)	34
算術結果の割り当て (MOVE または COMPUTE)	35
画面またはファイルからの入力の割り当て (ACCEPT)	35
画面上またはファイル内での値の表示 (DISPLAY)	37
組み込み関数の使用 (組み込み関数)	37
テーブル (配列) とポインターの使用	38
第 3 章 数値および算術演算	41
数値データの定義	41
数値データの表示	43
数値データの保管方法の制御	44
数値データの形式	45
外部 10 進数 (DISPLAY および NATIONAL) 項目	46
外部浮動小数点 (DISPLAY および NATIONAL) 項目	46
2 進数 (COMP) 項目	47
固有 2 進数 (COMP-5) 項目	47
パック 10 進数 (COMP-3) 項目	48
内部浮動小数点 (COMP-1 および COMP-2) 項目	48
例: 数値データおよび内部表現	49
データ形式の変換	51
変換および精度	52
精度が低下する変換	52
精度を保つ変換	52
丸めを生じさせる変換	53
ゾーンおよびパック 10 進数データのサイン表記	53
非互換データの検査 (数値のクラス・テスト)	54
算術の実行	54
COMPUTE およびその他の算術ステートメントの使用	54
算術式の使用	55
数字組み込み関数の使用	56
例: 数字組み込み関数	57
一般数値処理	58
日付および時刻	58
金融	58
数学	59
統計	59
固定小数点演算と浮動小数点演算の対比	60
浮動小数点計算	60
固定小数点計算	61
算術比較 (比較条件)	61
例: 固定小数点計算および浮動小数点計算	61
通貨記号の使用	62
例: 複数の通貨符号	63
第 4 章 テーブルの処理	65
テーブルの定義 (OCCURS)	65
テーブルのネスト	67
例: 添え字付け	68
例: 指標付け	68

テーブル内の項目の参照	69	例: 参照修飾子としての組み込み関数	109
添え字付け	69	データ項目の計算および置換 (INSPECT)	110
索引付け	70	例: INSPECT ステートメント	110
テーブルに値を入れる方法	71	データ項目の変換 (組み込み関数)	111
テーブルの動的なロード	72	大文字または小文字への変換	
テーブルの初期化 (INITIALIZE)	72	(UPPER-CASE、LOWER-CASE)	112
テーブルの定義時の値の割り当て (VALUE).	73	逆順への変換 (REVERSE)	112
それぞれのテーブル項目の個別の初期化	74	数値への変換 (NUMVAL、NUMVAL-C)	113
グループ・レベルでのテーブルの初期化	74	あるコード・ページから別のコード・ページへ	
ある特定テーブル・エレメントのすべての出現		変換	114
の初期化	75	データ項目の評価 (組み込み関数)	114
例: PERFORM と添え字付け	75	照合シーケンスに関する単一文字の評価	115
例: PERFORM および索引付け	76	最大または最小データ項目の検出	115
可変長テーブルの作成 (DEPENDING ON)	77	英数字または国別関数によって戻される可変	
可変長テーブルのロード	79	長結果	116
可変長テーブルへの値の割り当て	80	データ項目の長さの検出	118
テーブルの探索	80	コンパイルの日付の検出	118
逐次探索 (SEARCH)	81		
例: 逐次探索	81	第 7 章 ファイルの処理	121
二分探索 (SEARCH ALL).	82	ファイルの識別	121
例: 二分探索	83	VSAM ファイルの識別	122
組み込み関数を使用したテーブル項目の処理	83	Encina SFS ファイルの識別	122
例: 組み込み関数を使用したテーブルの処理	84	STL ファイルの識別	123
		RSD ファイルの識別	123
第 5 章 プログラム・アクションの選択と反復	85	ファイル・システム	124
プログラム・アクションの選択	85	VSAM ファイル・システム	125
アクションの選択項目のコーディング	85	SFS ファイル・システム	125
ネストされた IF ステートメントの使用	86	STL ファイル・システム	127
EVALUATE ステートメントの使用	87	STL ファイル・システムの戻りコード	127
条件式のコーディング	90	RSD ファイル・システム	129
スイッチおよびフラグ	91	2 GB を超える STL ファイルおよび VSAM フ	
スイッチおよびフラグの定義	91	ァイル	130
例: スイッチ	92	ファイル・システム要約	131
例: フラグ	92	ファイル編成およびアクセス・モードの指定	132
スイッチとフラグのリセット	92	ファイル編成およびアクセス・モード	132
例: スイッチをオンに設定する	93	順次ファイルの編成	133
例: スイッチをオフに設定する	93	行順次ファイル編成	134
プログラム・アクションの繰り返し	94	索引付きファイル編成	134
インラインまたはライン外 PERFORM の選択	94	相対ファイル編成	134
例: インライン PERFORM ステートメント	95	順次アクセス	135
ループのコーディング	95	ランダム・アクセス	135
テーブルのループ処理	96	動的アクセス	135
複数の段落またはセクションの実行	97	ファイル・オープン時のエラーからの保護	136
		ファイル状況フィールドの設定	136
第 6 章 スtringの処理	99	ファイル構造の詳細記述	137
データ項目の結合 (STRING).	99	ファイルの入出カステートメントのコーディング	137
例: STRING ステートメント	100	例: COBOL でのファイルのコーディング	138
STRING の結果	101	ファイル位置標識	140
データ項目の分割 (UNSTRING)	102	ファイルのオープン	140
例: UNSTRING ステートメント	103	順次ファイルに有効な COBOL ステートメン	
UNSTRING の結果	104	ト	141
ヌル終了Stringの取り扱い	105	行順次ファイルに有効な COBOL ステートメ	
例: ヌル終了String	106	ント	141
データ項目のサブStringの参照	106	索引付きファイルおよび相対ファイルに有効	
参照修飾子	108	な COBOL ステートメント	142
例: 参照修飾子としての演算式	109	ファイルからのレコードの読み取り	143

ファイルへのレコード書き込み時に使用するステートメント	144
ファイルへのレコードの追加	145
ファイル内のレコードの置換	145
ファイルからのレコードの削除	146
ファイルの更新に使用する PROCEDURE DIVISION ステートメント	147
COBOL アプリケーションで SFS ファイルにアクセスできるようにする	148
例: SFS ファイルへのアクセス	149
COBOL ファイル記述	149
sfsadmin コマンド	150
export コマンド	151
SFS パフォーマンスの向上	151
クライアント側のキャッシュの使用可能化	151
変更の保存回数の削減	153
第 8 章 ファイルのソートおよびマージ	155
ソートおよびマージ・プロセス	156
ソートまたはマージ・ファイルの記述	156
ソートまたはマージへの入力の記述	157
例: SORT 用のソート・ファイルおよび入力ファイルの記述	157
入力プロシージャのコーディング	158
ソートまたはマージからの出力の記述	159
出力プロシージャのコーディング	160
入出力プロシージャに関する制約事項	160
ソートまたはマージの要求	161
ソートまたはマージ基準の設定	162
代替照合シーケンスの選択	163
例: 入出力プロシージャを使用したソート	163
ソートまたはマージの成否の判断	164
ソートおよびマージ・エラー番号	165
ソートまたはマージ操作の途中停止	168
第 9 章 エラーの処理	169
ストリングの結合および分割におけるエラーの処理	169
算術演算でのエラーの処理	170
例: 0 による除算の検査	170
入出力操作でのエラーの処理	171
ファイルの終わり条件 (AT END) の使用	172
ERROR 宣言のコーディング	173
ファイル状況キーの使用	174
例: ファイル状況キー	175
ファイル・システム状況コードの使用	175
例: ファイル・システム状況コードの検査	176
INVALID KEY 句のコーディング	177
例: FILE STATUS および INVALID KEY	178
プログラム呼び出し時のエラーの処理	178

第 1 章 プログラムの構造

COBOL プログラムは、4 つの DIVISION (IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、DATA DIVISION、および PROCEDURE DIVISION) から成ります。それぞれの DIVISION には、固有の論理関数があります。

プログラムを定義するには、IDENTIFICATION DIVISION のみが必要です。

COBOL クラスまたはメソッドを定義するには、プログラムの場合とは違った方法でいくつかの部を定義することが必要です。

関連タスク

『プログラムの識別』

7 ページの『コンピューター環境の記述』

12 ページの『データの記述』

17 ページの『データの処理』

430 ページの『クラスの定義』

436 ページの『クラス・インスタンス・メソッドの定義』

473 ページの『OO アプリケーションの構造化』

プログラムの識別

IDENTIFICATION DIVISION は、プログラムの名前を指定し、また必要があればその他の識別情報を与えるために使用されます。

オプションの AUTHOR、INSTALLATION、DATE-WRITTEN、および DATE-COMPILED 段落を使用して、プログラムに関する記述情報を指定することができます。

DATE-COMPILED 段落に入力したデータは、最新のコンパイル日付で置き換えられます。

```
IDENTIFICATION DIVISION.  
Program-ID. Helloprog.  
Author. A. Programmer.  
Installation. Computing Laboratories.  
Date-Written. 12/07/2008.  
Date-Compiled. 12/14/2008.
```

PROGRAM-ID 段落を使用して、プログラムの名前を指定します。割り当てるプログラム名は、以下のように使用されます。

- プログラムを呼び出すために他のプログラムがその名前を使用します。
- プログラムのコンパイル時に生成されるプログラム・リストの各ページ (最初のページを除く) のヘッダーにその名前が入れます。

ヒント: プログラム名に大/小文字の区別がある場合は、コンパイラーの探索対象である名前とのミスマッチが起こらないようにしてください。PGMNAME コンパイラー・オプションでの該当する設定が有効であるか検査してください。

関連タスク

7 ページの『ソース・リストのヘッダーの変更』

『プログラムを再帰的として識別する』
『収容プログラムによってプログラムに呼び出し可能のマークを付ける』
『プログラムを初期状態に設定する』

関連参照

コンパイラ限界値 (COBOL for AIX 言語解説書)
プログラム名の規則 (COBOL for AIX 言語解説書)

プログラムを再帰的として識別する

前の呼び出しがまだアクティブである間にプログラムに再帰的に再入できるようにするには、PROGRAM-ID 節に RECURSIVE 属性をコーディングしてください。

RECURSIVE は、コンパイル単位の最外部のプログラムにのみコーディングすることができます。ネストされたサブプログラムも、ネストされたサブプログラムを含むプログラムも、再帰的にすることはできません。

関連タスク

17 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共用』
515 ページの『再帰呼び出しの実行』

収容プログラムによってプログラムに呼び出し可能のマークを付ける

収容プログラムまたは収容プログラム内の任意のプログラムによってプログラムを呼び出せることを指定するには、PROGRAM-ID 段落で COMMON 属性を使用してください。ただし、COMMON プログラムは、それ自体に含まれているプログラムによって呼び出すことはできません。

含まれているプログラムだけが COMMON 属性を持つことができます。

関連概念

503 ページの『ネストされたプログラム』

プログラムを初期状態に設定する

プログラムを呼び出すたびにプログラムおよびそのプログラムに含まれるネストされたプログラムを初期状態にすることを指定するには、INITIAL 属性を使用します。

プログラムが初期状態になるのは、以下のとおりです。

- VALUE 節を持つデータ項目が、指定された値に設定された。
- 変更された GO TO ステートメントおよび PERFORM ステートメントが、それぞれ初期状態になった。
- 非 EXTERNAL ファイルがクローズされた。

関連タスク

502 ページの『メインプログラムまたはサブプログラムの終了と再入』

ソース・リストのヘッダーの変更

ソース・リストの最初のページのヘッダーには、コンパイラおよび現行リリース・レベルの識別、コンパイルの日時、およびページ番号が入っています。

以下の例はこれら 5 つのエLEMENTを示しています。

```
PP 5724-V62 IBM COBOL for AIX 3.1.0      Date 12/14/2008   Time 15:05:19   Page    1
```

ヘッダーは、コンパイル・プラットフォームを示します。コンパイラ指示 TITLE ステートメントを使用すれば、リストの後続のページのヘッダーをカスタマイズすることができます。

関連参照

TITLE ステートメント (*COBOL for AIX 言語解説書*)

コンピューター環境の記述

プログラムの ENVIRONMENT DIVISION では、コンピューター環境に依存するプログラムの局面について記述します。

CONFIGURATION SECTION を使用して、次の項目を指定します。

- プログラムをコンパイルするコンピューター (SOURCE-COMPUTER 段落)
- プログラムを実行するコンピューター (OBJECT-COMPUTER 段落)
- 通貨記号やシンボリック文字などの特殊な項目 (SPECIAL-NAMES 段落)
- ユーザー定義のクラス (REPOSITORY 段落)

INPUT-OUTPUT SECTION の FILE-CONTROL および I-O-CONTROL 段落は、以下の目的に使用します。

- プログラム内のファイルの特性を識別および記述する。
- ファイルを対応するシステム・ファイル名と直接的または間接的に関連付けます。
- オプションとして、ファウルに関連したファイル・システムを指定します (例: VSAM または STL ファイル・システム)。これは、実行時にも行えます。
- ファイルのアクセス方法に関する情報を提供します。

8 ページの『例: FILE-CONTROL 段落』

関連タスク

8 ページの『照合シーケンスの指定』

10 ページの『シンボリック文字を定義する』

10 ページの『ユーザー定義のクラスを定義する』

10 ページの『オペレーティング・システムに対してファイルを識別する』

関連参照

セクションおよび段落 (*COBOL for AIX 言語解説書*)

例: FILE-CONTROL 段落

次の例は、FILE-CONTROL 段落を使用して、COBOL プログラム内の各ファイルと、ファイル・システムに既知の物理ファイルに関連付ける方法を示しています。この例は、VSAM 索引付きファイル用の FILE-CONTROL 段落を示しています。

```
SELECT COMMUTER-FILE (1)
  ASSIGN TO COMMUTER (2)
  ORGANIZATION IS INDEXED (3)
  ACCESS IS RANDOM (4)
  RECORD KEY IS COMMUTER-KEY (5)
  FILE STATUS IS (5)
  COMMUTER-FILE-STATUS
  COMMUTER-STL-STATUS.
```

- (1) SELECT 文節は、COBOL プログラム内のファイルに対応するシステム・ファイルと関連付けます。
- (2) ASSIGN 文節は、プログラム内のファイル名を、システムに既知のファイル名と関連付けます。COMMUTER は、システム・ファイル名または環境変数名を表す場合があります。この値は、(実行時に) オプションのディレクトリーおよびパス名とともに、システム・ファイル名として使用されます。
- (3) ORGANIZATION 節は、ファイルの編成を記述します。これを省略した場合は、デフォルトの ORGANIZATION IS SEQUENTIAL が使用されます。
- (4) ACCESS MODE 文節は、ファイル内のレコードを処理できるようにする方式(順次、ランダム、または動的)を定義します。この文節を省略した場合は、ACCESS IS SEQUENTIAL が定義されたものと見なされます。
- (5) 使用するファイルおよびファイル・システムのタイプによって、FILE-CONTROL 段落に追加のステートメントを指定することができます。

関連タスク

7 ページの『コンピューター環境の記述』

照合シーケンスの指定

PROGRAM COLLATING SEQUENCE 節、および SPECIAL-NAMES 段落の ALPHABET 節を使用すれば、英数字項目に対するいくつかの操作で使用される照合シーケンスを設定できます。

これらの節では、英数字項目に対する以下の操作の照合シーケンスを指定します。

- 比較条件および条件名条件で明示的に指定された比較
- HIGH-VALUE および LOW-VALUE の設定
- SEARCH ALL
- SORT および MERGE (SORT または MERGE ステートメントの COLLATING SEQUENCE 句でオーバーライドされていない場合)

9 ページの『例: 照合シーケンスの指定』

以下のいずれかのアルフベットを基に、使用するシーケンスを選択できます。

- EBCDIC: EBCDIC 文字セットに関連付けられた照合シーケンスを参照します。

- **NATIVE:** ロケールの設定で指定された照合シーケンスを参照します。ロケール設定は、コンパイル時に有効化された各国語ロケール名を参照します。通常、インストール時に設定されます。
- **STANDARD-1:** *ANSI INCITS X3.4, Coded Character Sets - 7-bit American National Standard Code for Information Interchange (7-bit ASCII)* により定義された ASCII 文字セットに関連付けられた照合シーケンスを参照します。
- **STANDARD-2:** *ISO/IEC 646 -- Information technology -- ISO 7-bit coded character set for information interchange, International Reference Version* により定義されたコード化文字セットに関連付けられた照合シーケンスを参照します。
- **SPECIAL-NAMES** 段落で定義された ASCII シーケンスの代替。

自分で定義した照合シーケンスを指定することもできます。

制約事項: コード・ページが DBCS、拡張 UNIX[®] コード (EUC)、または UTF-8 の場合、ALPHABET 節を使用することはできません。

PROGRAM COLLATING SEQUENCE 節は国別または DBCS オペランドを含む比較に影響を及ぼしません。

関連タスク

- 163 ページの『代替照合シーケンスの選択』
- 216 ページの『ロケール付きの照合シーケンスの制御』
- 209 ページの『第 11 章 ロケールの設定』
- 202 ページの『国別 (UTF-16) データの比較』

例: 照合シーケンスの指定

次の例は、比較およびソート/マージの場合に大文字と小文字が同様に処理される照合シーケンスを指定する際に使用できる ENVIRONMENT DIVISION コーディングを示しています。

SPECIAL-NAMES 段落の ASCII シーケンスを変更すると、SPECIAL-NAMES 段落に含まれている文字の照合シーケンスだけでなく、全体の照合シーケンスが影響を受けます。

```
IDENTIFICATION DIVISION.
. . .
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
  Object-Computer.
    Program Collating Sequence Special-Sequence.
  Special-Names.
    Alphabet Special-Sequence Is
      "A" Also "a"
      "B" Also "b"
      "C" Also "c"
      "D" Also "d"
      "E" Also "e"
      "F" Also "f"
      "G" Also "g"
      "H" Also "h"
      "I" Also "i"
      "J" Also "j"
      "K" Also "k"
      "L" Also "l"
      "M" Also "m"
```

```
"N" Also "n"  
"O" Also "o"  
"P" Also "p"  
"Q" Also "q"  
"R" Also "r"  
"S" Also "s"  
"T" Also "t"  
"U" Also "u"  
"V" Also "v"  
"W" Also "w"  
"X" Also "x"  
"Y" Also "y"  
"Z" Also "z".
```

関連タスク

8 ページの『照合シーケンスの指定』

シンボリック文字を定義する

SYMBOLIC CHARACTERS 節を使用すると、指定したアルファベットの任意の文字に記号名を与えることができます。序数位置を使用して、文字を識別します。位置 1 は、文字 X'00' に対応します。

例えば、プラス文字 (ASCII アルファベットで、X'2B') に名前を与えるには、次のようにコーディングします。

```
SYMBOLIC CHARACTERS PLUS IS 44
```

ロケールによって指定されたコード・ページがマルチバイト文字のコード・ページである場合には、SYMBOLIC CHARACTERS 節を使用することはできません。

関連タスク

209 ページの『第 11 章 ロケールの設定』

ユーザー定義のクラスを定義する

CLASS 節は、節内にリストした文字のセットに名前を与えるために使用します。

例えば、次の節をコーディングして、数字のセットに名前を与えます。

```
CLASS DIGIT IS "0" THROUGH "9"
```

クラス名は、クラス条件でのみ参照できます。(このユーザー定義クラスは、オブジェクト指向クラスと同じ概念ではありません。)

ロケールによって指定されたコード・ページがマルチバイト文字のコード・ページである場合には、CLASS 節を使用することはできません。

オペレーティング・システムに対してファイルを識別する

ASSIGN 文節は、プログラム内のファイル名を、オペレーティング・システムに既知のファイル名と関連付けます。

ASSIGN 文節には、環境変数、システム・ファイル名、リテラル、データ名のいずれかを使用することができます。環境変数を割り当て名として指定した場合は、実行

時に環境変数が評価され、その値が (オプションのディレクトリーおよびパス名とともに) システム・ファイル名として使用されます。

デフォルト以外のファイル・システムを使用する場合は、システム・ファイル名の前にファイル・システム ID を指定するなどの方法で、ファイル・システムを明示的に示す必要があります。例えば、MYFILE が STL ファイルで、プログラム内でファイル名として F1 を使用する場合、ASSIGN 文節は次のようになります。

```
SELECT F1 ASSIGN TO STL-MYFILE
```

MYFILE が環境変数ではない場合、前述のコードは MYFILE をシステム・ファイル名として扱います。MYFILE が環境変数の場合は、その環境変数の値が使用されます。例えば、環境変数 MYFILE が MYFILE=STL-YOURFILE として設定されている場合、システム・ファイル名は実行時に YOURFILE となり、ファイルは VSAM ファイルとして扱われ、ASSIGN 文節で使用されているファイル・システム ID はオーバーライドされます。

ただし、割り当て名が引用符または一重引用符で囲まれている場合は (例: "STL-MYFILE")、環境変数に値が指定されていても無視されます。この場合、割り当て名はリテラルとして扱われます。

関連タスク

『実行時の入出力ファイルの変更』

121 ページの『ファイルの識別』

関連参照

328 ページの『FILESYS』

ASSIGN 節 (COBOL for AIX 言語解説書)

実行時の入出力ファイルの変更

SELECT 節でコーディングした *file-name* は、COBOL プログラム全体にわたって定数として使用されますが、ファイルの名前を実行時に異なる実ファイルに関連付けることができます。

COBOL プログラム内の *file-name* を変更するには、入力ステートメントと出力ステートメントを変更し、プログラムを再コンパイルしなければなりません。または、DD ステートメントの *export* の *assignment-name* コマンドを変更して、実行時に異なるファイルを使用できます。

OPEN ステートメントの実行時に有効な環境変数値が、COBOL ファイル名とシステム・ファイル名の関連付け (任意のパス指定を含む) に使用されます。

例: さまざまな入力ファイルの使用:

この例では、プログラムの実行前に環境変数を設定することにより、同じ COBOL プログラムを使用してさまざまなファイルにアクセスする場合は示します。

次の SELECT 節を含む COBOL プログラムを考えてみます。

```
SELECT MASTER ASSIGN TO MASTERA
```

プログラムは、そのプログラム内の MASTER というファイルを使用して、checking または savings ファイルにアクセスするものとします。これを行うには、プログラムの実行前に次のうち該当する方のステートメントを使用して、MASTERA 環境変数を設定します。なお、次のステートメントでは、checking ファイルおよび savings ファイルが /accounts ディレクトリーにあることを前提としています。

```
export MASTERA=/accounts/checking
export MASTERA=/accounts/savings
```

ソースを変更または再コンパイルしなくても、同じプログラムを使用して、プログラム内の MASTER というファイルとして checking または savings ファイルのどちらにでもアクセスすることができます。

データの記述

データの特徴を定義し、データ定義をグループ化して、DATA DIVISION の 1 つ以上のセクションに入れなければなりません。

これらのセクションは、以下のタイプのデータを定義するときに使用できます。

- 入出力操作で使用するデータ: FILE SECTION
- 内部処理用に作成するデータ。
 - ストレージを静的に割り振り、実行単位 の存続期間中存在させる場合: WORKING-STORAGE SECTION
 - プログラムに入るたびにストレージを割り振り、プログラムからの戻り時に割り振りを解除させる場合: LOCAL-STORAGE SECTION
- 別のプログラムからのデータ: LINKAGE SECTION

COBOL for AIX コンパイラーでは、DATA DIVISION エLEMENTの最大サイズの制限があります。詳細については、下記のコンパイラー限界値に関する関連参照を参照してください。

関連概念

14 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

関連タスク

『入出力操作でのデータの使用』

16 ページの『別のプログラムからのデータの使用』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

266 ページの『ADDR』

コンパイラー限界値 (COBOL for AIX 言語解説書)

入出力操作でのデータの使用

入出力操作で使用するデータは、FILE SECTION で定義します。

データに関する以下の情報を提供してください。

- プログラムが使用する入出力ファイルの名前を指定します。FD 記入項目を使用して、PROCEDURE DIVISION の入出力ステートメントで参照できるファイルに名前を与えます。

FILE SECTION 内で定義されたデータ項目は、ファイルが正常にオープンされるまでは PROCEDURE DIVISION のステートメントにとって使用可能ではありません。

- FD 記入項目の後のレコード記述で、ファイル内のレコードおよびフィールドを記述します。レコード名は、WRITE および REWRITE ステートメントの対象となります。

同じ実行単位内のプログラムは、同じ COBOL ファイル名を参照できます。

別々にコンパイルされたプログラムには EXTERNAL 節を使用することができます。EXTERNAL として定義されたファイルは、実行単位の中でそのファイルを記述する任意のプログラムによって参照することができます。

ネストされた構造、すなわち含まれている構造の中のプログラムには、GLOBAL 節を使用できます。あるプログラムが別のプログラムを (直接または間接的に) 含む場合には、どちらのプログラムも GLOBAL ファイル名を参照することによって共通のファイルにアクセスすることができます。

COBOL ソース・プログラムで外部またはグローバル・ファイル定義を使用せずに、物理ファイルを共有できます。例えば、アプリケーションに 2 つの COBOL ファイル名があるが、それらの COBOL ファイルが 1 つのシステム・ファイルに関連付けられるように指定できます。

```
SELECT F1 ASSIGN TO MYFILE.
SELECT F2 ASSIGN TO MYFILE.
```

関連概念

503 ページの『ネストされたプログラム』

関連タスク

528 ページの『プログラム間でのファイルの共用 (外部ファイル)』

関連参照

『FILE SECTION 記入項目』

FILE SECTION 記入項目

FILE SECTION で使用できる項目の要約を以下の表に示します。

表 1. FILE SECTION 記入項目

節	定義対象
FD	PROCEDURE DIVISION の入出力ステートメント (OPEN、CLOSE、READ、START、および DELETE) 内で参照される <i>file-name</i> 。SELECT 節内の <i>file-name</i> と一致しなければなりません。 <i>file-name</i> は、 <i>assignment-name</i> を通じてシステム・ファイルに関連付けられます。
RECORD CONTAINS <i>n</i>	論理レコード (固定長) のサイズ。整数サイズは、レコード内のデータ項目の USAGE とは無関係に、レコードのバイト数を示します。

表 1. FILE SECTION 記入項目 (続き)

節	定義対象
RECORD IS VARYING	論理レコード (可変長) のサイズ。 整数サイズが指定された場合は、レコード内のデータ項目の USAGE とは無関係に、レコードのバイト数を示します。
RECORD CONTAINS <i>n</i> TO <i>m</i>	論理レコード (可変長) のサイズ。 整数サイズは、レコード内のデータ項目の USAGE とは無関係に、レコードのバイト数を示します。
VALUE OF	ファイルに関連付けられているラベル・レコードの項目。コメントのみ。
DATA RECORDS	ファイルに関連付けられているレコードの名前。コメントのみ。
RECORDING MODE	順次ファイル用レコード・タイプ

関連参照

File section (「*COBOL for AIX 言語解説書*」)

WORKING-STORAGE と LOCAL-STORAGE の比較

データ項目の割り振りと初期化がどのように行われるかは、それらの項目が WORKING-STORAGE SECTION の項目であるか LOCAL-STORAGE SECTION の項目であるかによって異なります。

プログラムが呼び出されると、そのプログラムに関連付けられている WORKING-STORAGE が割り振られます。

VALUE 節を持つすべてのデータ項目は、そのときに適切な値に初期化されます。その実行単位の存続期間中、WORKING-STORAGE 項目はそれぞれ最後に使われた状態を維持します。例外は次のとおりです。

- PROGRAM-ID 段落に INITIAL が指定されたプログラム。

この場合、WORKING-STORAGE データ項目は、プログラムに入るたびに再初期化されます。

- 動的に呼び出されてから取り消されるサブプログラム。

この場合、WORKING-STORAGE データ項目は、CANCEL 後のプログラムへの最初の再入時に再初期化されます。

WORKING-STORAGE は、実行単位の終了時に割り振り解除されます。

COBOL のクラス定義における WORKING-STORAGE については、関連するタスクを参照してください。

LOCAL-STORAGE データの別のコピーがプログラムまたはメソッドの個々の呼び出しに対して割り振られ、プログラムまたはメソッドから戻る時点で解放されます。LOCAL-STORAGE 項目に対して VALUE 節を指定すると、起動または呼び出しのたびに、項目はその値に初期化されます。VALUE 節を指定しないと、項目の初期値は未定義になります。

15 ページの『例: ストレージ・セクション』

関連タスク

502 ページの『メインプログラムまたはサブプログラムの終了と再入』
545 ページの『第 29 章 マルチスレッド化のための COBOL プログラムの準備』
434 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

関連参照

作業用ストレージ・セクション (COBOL for AIX 言語解説書)
ローカル・ストレージ・セクション (COBOL for AIX 言語解説書)

例: ストレージ・セクション

以下に、WORKING-STORAGE と LOCAL-STORAGE の両方を使用する再帰的プログラムの例を示します。

```
CBL apost,pgmn(lu)
*****
* Recursive Program - Factorials
*****
IDENTIFICATION DIVISION.
Program-Id. factorial recursive.
ENVIRONMENT DIVISION.
DATA DIVISION.
Working-Storage Section.
01 numb pic 9(4) value 5.
01 fact pic 9(8) value 0.
Local-Storage Section.
01 num pic 9(4).
PROCEDURE DIVISION.
    move numb to num.

    if numb = 0
        move 1 to fact
    else
        subtract 1 from numb
        call 'factorial'
        multiply num by fact
    end-if.

    display num '! = ' fact.
    goback.
End Program factorial.
```

このプログラムは、次のような出力を生成します。

```
0000! = 00000001
0001! = 00000001
0002! = 00000002
0003! = 00000006
0004! = 00000024
0005! = 00000120
```

次の表は、プログラムの連続する再帰呼び出し (CALL) および結果として起こる戻り (GOBACK) における、LOCAL-STORAGE および WORKING-STORAGE 内のデータ項目の値の変化を示しています。戻り時、fact は 5! の値を次々に累算します (5 階乗)。

再帰呼び出し (CALL)	LOCAL-STORAGE の num の値	WORKING-STORAGE の numb の値	WORKING-STORAGE の fact の値
メイン	5	5	0

再帰呼び出し (CALL)	LOCAL-STORAGE の num の値	WORKING-STORAGE の numb の値	WORKING-STORAGE の fact の値
1	4	4	0
2	3	3	0
3	2	2	0
4	1	1	0
5	0	0	0

戻り (GOBACK)	LOCAL-STORAGE の num の値	WORKING-STORAGE の numb の値	WORKING-STORAGE の fact の値
5	0	0	1
4	1	0	1
3	2	0	2
2	3	0	6
1	4	0	24
メイン	5	0	120

関連概念

14 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

別のプログラムからのデータの使用

データを共有する方法は、プログラムのタイプによって異なります。別々にコンパイルされたプログラムでは、ネストされたプログラムや、再帰的またはマルチスレッド化されたプログラムの場合とは異なる方法でデータを共有します。

関連タスク

『別々にコンパイルされたプログラムでのデータの共有』

17 ページの『ネストされたプログラムでのデータの共有』

17 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共有』

517 ページの『データの受け渡し』

別々にコンパイルされたプログラムでのデータの共有

多くのアプリケーションは、相互に呼び出し、データを受け渡しする、別個にコンパイルされたプログラムから構成されます。呼び出されるプログラム内の LINKAGE SECTION を用いて、別のプログラムから渡されるデータを記述します。

呼び出し側プログラムでは、CALL . . . USING または INVOKE . . . USING ステートメントを使用してデータを渡してください。

関連タスク

517 ページの『データの受け渡し』

ネストされたプログラムでのデータの共用

アプリケーションの中には、ネストされたプログラム (他のプログラムに含まれているプログラム) から構成されるものもあります。レベル 01 のデータ項目には、GLOBAL 属性を指定できます。この属性を使用すると、宣言を含むネストされたプログラムがこれらのデータ項目にアクセスできるようになります。

ネストされたプログラムは、COMMON 属性で宣言された兄弟プログラム (同一の含まれているプログラム内で同じネスト・レベルにあるプログラム) のデータ項目にもアクセスできます。

関連概念

503 ページの『ネストされたプログラム』

再帰的またはマルチスレッド化されたプログラムでのデータの共用

プログラムが RECURSIVE 属性を持っている場合、またはプログラムを THREAD コンパイラー・オプションを指定してコンパイルする場合、プログラムの以降の呼び出しでは、LINKAGE SECTION で定義されたデータにアクセスできません。

LINKAGE SECTION のレコードをアドレッシングするには、以下のいずれかの技法を使用します。

- プログラムに引数を渡し、プログラムの USING 句に適切な位置のレコードを指定する。
- フォーマット-5 の SET ステートメントを使用する。

プログラムが RECURSIVE 属性を持っている場合、またはプログラムを THREAD コンパイラー・オプションを指定してコンパイルする場合、レコードのアドレスは、プログラム起動の特定のインスタンスについて有効です。同じプログラムの別の実行インスタンスにあるレコードのアドレスは、その実行インスタンスに対して再設定する必要があります。アドレスが設定されていないデータ項目を参照すると、予測できない結果が生じます。

関連概念

545 ページの『マルチスレッド化』

関連タスク

515 ページの『再帰呼び出しの実行』

関連参照

311 ページの『THREAD』

SET ステートメント (COBOL for AIX 言語解説書)

データの処理

プログラムの PROCEDURE DIVISION では、他の部で定義したデータを処理する実行可能ステートメントをコーディングします。PROCEDURE DIVISION には、1 つまたは 2 つのヘッダーと、プログラムのロジックが入れられます。

PROCEDURE DIVISION は、部のヘッダーとプロシージャー名のヘッダーで始まります。プログラムの部のヘッダーは、単に次のようにすることができます。

```
PROCEDURE DIVISION.
```

あるいは、USING 句を使用してパラメーターを受け取ったり、RETURNING 句を使用して値を戻すような部のヘッダーをコーディングすることができます。

参照によって (デフォルト) あるいは内容によって渡された引数を受け取るには、プログラムの部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION USING dataname  
PROCEDURE DIVISION USING BY REFERENCE dataname
```

dataname は、DATA DIVISION の LINKAGE SECTION で定義しなければなりません。

値によって渡されたパラメーターを受け取るには、プログラムの部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION USING BY VALUE dataname
```

結果として値を戻すためには、部のヘッダーを次のようにコーディングしてください。

```
PROCEDURE DIVISION RETURNING dataname2
```

また、PROCEDURE DIVISION のヘッダーの中で USING と RETURNING を組み合わせることもできます。

```
PROCEDURE DIVISION USING dataname RETURNING dataname2
```

dataname および *dataname2* は、LINKAGE SECTION で必ず定義しなければなりません。

関連概念

『PROCEDURE DIVISION 内でロジックが分割される方法』

関連タスク

601 ページの『反復コーディングの除去』

関連参照

手続き部ヘッダー (COBOL for AIX 言語解説書)

USING 句 (COBOL for AIX 言語解説書)

CALL ステートメント (COBOL for AIX 言語解説書)

PROCEDURE DIVISION 内でロジックが分割される方法

プログラムの PROCEDURE DIVISION は、セクションと段落に分割されます。セクションおよび段落には、文、ステートメント、および句が含まれています。

セクション

処理ロジックの論理的な副区分。

セクションにはセクション・ヘッダーがあり、オプションとして後ろに 1 つ以上の段落が続きます。

セクションは、PERFORM ステートメントのサブジェクトにすることができます。セクションのタイプの 1 つは宣言用です。

段落 セクション、プロシージャ、またはプログラムの再分割。

段落は、後ろにピリオドが付いた名前を持ち、その後文が続く場合と続かない場合があります。

段落は、ステートメントのサブジェクトにすることができます。

文 1 つ以上の COBOL ステートメントの並びであって、ピリオドで終わるもの。

ステートメント

2 つの数値の加算など、COBOL 処理の定義されたステップを実行します。

ステートメントはワードの有効な組み合わせで、COBOL 動詞で始まります。ステートメントには、命令ステートメント (無条件アクションを示す)、条件ステートメント、およびコンパイラ指示ステートメントがあります。ステートメントの論理的な終わりを示すためには、ピリオドではなく明示範囲終了符号を使用することをお勧めします。

句 ステートメントの再分割。

関連概念

21 ページの『コンパイラ指示ステートメント』

21 ページの『範囲終了符号』

『命令ステートメント』

『条件ステートメント』

22 ページの『宣言』

関連参照

PROCEDURE DIVISION の構成 (COBOL for AIX 言語解説書)

命令ステートメント

命令ステートメント (ADD、MOVE、INVOKE、または CLOSE など) は、取るべき無条件アクションを指示します。

命令ステートメントは、暗黙のまたは明示的な範囲終了符号を使用して終了することができます。

明示範囲終了符号で終わる条件ステートメントは、**範囲区切りステートメント** と呼ばれる命令ステートメントになります。命令ステートメント (または範囲区切りステートメント) のみをネストすることができます。

関連概念

『条件ステートメント』

21 ページの『範囲終了符号』

条件ステートメント

条件ステートメントには、単純な条件ステートメント (IF、EVALUATE、SEARCH) と、条件句またはオプションを含む命令ステートメントから構成された条件ステートメントの 2 種類があります。

条件ステートメントは、暗黙のまたは明示的な範囲終了符号を使用して終了することができます。条件ステートメントを明示的に終わらせると、それは範囲区切りステートメント (命令ステートメント) になります。

範囲区切りステートメントは、次のような方法で使用できます。

- COBOL 条件ステートメントの操作の範囲を区切り、ネストのレベルを明示的に指示する場合。

例えば、ネストされた IF の中の IF ステートメントの範囲を終了させるために、ピリオドではなく END-IF 句を使用します。

- COBOL 構文が命令ステートメントを必要とする条件ステートメントをコーディングする場合。

例えば、インライン PERFORM のオブジェクトとして条件ステートメントをコーディングします。

```
PERFORM UNTIL TRANSACTION-EOF
  PERFORM 200-EDIT-UPDATE-TRANSACTION
  IF NO-ERRORS
    PERFORM 300-UPDATE-COMMUTER-RECORD
  ELSE
    PERFORM 400-PRINT-TRANSACTION-ERRORS
  END-IF
  READ UPDATE-TRANSACTION-FILE INTO WS-TRANSACTION-RECORD
  AT END
    SET TRANSACTION-EOF TO TRUE
  END-READ
END-PERFORM
```

インライン PERFORM ステートメントには、明示範囲終了符号が必須ですが、これはライン外の PERFORM ステートメントについては無効です。

追加のプログラム制御として、条件ステートメントと一緒に NOT 句を使用することもできます。例えば、NOT ON SIZE ERROR のように、特定の例外が発生しない場合に実行される命令を指定することができます。NOT 句は、CALL ステートメントの ON OVERFLOW 句とは一緒に使用できませんが、ON EXCEPTION 句とは一緒に使用できます。

条件ステートメントをネストしてはなりません。ネストされるステートメントは、命令ステートメント (または範囲区切りステートメント) でなければならず、命令ステートメントの規則に従っていなければなりません。

以下に、範囲終了符号なしでコーディングされる場合の条件ステートメントの例を示します。

- ON SIZE ERROR が指定された算術ステートメント
- ON OVERFLOW が指定されたデータ操作ステートメント
- ON OVERFLOW が指定された CALL ステートメント
- INVALID KEY、AT END、または AT END-OF-PAGE が指定された I/O ステートメント
- AT END が指定された RETURN

関連概念

- 19 ページの『命令ステートメント』
- 21 ページの『範囲終了符号』

関連タスク

- 85 ページの『プログラム・アクションの選択』

関連参照

条件ステートメント (*COBOL for AIX 言語解説書*)

コンパイラ指示ステートメント

コンパイラ指示ステートメントは、コンパイラに、プログラム構造、COPY 処理、リスト制御、制御フロー、または CALL インターフェース規則について特定のアクションを行わせます。

コンパイラ指示ステートメントは、プログラム・ロジックの一部ではありません。

関連参照

319 ページの『第 15 章 コンパイラ指示ステートメント』
コンパイラ指示ステートメント (*COBOL for AIX 言語解説書*)

範囲終了符号

範囲終了符号は動詞またはステートメントの終了を示します。範囲終了符号には、明示的なものと暗黙的なものがあります。

明示範囲終了符号は、文を終了させることなく、動詞を終了させます。これは、END の後にハイフンと、終了させる動詞の名前を続けたものから構成されます (例えば、END-IF)。暗黙の範囲終了符号はピリオド (.) で、これはまだ終了されていないすべての先行ステートメントの範囲を終了させます。

次のプログラムの断片における 2 つのピリオドは、それぞれ IF ステートメントを終了させます。そのため、このコードは、明示範囲終了符号を持つ以下の例と同等です。

```
IF ITEM = "A"  
    DISPLAY "THE VALUE OF ITEM IS " ITEM  
    ADD 1 TO TOTAL  
    MOVE "C" TO ITEM  
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM.  
IF ITEM = "B"  
    ADD 2 TO TOTAL.  
  
IF ITEM = "A"  
    DISPLAY "THE VALUE OF ITEM IS " ITEM  
    ADD 1 TO TOTAL  
    MOVE "C" TO ITEM  
    DISPLAY "THE VALUE OF ITEM IS NOW " ITEM  
END-IF  
IF ITEM = "B"  
    ADD 2 TO TOTAL  
END-IF
```

暗黙の範囲終了符号を使用すると、ステートメントの終わる場所が不明確になることがあります。結果として、ステートメントを意図に反して終了させ、プログラムのロジックが変わる可能性があります。明示範囲終了符号を使用すると、プログラムが理解しやすくなり、ステートメントを意図に反して終了させることがなくなります。例えば、次のプログラム断片で、最初の暗黙の範囲例の最初のピリオドの位置を変更すると、コードの意味が変更されます。

```
IF ITEM = "A"  
    DISPLAY "VALUE OF ITEM IS " ITEM  
    ADD 1 TO TOTAL.
```

```
MOVE "C" TO ITEM
DISPLAY " VALUE OF ITEM IS NOW " ITEM
IF ITEM = "B"
  ADD 2 TO TOTAL.
```

最初のピリオドが IF ステートメントを終わらせるため、その後の MOVE ステートメントおよび DISPLAY ステートメントは、字下げの意味を無視し、ITEM の値に関係なく実行されます。

プログラムをより読みやすくし、ステートメントの意図しない終了を防ぐために、特に段落内では、明示範囲終了符号を使用するようにすべきです。暗黙の範囲終了符号は、段落の終わりまたはプログラムの終わりでのみ使用してください。

条件ステートメント内にネストされている命令ステートメントについての明示的範囲終了符号をコーディングする際には、注意が必要です。範囲終了符号が、それが意図されたステートメントと対にされるようにしてください。次の例では、範囲終了符号は最初の READ ステートメントと対になるように意図されましたが、実際には 2 つ目と対にされます。

```
READ FILE1
  AT END
    MOVE A TO B
    READ FILE2
END-READ
```

明示範囲終了符号が意図されたステートメントと対にされるようにするために、上記の例を次のようにコーディングし直すことができます。

```
READ FILE1
  AT END
    MOVE A TO B
  READ FILE2
END-READ
END-READ
```

関連概念

- 19 ページの『条件ステートメント』
- 19 ページの『命令ステートメント』

宣言

宣言は、例外条件が起こったときに実行される 1 つ以上の特殊目的セクションを提供します。

各宣言セクションは、そのセクションの機能を識別する USE ステートメントで始まります。プロシージャータン中に、条件が起こった場合に取りべきアクションを指定します。

関連タスク

- 332 ページの『入出力エラーの検出および処理』

関連参照

宣言 (COBOL for AIX 言語解説書)

第 2 章 データの使用

ここに示す情報は、非 COBOL プログラマーが、他のプログラム言語で使用されているデータに関する用語を COBOL 用語と関連付けるのに役立ちます。ここでは、COBOL の基礎である変数、構造、リテラル、定数、値の割り当てと表示、組み込み関数、およびテーブル (配列) とポインターについて紹介します。

関連タスク

『変数、構造、リテラル、および定数の使用』

27 ページの『データ項目への値の割り当て』

37 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

37 ページの『組み込み関数の使用 (組み込み関数)』

38 ページの『テーブル (配列) とポインターの使用』

183 ページの『第 10 章 国際環境でのデータの処理』

変数、構造、リテラル、および定数の使用

大半の高水準プログラム言語では、変数、構造 (グループ項目)、リテラル、または定数としてデータを表すという概念は同じです。

COBOL プログラムのデータは、英字、英数字、2 バイト文字セット (DBCS)、国別、または数値が可能です。また指標名を定義したり、USAGE POINTER、USAGE FUNCTION-POINTER、USAGE PROCEDURE-POINTER、または USAGE OBJECT REFERENCE として記述されたデータ項目を定義することもできます。データ定義はすべて、プログラムの DATA DIVISION に入れます。

関連タスク

『変数の使用』

24 ページの『データ項目とグループ項目の使用』

26 ページの『リテラルの使用』

26 ページの『定数の使用』

27 ページの『形象定数の使用』

関連参照

データのクラスおよびカテゴリ (*COBOL for AIX 言語解説書*)

変数の使用

変数 は、プログラム実行時に値を変更できるデータ項目です。ただし、値は、データ項目の名前と長さを指定するときに定義されたデータ型に制限されます。

例えば、お客様名がプログラム内の英数字データ項目であれば、次に示すように、そのお客様名を定義し使用することができます。

```
Data Division.  
01 Customer-Name           Pic X(20).  
01 Original-Customer-Name Pic X(20).
```

```

. . .
Procedure Division.
  Move Customer-Name to Original-Customer-Name
. . .

```

代わりに、PICTURE 節を Pic N(20) と指定し、その項目に USAGE NATIONAL 節を指定することにより、上記のお客様名を国別データ項目として宣言できます。国別データ項目は Unicode UTF-16 で表され、その場合、ほとんどの文字が 2 バイトのストレージで表されます。

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

189 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

296 ページの『NSYMBOL』

196 ページの『文字データの保管』

PICTURE 節 (*COBOL for AIX 言語解説書*)

データ項目とグループ項目の使用

連データ項目は、階層データ構造の一部となることができます。従属データ項目を持たないデータ項目は、**基本項目** と呼ばれます。1 つ以上の従属データ項目で構成されるデータ項目を**グループ項目**と呼びます。

レコードは、基本項目またはグループ項目のどちらでも構いません。グループ項目は、**英数字グループ項目** または**国別グループ項目** のいずれでも構いません。

例えば、以下の Customer-Record は英数字グループ項目であり、それぞれが基本データ項目を含んでいる 2 つの従属英数字グループ項目 (Customer-Name と Part-Order) で構成されています。これらのグループ項目は暗黙的に USAGE DISPLAY を持ちます。以下に示すように、PROCEDURE DIVISION の MOVE ステートメントで、グループ項目全体またはグループ項目の一部を参照できます。

```

Data Division.
File Section.
FD Customer-File
  Record Contains 45 Characters.
01 Customer-Record.
  05 Customer-Name.
    10 Last-Name      Pic x(17).
    10 Filler         Pic x.
    10 Initials       Pic xx.
  05 Part-Order.
    10 Part-Name      Pic x(15).
    10 Part-Color     Pic x(10).
Working-Storage Section.
01 Orig-Customer-Name.
  05 Surname         Pic x(17).
  05 Initials        Pic x(3).
01 Inventory-Part-Name Pic x(15).
. . .
Procedure Division.
  Move Customer-Name to Orig-Customer-Name
  Move Part-Name to Inventory-Part-Name
. . .

```


代わりに、以下に示すように DATA DIVISION の宣言を変更することにより、Customer-Record を、2 つの従属国別グループ項目で構成される国別グループ項目として定義できます。国別グループ項目の振る舞いは、ほとんどの操作でカテゴリー国別の基本データ項目と同じです。GROUP-USAGE NATIONAL 節は、グループ項目およびその従属グループ項目が国別グループであることを示します。国別グループ内の従属基本項目は、明示的または暗黙的に USAGE NATIONAL として記述されている必要があります。

```
Data Division.  
File Section.  
FD Customer-File  
   Record Contains 90 Characters.  
01 Customer-Record      Group-Usage National.  
   05 Customer-Name.  
     10 Last-Name       Pic n(17).  
     10 Filler          Pic n.  
     10 Initials       Pic nn.  
   05 Part-Order.  
     10 Part-Name      Pic n(15).  
     10 Part-Color     Pic n(10).  
Working-Storage Section.  
01 Orig-Customer-Name  Group-Usage National.  
   05 Surname          Pic n(17).  
   05 Initials         Pic n(3).  
01 Inventory-Part-Name Pic n(15) Usage National.  
.  
.  
.  
Procedure Division.  
   Move Customer-Name to Orig-Customer-Name  
   Move Part-Name to Inventory-Part-Name  
.  
.  
.
```

上記の例のグループ項目は、グループ・レベルで USAGE NATIONAL 節を指定することもできます。グループ・レベルの USAGE 節は、グループ内のそれぞれの基本データ項目に適用されます (ですから、これは便利な省略表現と言えます)。ただし、USAGE NATIONAL 節を指定しているグループは、グループ内の基本項目の表記にもかかわらず、国別グループではありません。この USAGE 節を指定しているグループは英数字グループであり、多数の操作 (移動や比較など) において USAGE DISPLAY の基本データ項目と同様の振る舞いをします (ただし、データの編集や変換は行われません)。

関連概念

188 ページの『Unicode および言語文字のエンコード』

192 ページの『国別グループ』

関連タスク

189 ページの『COBOL での国別データ (Unicode) の使用』

194 ページの『国別グループの使用』

関連参照

13 ページの『FILE SECTION 記入項目』

196 ページの『文字データの保管』

グループ項目のクラスおよびカテゴリー (*COBOL for AIX 言語解説書*)

PICTURE 節 (*COBOL for AIX 言語解説書*)

MOVE ステートメント (*COBOL for AIX 言語解説書*)

USAGE 節 (*COBOL for AIX 言語解説書*)

リテラルの使用

リテラル とは、値が文字それ自体によって与えられる文字ストリングのことです。データ項目に使用したい値が分かっている場合には、PROCEDURE DIVISION でそのデータ値のリテラル表記を使用できます。

値のデータ項目を宣言したり、データ名を使用してデータ項目を参照したりする必要はありません。例えば、以下に示すように英数字リテラルを移動することにより、出力ファイル用のエラー・メッセージを準備できます。

```
Move "Name is not valid" To Customer-Name
```

以下に示すように数値リテラルを使用すれば、データ項目を特定の整数値と比較できます。次の例では、"Name is not valid" は英数字リテラルであり、03519 は数値リテラルです。

```
01 Part-number      Pic 9(5).  
  . . .  
  If Part-number = 03519 then display "Part number was found"
```

英数字リテラル内では、16 進表記形式 (X') を使用して、制御文字 X'00' から X'1F' を表すことができます。これらの制御文字を英数字リテラルの基本形式で指定すると、結果は予測不能です。

NSYMBOL(NATIONAL) コンパイラー・オプションが有効であるときには、開始区切り文字 N" または N' を使用して国別リテラルを指定でき、NSYMBOL(DBCS) コンパイラー・オプションが有効であるときには、同様にして DBCS リテラルを指定できます。

開始区切り文字 NX" または NX' を使用すれば、(NSYMBOL コンパイラー・オプションの設定とは無関係に) 16 進表記の国別リテラルを指定できます。4 桁の 16 進数字のそれぞれのグループが、単一国別文字を指定します。

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

190 ページの『国別リテラルの使用』

206 ページの『DBCS リテラルの使用』

関連参照

296 ページの『NSYMBOL』

リテラル (COBOL for AIX 言語解説書)

定数の使用

定数 は、1 つの値しか持たないデータ項目です。COBOL では、定数を表す構造を定義していません。ただし、データ記述に VALUE 節をコーディングすることにより (INITIALIZE ステートメントをコーディングする代わりに)、初期値を使用してデータ項目を定義することができます。

```

Data Division.
01 Report-Header  pic x(50)  value "Company Sales Report".
. . .
01 Interest      pic 9v9999 value 1.0265.

```

上記の例は、英数字および数字データ項目を初期化します。同様に、VALUE 節を、国別または DBCS 定数の定義に使用できます。

関連タスク

189 ページの『COBOL での国別データ (Unicode) の使用』
 205 ページの『DBCS サポート用のコーディング』

形象定数の使用

通常用いられる特定の定数およびリテラルは、**形象定数** と呼ばれる予約語として次のものが用意されています。ZERO、SPACE、HIGH-VALUE、LOW-VALUE、QUOTE、NULL、および ALL *literal*。これらは固定値を表すため、形象定数はデータ定義を必要としません。

以下に例を示します。

```
Move Spaces To Report-Header
```

関連タスク

191 ページの『国別文字形象定数の使用』
 205 ページの『DBCS サポート用のコーディング』

関連参照

形象定数 (COBOL for AIX 言語解説書)

データ項目への値の割り当て

データ項目を定義した後、いつでもそれに値を割り当てることができます。COBOL における割り当ては、その背後にある目的によって多くの形式を取ります。

表 2. プログラム内でのデータ項目の割り当て

目的	方法
データ項目または大きいデータ域に値を割り当てる。	以下のいずれかの方法を使用する。 <ul style="list-style-type: none"> INITIALIZE ステートメント MOVE ステートメント STRING または UNSTRING ステートメント VALUE 節 (データ項目を、プログラムが初期状態にあるときにそれに与えたい値に設定する場合)
算術の結果を割り当てる。	COMPUTE、ADD、SUBTRACT、MULTIPLY、または DIVIDE ステートメントを使用します。
データ項目内の文字または文字グループを検査または置換します。	INSPECT ステートメントを使用する。
ファイルから値を受け取る。	READ (または READ INTO) ステートメントを使用する。

表 2. プログラム内でのデータ項目の割り当て (続き)

目的	方法
システム入力装置またはファイルから値を受け取る。	ACCEPT ステートメントを使用する。
定数を設定します。	データ項目の定義内で VALUE 節を使用し、そのデータ項目を受け取り側として使用しない。このような項目は、コンパイラが読み取り専用の定数としての扱いを強制しなくても、実質的に定数となります。
次のいずれかの処置。 <ul style="list-style-type: none"> • テーブル・エレメントに関連付けられた値を指標に設定する • 外部スイッチの状況を ON または OFF に設定する • 条件名にデータを移動して、条件を真にする • POINTER、PROCEDURE-POINTER、または FUNCTION-POINTER データ項目にアドレスを設定する • OBJECT REFERENCE データ項目にオブジェクト・インスタンスに関連付ける 	SET ステートメントを使用する。

『例: データ項目の初期化』

関連タスク

- 31 ページの『構造の初期化 (INITIALIZE)』
- 33 ページの『基本データ項目への値の割り当て (MOVE)』
- 34 ページの『グループ・データ項目への値の割り当て (MOVE)』
- 35 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
- 99 ページの『データ項目の結合 (STRING)』
- 102 ページの『データ項目の分割 (UNSTRING)』
- 35 ページの『算術結果の割り当て (MOVE または COMPUTE)』
- 110 ページの『データ項目の計算および置換 (INSPECT)』
- 183 ページの『第 10 章 国際環境でのデータの処理』

例: データ項目の初期化

以下の例は、INITIALIZE ステートメントを使用して、英数字、国別編集、および数字編集データ項目を含めたいろいろな種類のデータ項目を初期化する方法を示しています。

INITIALIZE ステートメントは、機能の面で 1 つ以上の MOVE ステートメントと同等です。初期化に関する関連作業を見るならば、グループ項目に対して INITIALIZE ステートメントを使用して、ある特定データ・カテゴリー内にあるすべての従属データ項目をどのように便利な方法で初期化できるかが分かります。

データ項目のブランクまたはゼロへの初期化:

INITIALIZE *identifier-1*

<i>identifier-1</i> PICTURE	<i>identifier-1</i> (初期化前)	<i>identifier-1</i> (初期化後)
9(5)	12345	00000
X(5)	AB123	bbbb ¹

<i>identifier-1</i> PICTURE	<i>identifier-1</i> (初期化前)	<i>identifier-1</i> (初期化後)
N(3)	004100420031 ²	002000200020 ³
99XX9	12AB3	bbbb ¹
XXBX/XX	ABbC/DE	bbbb/bb ¹
**99.9CR	1234.5CR	**00.0bb ¹
A(5)	ABCDE	bbbb ¹
+99.99E+99	+12.34E+02	+00.00E+00

1. 記号 *b* は、ブランク・スペースを表します。

2. 国別 (UTF-16) 文字「AB1」の 16 進表記。例では、*identifier-1* が Usage National を持っているとして想定しています。

3. 国別 (UTF-16) 文字「 」(3 個のブランク・スペース) の 16 進表記。*identifier-1* が Usage National として定義されておらず、また NSYMBOL(DBCS) が有効である場合、INITIALIZE は代わりに DBCS スペース(「2020」)を *identifier-1* に保管することに注意してください。

英数字データ項目の初期化:

```
01 ALPHANUMERIC-1 PIC X VALUE "y".
01 ALPHANUMERIC-3 PIC X(1) VALUE "A".
...
INITIALIZE ALPHANUMERIC-1
REPLACING ALPHANUMERIC DATA BY ALPHANUMERIC-3
```

ALPHANUMERIC-3	ALPHANUMERIC-1 (初期化前)	ALPHANUMERIC-1 (初期化後)
A	y	A

英数字右揃えデータ項目の初期化:

```
01 ANJUST PIC X(8) VALUE SPACES JUSTIFIED RIGHT.
01 ALPHABETIC-1 PIC A(4) VALUE "ABCD".
...
INITIALIZE ANJUST
REPLACING ALPHANUMERIC DATA BY ALPHABETIC-1
```

ALPHABETIC-1	ANJUST (初期化前)	ANJUST (初期化後)
ABCD	bbbbbbbb ¹	bbbbABCD ¹

1. 記号 *b* は、ブランク・スペースを表します。

英数字編集データ項目の初期化:

```
01 ALPHANUM-EDIT-1 PIC XXBX/XXX VALUE "ABbC/DEF".
01 ALPHANUM-EDIT-3 PIC X/BB VALUE "M/bb".
...
INITIALIZE ALPHANUM-EDIT-1
REPLACING ALPHANUMERIC-EDITED DATA BY ALPHANUM-EDIT-3
```

ALPHANUM-EDIT-3	ALPHANUM-EDIT-1 (初期化前)	ALPHANUM-EDIT-1 (初期化後)
M/bb ¹	ABbC/DEF ¹	M/bb/bbb ¹

1. 記号 *b* は、ブランク・スペースを表します。

国別データ項目の初期化:

```

01 NATIONAL-1      PIC NN  USAGE NATIONAL  VALUE N"AB".
01 NATIONAL-3      PIC NN  USAGE NATIONAL  VALUE N"CD".
. . .
      INITIALIZE NATIONAL-1
      REPLACING NATIONAL DATA BY NATIONAL-3

```

NATIONAL-3	NATIONAL-1 (初期化前)	NATIONAL-1 (初期化後)
00430044 ¹	00410042 ²	00430044 ¹
1. 国別文字「CD」の 16 進表記 2. 国別文字「AB」の 16 進表記		

国別編集データ項目の初期化:

```

01 NATL-EDIT-1     PIC 0NN  USAGE NATIONAL  VALUE N"123".
01 NATL-3          PIC NNN  USAGE NATIONAL  VALUE N"456".
. . .
      INITIALIZE NATL-EDIT-1
      REPLACING NATIONAL-EDITED DATA BY NATL-3

```

NATL-3	NATL-EDIT-1 (初期化前)	NATL-EDIT-1 (初期化後)
003400350036 ¹	003100320033 ²	003000340035 ³
1. 国別文字「456」の 16 進表記 2. 国別文字「123」の 16 進表記 3. 国別文字「045」の 16 進表記		

数値 (ゾーン 10 進数) データ項目の初期化:

```

01 NUMERIC-1       PIC 9(8)      VALUE 98765432.
01 NUM-INT-CMPT-3  PIC 9(7)  COMP VALUE 1234567.
. . .
      INITIALIZE NUMERIC-1
      REPLACING NUMERIC DATA BY NUM-INT-CMPT-3

```

NUM-INT-CMPT-3	NUMERIC-1 (初期化前)	NUMERIC-1 (初期化後)
1234567	98765432	01234567

数値 (国別 10 進数) データ項目の初期化:

```

01 NAT-DEC-1       PIC 9(3)  USAGE NATIONAL VALUE 987.
01 NUM-INT-BIN-3   PIC 9(2)  BINARY VALUE 12.
. . .
      INITIALIZE NAT-DEC-1
      REPLACING NUMERIC DATA BY NUM-INT-BIN-3

```

NUM-INT-BIN-3	NAT-DEC-1 (初期化前)	NAT-DEC-1 (初期化後)
12	003900380037 ¹	003000310032 ²
1. 国別文字「987」の 16 進表記 2. 国別文字「012」の 16 進表記		

数字編集 (USAGE DISPLAY) データ項目の初期化:

```

01 NUM-EDIT-DISP-1 PIC $ZZ9V VALUE "$127".
01 NUM-DISP-3      PIC 999V  VALUE 12.
. . .
      INITIALIZE NUM-EDIT-DISP-1
        REPLACING NUMERIC DATA BY NUM-DISP-3

```

NUM-DISP-3	NUM-EDIT-DISP-1 (初期化前)	NUM-EDIT-DISP-1 (初期化後)
012	\$127	\$ 12

数字編集 (USAGE NATIONAL) データ項目の初期化:

```

01 NUM-EDIT-NATL-1 PIC $ZZ9V NATIONAL VALUE N"$127".
01 NUM-NATL-3      PIC 999V  NATIONAL VALUE 12.
. . .
      INITIALIZE NUM-EDIT-NATL-1
        REPLACING NUMERIC DATA BY NUM-NATL-3

```

NUM-NATL-3	NUM-EDIT-NATL-1 (初期化前)	NUM-EDIT-NATL-1 (初期化後)
003000310032 ¹	0024003100320037 ²	0024002000310032 ³

1. 国別文字「012」の 16 進表記
2. 国別文字「\$127」の 16 進表記
3. 国別文字「\$ 12」の 16 進表記

関連タスク

- 『構造の初期化 (INITIALIZE)』
- 72 ページの『テーブルの初期化 (INITIALIZE)』
- 41 ページの『数値データの定義』

関連参照

- 296 ページの『NSYMBOL』

構造の初期化 (INITIALIZE)

INITIALIZE ステートメントをそのグループ項目に適用することによって、グループ項目内のすべての従属データ項目の値を初期化することができます。ただし、グループ内のすべての項目を初期化することが必要な場合を除き、グループ全体を初期化することは非効率的です。

以下の例は、プログラムが作成するトランザクション・レコードの各フィールドをスペースおよびゼロにリセットする方法を示しています。フィールドの値は、作成される各レコードで同一というわけではありません。(トランザクション・レコードは、英数字グループ項目 TRANSACTION-OUT として定義されています。)

```

01 TRANSACTION-OUT.
   05 TRANSACTION-CODE      PIC X.
   05 PART-NUMBER           PIC 9(6).
   05 TRANSACTION-QUANTITY PIC 9(5).
   05 PRICE-FIELDS.
      10 UNIT-PRICE         PIC 9(5)V9(2).
      10 DISCOUNT          PIC V9(2).
      10 SALES-PRICE        PIC 9(5)V9(2).
. . .
      INITIALIZE TRANSACTION-OUT

```

レコード	TRANSACTION-OUT (初期化前)	TRANSACTION-OUT (初期化後)
1	R001383000240000000000000000000	b000000000000000000000000000000 ¹
2	R001390000480000000000000000000	b000000000000000000000000000000 ¹
3	S001410000120000000000000000000	b000000000000000000000000000000 ¹
4	C00138300000000004250000000000	b000000000000000000000000000000 ¹
5	C00201000000000000001000000000	b000000000000000000000000000000 ¹

1. 記号 *b* は、ブランク・スペースを表します。

同様に国別グループ項目内のすべての従属データ項目の値は、そのグループ項目に INITIALIZE ステートメントを適用することにより、リセットできます。以下の構造は、上記の構造と似ていますが、Unicode UTF-16 データを使用している点で異なります。

```

01 TRANSACTION-OUT GROUP-USAGE NATIONAL.
   05 TRANSACTION-CODE          PIC N.
   05 PART-NUMBER                PIC 9(6).
   05 TRANSACTION-QUANTITY      PIC 9(5).
   05 PRICE-FIELDS.
       10 UNIT-PRICE            PIC 9(5)V9(2).
       10 DISCOUNT             PIC V9(2).
       10 SALES-PRICE           PIC 9(5)V9(2).
   . . .
INITIALIZE TRANSACTION-OUT

```

トランザクション・レコードの直前の内容とは無関係に、上記の INITIALIZE ステートメントの実行後には次のようになります。

- TRANSACTION-CODE には NX"0020" (国別スペース) が入ります。
- TRANSACTION-OUT の残りの 27 の国別文字の位置には、NX"0030" (国別 10 進数のゼロ) が入ります。

INITIALIZE ステートメントを使用して英数字または国別グループ・データ項目を初期化すると、そのデータ項目はグループ項目として、つまりグループ・セマンティクスで処理されます。グループ内の基本データ項目は、上記の例に示されているように認識および処理されます。INITIALIZE ステートメントの REPLACING 句をコーディングしない場合、次のようになります。

- SPACE は、英字、英数字、英数字編集、DBCS、カテゴリ国別、および国別編集の各受信項目用の暗黙の送信項目です。
- ZERO は、数字および数字編集受信項目用の暗黙の送信項目です。

関連概念

192 ページの『国別グループ』

関連タスク

72 ページの『テーブルの初期化 (INITIALIZE)』

194 ページの『国別グループの使用』

関連参照

INITIALIZE ステートメント (COBOL for AIX 言語解説書)

基本データ項目への値の割り当て (MOVE)

MOVE ステートメントを使用して、基本データ項目に値を割り当てます。

以下の文は、基本データ項目 `Customer-Name` の内容を、基本データ項目 `Orig-Customer-Name` に割り当てます。

```
Move Customer-Name to Orig-Customer-Name
```

`Customer-Name` が `Orig-Customer-Name` より長い場合は、右側で切り捨てが起こります。`Customer-Name` が短い場合には、`Orig-Customer-Name` の右側の余分な文字位置がスペースで埋められます。

数値を含んでいるデータ項目の場合、文字データ項目の場合よりも移動が複雑になることがあります。これは、数値には表現方法が幾通りもあるためです。文字データでは桁ごとの移動が行われますが、一般に数値では、可能な場合には代数值が移動されます。例えば、以下の MOVE ステートメントの場合、`Item-x` には、値 3.0 (0030 で表される) が入ります。

```
01 Item-x      Pic 999v9.  
...  
    Move 3.06 to Item-x
```

英字、英数字、英数字編集、DBCS、整数、または数字編集の各データ項目を、カテゴリ国別または国別編集データ項目に移動でき、送出項目が変換されます。国別データ項目をカテゴリ国別または国別編集のデータ項目に移動できます。カテゴリ国別データ項目の内容に数値が含まれている場合、その項目を、数値、数字編集、外部浮動小数点、または内部浮動小数点のデータ項目に移動できます。国別編集データ項目は、カテゴリ国別データ項目または別の国別編集データ項目にのみ移動できます。埋め込みや切り捨てが行われることがあるので、

基本移動の全詳細については、MOVE ステートメントに関する以下の関連資料を参照してください。

以下の例は、国別データ項目に移動される、ギリシャ語の英数字データ項目を示しています。

```
...  
01 Data-in-Unicode  Pic N(100) usage national.  
01 Data-in-Greek   Pic X(100).  
...  
    Read Greek-file into Data-in-Greek  
    Move Data-in-Greek to Data-in-Unicode
```

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

34 ページの『グループ・データ項目への値の割り当て (MOVE)』

197 ページの『国別 (Unicode) 表現との間の変換』

関連参照

データのクラスおよびカテゴリ (COBOL for AIX 言語解説書)

MOVE ステートメント (COBOL for AIX 言語解説書)

グループ・データ項目への値の割り当て (MOVE)

グループ・データ項目に値を割り当てるには、MOVE ステートメントを使用してください。

国別グループ項目 (GROUP-USAGE NATIONAL 節で記述されているデータ項目) を別の国別グループ項目に移動できます。それぞれの国別グループ項目がカテゴリ-国別の基本項目であるかのように、すなわち、それぞれの項目が PIC N(*m*) (ここで、*m* はその項目の長さ (国別文字位置数) です) として記述されているかのように、コンパイラーは移動を処理します。

英数字グループ項目を、英数字グループ項目または国別グループ項目に移動できます。国別グループ項目を英数字グループ項目に移動することもできます。コンパイラーはこのような移動をグループ移動として実行します。すなわち、送信グループまたは受信グループ内の個々の基本項目を考慮に入れずに、また送信データ項目を変換せずに実行します。送信および受信グループ項目内の従属データ記述の互換性が保たれるようにしてください。実行時に破壊オーバーラップが起きたとしても移動は行われます。

CORRESPONDING 句を MOVE ステートメントにコーディングすれば、従属基本項目を、あるグループ項目から別のグループ項目の同一名の対応する従属基本項目に移動できます。

```
01 Group-X.  
   02 T-Code    Pic X    Value "A".  
   02 Month     Pic 99   Value 04.  
   02 State     Pic XX   Value "CA".  
   02 Filler    PIC X.  
01 Group-N     Group-Usage National.  
   02 State     Pic NN.  
   02 Month     Pic 99.  
   02 Filler    Pic N.  
   02 Total     Pic 999.  
...  
MOVE CORR Group-X TO Group-N
```

上記の例では、Group-N 内の State および Month は、Group-X から State および Month の国別表現の値をそれぞれ受け取ります。Group-N 内の他のデータ項目は未変更のままです。(受信グループ項目内の Filler 項目は、MOVE CORRESPONDING ステートメントによっては変更されません。)

MOVE CORRESPONDING ステートメントでは、送信グループ項目および受信グループ項目はグループ項目として扱われ、基本データ項目としては扱われません。グループ・セマンティクスが適用されます。すなわち、それぞれのグループ内の基本データ項目が認識され、結果は、対応するデータ項目のそれぞれのペアが、別個の MOVE ステートメントで参照された場合と同じになります。データ変換は、以下の関連した解説書に明記されている MOVE ステートメントの規則に従って実行されます。どのタイプの基本データ項目が対応するかについての詳細は、CORRESPONDING 句に關する関連した解説書を参照してください。

関連概念

188 ページの『Unicode および言語文字のエンコード』

192 ページの『国別グループ』

関連タスク

33 ページの『基本データ項目への値の割り当て (MOVE)』
194 ページの『国別グループの使用』
197 ページの『国別 (Unicode) 表現と間の変換』

関連参照

グループ項目のクラスおよびカテゴリー (*COBOL for AIX 言語解説書*)
MOVE ステートメント (*COBOL for AIX 言語解説書*)
CORRESPONDING 句 (*COBOL for AIX 言語解説書*)

算術結果の割り当て (MOVE または COMPUTE)

データ項目に数値を割り当てるときには、MOVE ステートメントではなく COMPUTE ステートメントを使用することを考慮してください。

```
Move w to z  
Compute z = w
```

上記の例では、ほとんどの場合、2 つのステートメントは同じ効果があります。ただし、MOVE ステートメントは割り当て時に切り捨てを行います。DIAGTRUNC コンパイラー・オプションを使用して、数値受け取り側で切り捨てが起こる可能性のある MOVE ステートメントについてコンパイラーが警告を出すように要求することができます。

ただし、実行時に左側の有効数字が失われる場合、COMPUTE ステートメントを使用するとこの条件を検出し、それに対処することができます。COMPUTE ステートメントの ON SIZE ERROR 句を使用すると、コンパイラーはサイズ・オーバーフロー条件を検出するコードを生成します。条件が起こると、ON SIZE ERROR 句内のコードが実行され、z の内容は未変更のままになります。ON SIZE ERROR 句を指定しないと、割り当て時に切り捨てが行われます。MOVE ステートメントの ON SIZE ERROR サポートはありません。

COMPUTE ステートメントを使用して、算術式 または組み込み関数 の結果をデータ項目に割り当てすることもできます。以下に例を示します。

```
Compute z = y + (x ** 3)  
Compute x = Function Max(x y z)
```

関連参照

279 ページの『DIAGTRUNC』
組み込み関数 (*COBOL for AIX 言語解説書*)

画面またはファイルからの入力の割り当て (ACCEPT)

データ項目に値を割り当てする方法の 1 つとして、画面またはファイルから値を読み取ることができます。

画面からデータを入力するには、最初にモニターを SPECIAL-NAMES 段落内の簡略名と関連付けます。次に、ACCEPT を使用して、画面から入力された入力行をデータ項目に割り当てます。以下に例を示します。

```
Environment Division.  
Configuration Section.  
Special-Names.  
    Console is Names-Input.  
    . . .  
    Accept Customer-Name From Names-Input
```

画面ではなくファイルから読み取る場合は、次のいずれかの変更を行います。

- `Console` を `device` に変更します (ここで、`device` は任意の有効なシステム装置 (例えば、`SYSIN`) です)。以下に、その例を示します。

```
SYSIN is Names-Input
```

- `export` コマンドを使用して、環境変数 `CONSOLE` を有効なファイル指定に設定します。以下に、その例を示します。

```
export CONSOLE=/myfiles/myinput.rpt
```

環境変数名は、使用されているシステム装置名と同じものでなければなりません。上記の例では、システム装置は `Console` ですが、環境変数をシステム装置名に割り当てる方法は、すべての有効なシステム装置に対してサポートされます。例えば、システム装置が `SYSIN` の場合は、ファイル指定に割り当てなければならない環境変数も `SYSIN` となります。

`ACCEPT` ステートメントは、入力行をデータ項目に割り当てます。入力行がデータ項目より短い場合は、適切な表現のスペースがデータ項目に埋め込まれます。画面から読み取るとき、入力行がデータ項目より長いと、残った文字は廃棄されます。ファイルから読み取るとき、入力行がデータ項目より長いと、残った文字はファイルの次の入力行として保存されます。

`ACCEPT` ステートメントを使用するなら、値を英数字または国別グループ項目に割り当てたり、`USAGE DISPLAY`、`USAGE DISPLAY-1`、または `USAGE NATIONAL` が指定されている基本データ項目に割り当てたりすることができます。

値を `USAGE NATIONAL` データ項目に割り当てると、入力データは、現在のランタイム・ロケールに関連付けられているコード・ページから国別 (Unicode UTF-16) 表現に変換されます。ただし、これは、端末からの入力の場合に限られます。

入力データが他の装置からのものであるときに変換を実行させるには、`NATIONAL-OF` 組み込み関数を使用します。

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

225 ページの『環境変数の設定』

198 ページの『英数字または DBCS から国別への変換 (NATIONAL-OF)』

365 ページの『CICS のもとでのシステム日付の取得』

関連参照

`ACCEPT` ステートメント (*COBOL for AIX* 言語解説書)

`SPECIAL-NAMES` 段落 (*COBOL for AIX* 言語解説書)

画面上またはファイル内での値の表示 (DISPLAY)

DISPLAY ステートメントを使用すると、データ項目の値を画面に表示したり、ファイルに書き込むことができます。

```
Display "No entry for surname '" Customer-Name "' found in the file."
```

上記の例で、データ項目 *Customer-Name* の内容が JOHNSON の場合、ステートメントは、次のメッセージを画面に表示します。

```
No entry for surname 'JOHNSON' found in the file.
```

データを画面以外の宛先に書き込みたい場合には、UPON 句を使用してください。例えば、次のステートメントは、SYSOUT 環境変数の値として指定したファイルに書き込みを行います。

```
Display "Hello" upon sysout.
```

USAGE NATIONAL データ項目の値を表示するとき、出力データは、現在のロケールに関連付けられているコード・ページに変換されます。

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

199 ページの『国別の英数字への変換 (DISPLAY-OF)』

364 ページの『CICS のもとで実行する COBOL プログラムのコーディング』

関連参照

228 ページの『ランタイム環境変数』

DISPLAY ステートメント (*COBOL for AIX* 言語解説書)

組み込み関数の使用 (組み込み関数)

一部の高水準プログラム言語には組み込み関数があります。組み込み関数は、プログラム内で、定義済み属性および事前定義値を持つ変数であるかのように参照することができます。COBOL では、これらの関数は **組み込み関数 (intrinsic functions)** と呼ばれます。これらの関数は文字列と数値を操作する機能を提供します。

組み込み関数の値は参照時に自動的に導き出されるため、関数を DATA DIVISION で定義する必要はありません。引数として使用する非リテラル・データ項目だけを定義してください。形象定数を引数として使用することはできません。

関数 ID は、COBOL 予約語 FUNCTION と、それに続く関数名 (Max など) と、それに続く関数の評価に使用される任意の引数 (x、y、z など) の組み合わせです。例えば、強調表示された語句のグループは関数 ID です。

```
Unstring Function Upper-case(Name) Delimited By Space  
      Into Fname Lname  
Compute A = 1 + Function Log10(x)  
Compute M = Function Max(x y z)
```

関数 ID は、関数の呼び出しと、関数によって戻されるデータ値の両方を表します。関数 ID は、実際にデータ項目を表すため、戻り値の属性を持つデータ項目が使用できる場所ならば、PROCEDURE DIVISION のほとんどの場所で使用することができます。

COBOL ワード `function` は予約語ですが、関数名は予約されていません。このため、関数名を他のコンテキスト（データ項目の名前など）で使用することができます。例えば、`Sqrt` は、組み込み関数を呼び出し、プログラム内のデータ項目を指定するために使用できます。

```
Working-Storage Section.  
01 x          Pic 99 value 2.  
01 y          Pic 99 value 4.  
01 z          Pic 99 value 0.  
01 Sqrt       Pic 99 value 0.  
.  
.  
    Compute Sqrt = 16 ** .5  
    Compute z = x + Function Sqrt(y)  
.  
.  
.
```

関数 ID は、英数字、国別、数字、または整数のいずれかのタイプの値を表します。英数字関数または国別関数の関数 ID には、サブstring指定（参照修飾子）を組み込むことができます。数字組み込み関数は、それらが戻す数値のタイプに応じてさらに分類されます。

関数 `MAX`、`MIN`、`DATEVAL`、および `UNDATE` は、指定された引数の型に応じていずれかのタイプの値を戻します。

関数 `DATEVAL`、`UNDATE`、および `YEARWINDOW` は、ウィンドウ表示日付フィールドの操作および変換を援助するために、2000 年言語拡張として提供されています。

関数は、ネストされた関数の結果が外側の関数の引数についての要件を満たす限り、引数として他の関数を参照することができます。例えば、`Function Sqrt(5)` は数値を戻します。したがって、下記の `MAX` 関数への 3 つの引数はすべて、この関数についての許容される引数型である数値になります。

```
Compute x = Function Max((Function Sqrt(5)) 2.5 3.5)
```

関連タスク

- 83 ページの『組み込み関数を使用したテーブル項目の処理』
- 111 ページの『データ項目の変換（組み込み関数）』
- 114 ページの『データ項目の評価（組み込み関数）』

テーブル (配列) とポインターの使用

COBOL では、配列はテーブルと呼ばれます。テーブルは、`OCCURS` 節を使用して `DATA DIVISION` に定義される論理的に連続するデータ項目の集合です。

ポインターは、仮想記憶アドレスを含むデータ項目です。ポインターは、`USAGE IS POINTER` 節を使用して `DATA DIVISION` の中に明示的に定義するか、または `ADDRESS OF` 特殊レジスターとして暗黙的に定義します。

ポインター・データ項目を使用して以下の操作を実行することができます。

- `CALL . . BY REFERENCE` ステートメントを使用してプログラム間でポインター・データ項目の受け渡しを行う。
- `SET` ステートメントを使用してそれらを他のポインターへ移動する。
- 比較条件を使用して他のポインターと比較し、等しいかどうかを調べる。
- `VALUE IS NULL` を使用して、それらが無効なアドレスを含むように初期化する。

ポインター・データ項目は、以下のことを行うために使用してください。

- 限定された基底アドレッシングの実施。特に、レコード域 (`OCCURS DEPENDING ON` で定義されるために可変位置である) のアドレスを受け渡ししたい場合。
- チェーン・リストの処理。

関連タスク

65 ページの『テーブルの定義 (`OCCURS`)』

526 ページの『プロシージャ・ポインターと関数ポインターの使用』

関連参照

266 ページの『`ADDR`』

第 3 章 数値および算術演算

一般的に、COBOL 数値データは、一連の 10 進数字の桁として表示することができます。ただし、数値項目は、算術符号や通貨記号などの特殊な特性を持つこともできます。

算術演算を効率的に実行できるように数値データを定義、表示、および格納するには、次のようにします。

- 数値データを定義するには、PICTURE 節と、文字 9、+、-、P、S、および V を使用します。
- 数値データを表示するには、PICTURE 節と、MOVE および DISPLAY ステートメントと一緒に編集文字 (Z、コンマ、ピリオドなど) を使用します。
- 数値データの格納方法を制御するには、さまざまな形式を指定した USAGE 節を使用します。
- データ値が適切であるかどうかを妥当性検査するには、数値のクラス・テストを使用します。
- 算術を実行するには、ADD、SUBTRACT、MULTIPLY、DIVIDE、および COMPUTE ステートメントを使用します。
- 必要な通貨記号を指定するには、CURRENCY SIGN 節と適切な PICTURE 文字を使用します。

関連タスク

『数値データの定義』

43 ページの『数値データの表示』

44 ページの『数値データの保管方法の制御』

54 ページの『非互換データの検査 (数値のクラス・テスト)』

54 ページの『算術の実行』

62 ページの『通貨記号の使用』

数値データの定義

数値項目を定義するには、数値の 10 進数の桁数を表すためにデータ記述で文字 9 を指定した PICTURE 節を使用します。英数字データ項目用の X を使用しないでください。

例えば、以下の Count-y は数値データ項目であり、USAGE DISPLAY を持つ外部 10 進数項目 (ゾーン 10 進数項目) です。

```
05 Count-y          Pic 9(4) Value 25.  
05 Customer-name   Pic X(20) Value "Johnson".
```

同様に、国別文字 (UTF-16) を保持する数値データ項目を定義できます。例えば、以下の Count-n は USAGE NATIONAL を持つ外部 10 進数データ (国別 10 進数項目) です。

```
05 Count-n          Pic 9(4) Value 25 Usage National.
```

デフォルトのコンパイラ・オプションである ARITH(COMPAT) (互換モード と呼ばれる) を使用してコンパイルする場合は、PICTURE 節には最大 18 桁までコーディングすることができます。ARITH(EXTEND) (拡張モード と呼ばれる) を使用してコンパイルする場合は、PICTURE 節には最大 31 桁までコーディングすることができます。

それ以外にコーディングできる特殊な意味を持つ文字は、次のとおりです。

- P 先行ゼロまたは後続ゼロを示します。
- S 正または負の符号を示します。
- V 小数点を暗黙指定します。

次の例の s は、値が符号付きであることを意味します。

```
05 Price Pic s99v99.
```

したがって、このフィールドには、正または負の値を格納することができます。v は、暗黙の小数点の位置を示しますが、ストレージ上の位置を占めないの、項目のサイズには含まれません。デフォルトでは s はストレージ上の位置を必要としないので、通常、s は数値項目のサイズに含まれません。

しかし、プログラムまたはデータを別のマシンに移植する予定である場合、ゾーン 10 進数データ項目用の符号をストレージ上の別個の位置としてコーディングすることができます。次の場合、符号は 1 バイトを占めます。

```
05 Price Pic s99V99 Sign Is Leading, Separate.
```

このようにすれば、現在使用中のマシンとは非分離符号を格納するための規則が異なるマシンを使用する場合に、予測外の結果が生じることがなくなります。

分離符号は、印刷または表示されるゾーン 10 進数データ項目にとっても望ましいものです。

分離符号は、符号付きの国別 10 進数データ項目には必要です。符号は、以下の例のように 2 バイトのストレージを占有します。

```
05 Price Pic s99V99 Usage National Sign Is Leading, Separate.
```

PICTURE 節に内部浮動小数点データ (COMP-1 または COMP-2) を指定することはできません。しかし、VALUE 節を使用して、内部浮動小数点リテラルの初期値を提供できます。

```
05 Compute-result Usage Comp-2 Value 06.23E-24.
```

外部浮動小数点データについては、以下に参照されている例および数値データの形式に関する関連概念を参照してください。

49 ページの『例: 数値データおよび内部表現』

関連概念

45 ページの『数値データの形式』

631 ページの『付録 D. 中間結果および算術精度』

関連タスク

43 ページの『数値データの表示』

- 44 ページの『数値データの保管方法の制御』
- 54 ページの『算術の実行』
- 192 ページの『国別数値データ項目の定義』

関連参照

- 53 ページの『ゾーンおよびパック 10 進数データのサイン表記』
- 196 ページの『文字データの保管』
- 268 ページの『ARITH』
- SIGN 節 (COBOL for AIX 言語解説書)

数値データの表示

数値項目を特定の編集記号 (小数点、コンマ、ドル記号、借方記号、貸方記号など) を付けて定義すると、項目の表示または印刷時に、項目をより見やすく理解しやすいようにすることができます。

例えば、以下のコードの Edited-price は、USAGE DISPLAY を持つ数字編集項目です。(数字編集項目に節 USAGE IS DISPLAY を指定することができますが、これは暗黙の指定です。これは、項目が文字形式で保管されることを意味します。)

```
05 Price          Pic      9(5)v99.  
05 Edited-price  Pic  $zz,zz9.99.  
...  
Move Price To Edited-price  
Display Edited-price
```

Price の内容が 0150099 (値 1,500.99 を表す) である場合は、コードを実行すると \$ 1,500.99 が表示されます。Edited-price の PICTURE 節内の z は、先行ゼロの抑止を示します。

英数字ではなく国別 (UTF-16) 文字を保持する数字編集データ項目を定義できます。そのためには、数字編集項目を USAGE NATIONAL と宣言してください。USAGE NATIONAL を持つ数字編集項目の場合の編集記号の効果は、USAGE DISPLAY を持つ数字編集項目の場合と同様ですが、編集が国別文字で行われる点は異なります。例えば、上記のコードで Edited-price が USAGE NATIONAL と宣言された場合、項目は国別文字を使用して編集および表示されます。

基本数値項目または数字編集項目に BLANK WHEN ZERO 節をコーディングすることにより、値ゼロが項目に保管されたとき、その項目がスペースで充てんされるようにすることができます。例えば、以下のそれぞれの DISPLAY ステートメントの場合、ゼロではなくブランクが表示されます。

```
05 Price          Pic      9(5)v99.  
05 Edited-price-D Pic  $99,999.99  
                  Blank When Zero.  
05 Edited-price-N Pic  $99,999.99 Usage National  
                  Blank When Zero.  
...  
Move 0 to Price  
Move Price to Edited-price-D  
Move Price to Edited-price-N  
Display Edited-price-D  
Display Edited-price-N
```

算術式の中、あるいは ADD、SUBTRACT、MULTIPLY、DIVIDE、または COMPUTE ステートメントの中で、数字編集項目を送信オペランドとして使用することはできません。(これらのステートメントのいずれかで数字編集項目が受信フィールドであるとき、あるいは MOVE ステートメントが数字編集受信フィールド、および数字編集または数値送信フィールドを持っているとき、数字編集が行われます)。数字編集項目は、主として、数値データの表示または印刷のために使用されます。

数字編集項目は、数値項目または数字編集項目に移動することができます。以下の例では、数字編集項目の値 (USAGE DISPLAY を持っているか USAGE NATIONAL を持っているかにかかわらず) は数値項目に移動します。

```
Move Edited-price to Price
Display Price
```

上記の最初の例のステートメントの直後にこれら 2 つのステートメントが続いている場合、Price は 0150099 (値 1,500.99 を表す) と表示されます。Edited-price が USAGE NATIONAL を持っている場合にも、Price は 0150099 と表示されます。

数字編集項目を、英数字データ項目、英数字編集データ項目、浮動小数点データ項目、および国別データ項目に移動することもできます。数字編集データの有効な受信項目の完全なリストについては、MOVE ステートメントに関する関連した解説書を参照してください。

49 ページの『例: 数値データおよび内部表現』

関連タスク

37 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

『数値データの保管方法の制御』

41 ページの『数値データの定義』

54 ページの『算術の実行』

192 ページの『国別数値データ項目の定義』

197 ページの『国別 (Unicode) 表現との間の変換』

関連参照

MOVE ステートメント (COBOL for AIX 言語解説書)

BLANK WHEN ZERO 節 (COBOL for AIX 言語解説書)

数値データの保管方法の制御

データ記述記入項目に USAGE 節をコーディングすることによって、コンピューターが数値データを保管する方法を制御することができます。

次のような理由から、形式を制御する場合があります。

- 計算データ型を使用して実行する算術の方が、USAGE DISPLAY や USAGE NATIONAL データ型を使用して実行する算術より効率的である。
- パック 10 進数形式の方が、USAGE DISPLAY または USAGE NATIONAL データ型に比べ、1 桁当たりのストレージが少なく済む。
- パック 10 進数形式の方が 2 進数形式の場合よりも、DISPLAY または NATIONAL 形式との変換が効率的である。

- 浮動小数点形式は、位取りが大きく変化するような算術オペランドおよび結果を格納するのに最適であり、最大数の有効数字が維持される。
- データをあるマシンから別のマシンに移すときに、データ形式を保持する必要がある。

プログラム内で使用する数値データは、COBOL で使用可能な以下の形式のいずれかです。

- 外部 10 進数 (USAGE DISPLAY または USAGE NATIONAL)
- 外部浮動小数点 (USAGE DISPLAY または USAGE NATIONAL)
- 内部 10 進数 (USAGE PACKED-DECIMAL)
- 2 進数 (USAGE BINARY)
- 固有 2 進数 (USAGE COMP-5)
- 内部浮動小数点 (USAGE COMP-1 または USAGE COMP-2)

COMP および COMP-4 は BINARY と同義であり、COMP-3 は PACKED-DECIMAL と同義です。

コンパイラーは、表示可能な数値をそれらの数値の内部表現に変換してから、それらを算術演算で使用します。したがって、データ項目を DISPLAY や NATIONAL ではなく BINARY または PACKED-DECIMAL と定義すると、さらに効率的になることがよくあります。以下に例を示します。

```
05 Initial-count Pic S9(4) Usage Binary Value 1000.
```

どの USAGE 節を使用して値の内部表現を制御するかにかかわらず、使用する PICTURE 節の規則および VALUE 節の 10 進値は同じです (ただし、内部浮動小数点データの場合は別で、この場合、PICTURE 節を使用できません)。

49 ページの『例: 数値データおよび内部表現』

関連概念

『数値データの形式』

51 ページの『データ形式の変換』

631 ページの『付録 D. 中間結果および算術精度』

関連タスク

41 ページの『数値データの定義』

43 ページの『数値データの表示』

54 ページの『算術の実行』

関連参照

52 ページの『変換および精度』

53 ページの『ゾーンおよびパック 10 進数データのサイン表記』

数値データの形式

数値データに使用できる形式にはいくつかあります。

外部 10 進数 (DISPLAY および NATIONAL) 項目

カテゴリ数値データ項目で USAGE DISPLAY が (コーディングされているため、あるいはデフォルトにより) 有効である場合、ストレージのそれぞれの位置 (バイト) は 1 つの 10 進数字を含みます。項目は表示可能な形式で保管されます。USAGE DISPLAY を持つ外部 10 進数項目は、ゾーン 10 進数 データ項目と呼ばれます。

カテゴリ数値データ項目で USAGE NATIONAL が有効である場合、それぞれの 10 進数字ごとに 2 バイトのストレージが必要です。項目は UTF-16 形式で保管されます。USAGE NATIONAL を持つ外部 10 進数項目は、国別 10 進数 データ項目と呼ばれます。

国別 10 進数データ項目は、符号付きの場合には、SIGN SEPARATE 節が有効になっている必要があります。ゾーン 10 進数項目のその他の規則すべてが国別 10 進数項目に適用されます。国別 10 進数項目は、他のカテゴリ数値データ項目を使用できる場所ならどこでも使用できます。

外部 10 進数 (ゾーン 10 進数と国別 10 進数の両方) データ項目は、プログラムとファイル、端末、またはプリンターとの間で数値をやり取りすることを主な目的としています。外部 10 進数項目は、算術処理で、オペランドおよび受け取り側として使用することもできます。ただし、プログラムで多数の算術計算を集中的に実行し、効率を優先させるのであれば、算術計算で使用するデータ項目には、COBOL の計算数値タイプを使用した方がよい場合もあります。

外部浮動小数点 (DISPLAY および NATIONAL) 項目

浮動小数点データ項目で USAGE DISPLAY が (コーディングされているため、あるいはデフォルトにより) 有効である場合、それぞれの PICTURE 文字位置 (使用されている場合、暗黙の小数点である v を除く) は 1 バイトのストレージを占有します。項目は表示可能な形式で保管されます。USAGE DISPLAY を持つ外部浮動小数点項目は、USAGE NATIONAL を持つ外部浮動小数点項目と区別する必要があるとき、本書では表示浮動小数点 データ項目と呼ばれます。

以下の例では、Compute-Result が暗黙的に表示浮動小数点項目として定義されています。

```
05 Compute-Result Pic -9v9(9)E-99.
```

負符号 (-) は、仮数および指数が必ず負の数値でなければならないことを意味するものではありません。負符号は、数値が表示されるときに、正数であれば符号がブランクとなり、負数であれば符号が負符号となることを意味しています。正符号 (+) をコーディングすると、符号は、正数であれば正符号となり、負数であれば負符号となります。

浮動小数点データ項目で USAGE NATIONAL が有効である場合、それぞれの PICTURE 文字位置 (使用されている場合、v を除く) は 2 バイトのストレージを占有します。項目は国別文字 (UTF-16) として保管されます。USAGE NATIONAL を持つ外部浮動小数点項目は、国別浮動小数点 データ項目と呼ばれます。

表示浮動小数点項目の既存の規則は、国別浮動小数点項目に適用されます。

以下の例では、Compute-Result-N は国別浮動小数点項目です。

```
05 Compute-Result-N Pic -9v9(9)E-99 Usage National.
```

Compute-Result-N が表示される場合、Compute-Result について上述したように符号が表示されますが、国別文字で表示されます。

外部浮動小数点項目に VALUE 節を使用することはできません。

浮動小数点数は、外部 10 進数と同様、(コンパイラーによって) 数値の内部表現に変換しなければ、算術演算で使用することはできません。デフォルト・オプションの ARITH (COMPAT) を使用してコンパイルした場合、外部浮動小数点数は長精度 (64 ビット) の浮動小数点形式に変換されます。ただし、ARITH (EXTEND) を使用してコンパイルすれば、外部浮動小数点数は拡張精度 (128 ビット) の浮動小数点形式に変換されます。

2 進数 (COMP) 項目

BINARY、COMP、および COMP-4 は同義語です。2 進数形式の数値は、2、4、または 8 バイトのストレージを占めます。PICTURE 節で項目が符号付きであることが指定されている場合は、左端ビットが演算符号として使用されます。

2 進数は、付随する PICTURE 記述が 4 個以下の 10 進数字であれば 2 バイトを占め、5 個から 9 個の 10 進数字であれば 4 バイトを占め、10 個から 18 個の 10 進数字であれば 8 バイトを占めます。9 桁以上の 2 進数項目には、コンパイラーによる余分な処理が必要となります。

2 進数項目には、例えば、指標、添え字、スイッチ、および算術オペランドや結果を入れることができます。

2 進データ (BINARY、COMP、または COMP-4) の切り捨て方法を指定するには、TRUNC(STD|OPT|BIN) コンパイラー・オプションを使用してください。

固有 2 進数 (COMP-5) 項目

USAGE COMP-5 として宣言したデータ項目は、ストレージ内では 2 進データとして表されます。しかし、これらは、USAGE COMP 項目とは異なり、PICTURE 節の 9 の数で暗黙指定される値に制限されるのではなく、固有 2 進数表現 (2、4、または 8 バイト) の容量までの大きさの値を含むことができます。

数値データを COMP-5 項目に移動または保管すると、COBOL PICTURE サイズ制限ではなく、2 進数フィールド・サイズで切り捨てが行われます。COMP-5 項目を参照する場合、演算では完全な 2 進数フィールド・サイズが使用されます。

したがって、COMP-5 は、データが COBOL PICTURE 節に適合しない可能性のある非 COBOL プログラムから生じる 2 進データ項目の場合に特に有用です。

次の表は、COMP-5 データ項目に指定可能な値の範囲を示しています。

表 3. COMP-5 データ項目の値の範囲

PICTURE	ストレージ表現	数値
S9(1) から S9(4)	2 進数ハーフワード (2 バイト)	-32768 から +32767
S9(5) から S9(9)	2 進数フルワード (4 バイト)	-2,147,483,648 から +2,147,483,647
S9(10) から S9(18)	2 進数ダブルワード (8 バイト)	-9,223,372,036,854,775,808 から +9,223,372,036,854,775,807
9(1) から 9(4)	2 進数ハーフワード (2 バイト)	0 から 65535
9(5) から 9(9)	2 進数フルワード (4 バイト)	0 から 4,294,967,295
9(10) から 9(18)	2 進数ダブルワード (8 バイト)	0 から 18,446,744,073,709,551,615

COMP-5 項目の PICTURE 節に、スケーリング (すなわち、小数部の桁数または暗黙の整数桁数) を指定することができます。その場合は、上記にリストされた最大容量とほぼ同じ大きさをスケーリングする必要があります。例えば、PICTURE S99V99 COMP-5 として記述したデータ項目は、ストレージ内では 2 進数ハーフワードとして表され、-327.68 から +327.67 までの範囲の値をサポートします。

VALUE 節のラージ・リテラル: COMP-5 項目の VALUE 節に指定されたリテラルは、一部の例外を除いて、固有 2 進数表現の容量までの大きさの値を含むことができます。例外については、*COBOL for AIX 言語解説書*を参照してください。

TRUNC コンパイラー・オプションの設定に関係なく、COMP-5 データ項目は、TRUNC(BIN) でコンパイルされたプログラムでは、2 進データのように動作します。

パック 10 進数 (COMP-3) 項目

PACKED-DECIMAL と COMP-3 は同義語です。パック 10 進数項目は、PICTURE 記述でコーディングされる 2 つの 10 進数字ごとに 1 バイトのストレージを占めます。ただし、右端のバイトだけが例外で、右端のバイトには 1 つの数字と符号が入ります。この形式が最も効率的に使用されるのは、PICTURE 記述で奇数の桁をコーディングして、左端のバイトが完全に使用されるようにするときです。パック 10 進数項目は、算術演算の目的では固定小数点数として扱われます。

内部浮動小数点 (COMP-1 および COMP-2) 項目

COMP-1 は短精度浮動小数点形式を指し、COMP-2 は長精度浮動小数点形式を指します。これらの形式は、それぞれ 4 バイトと 8 バイトのストレージを占めます。

COMP-1 および COMP-2 データ項目は、FLOAT(NATIVE) コンパイラー・オプション (デフォルト) が有効である場合、IEEE 形式で表されます。FLOAT(S390) (またはその同義語 FLOAT(HEX)) が有効な場合は、COMP-1 および COMP-2 データ項目が一貫

して zSeries[®]、つまり 16 進数の浮動小数点形式で表されます。詳細については、下記の FLOAT オプションの説明を参照してください。

関連概念

188 ページの『Unicode および言語文字のエンコード』
631 ページの『付録 D. 中間結果および算術精度』

関連タスク

41 ページの『数値データの定義』
192 ページの『国別数値データ項目の定義』

関連参照

196 ページの『文字データの保管』
312 ページの『TRUNC』
289 ページの『FLOAT』
データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)
SIGN 節 (COBOL for AIX 言語解説書)
VALUE 節 (COBOL for AIX 言語解説書)

例: 数値データおよび内部表現

次の表は、数値項目の内部表現を示しています。

次の表は、バイナリー・データ・タイプの数値項目の内部表現を示しています。

表 4. 2 進数値項目の内部表現

数値タイプ	PICTURE および USAGE 節とオプションの SIGN 節	値	内部表現
2 進数	PIC S9999 BINARY	+ 1234	04 D2
	PIC S9999 COMP	- 1234	FB 2E
	PIC S9999 COMP-4		
	PIC S9999 COMP-5	+ 12345 ¹	30 39
		- 12345 ¹	CF C7
	PIC 9999 BINARY	1234	04 D2
PIC 9999 COMP			
PIC 9999 COMP-4			
PIC 9999 COMP-5	60000 ¹	EA 60	

1. この例では、COMP-5 データ項目に含めることのできる値が、PICTURE 節の 9 の数によって暗黙指定された値に制限されるのではなく、固有 2 進数表現 (2、4、または 8 バイト) の容量までの大きさの値を入れることができることを示しています。

次の表は、固有データ形式の数値項目の内部表現を示しています。CHAR(NATIVE) および FLOAT(NATIVE) コンパイラー・オプションが有効であるとします。

表 5. 固有数値項目の内部表現

数値タイプ	PICTURE および USAGE 節とオプションの SIGN 節	値	内部表現
外部 10 進数	PIC S9999 DISPLAY	+ 1234	31 32 33 34
		- 1234	31 32 33 74
		1234	31 32 33 34
	PIC 9999 DISPLAY	1234	31 32 33 34
	PIC 9999 NATIONAL	1234	00 31 00 32 00 33 00 34
	PIC S9999 DISPLAY SIGN LEADING	+ 1234	31 32 33 34
		- 1234	71 32 33 34
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+ 1234	2B 31 32 33 34
		- 1234	2D 31 32 33 34
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+ 1234	31 32 33 34 2B
		- 1234	31 32 33 34 2D
	PIC S9999 NATIONAL SIGN LEADING SEPARATE	+ 1234	00 2B 00 31 00 32 00 33 00 34
		- 1234	00 2D 00 31 00 32 00 33 00 34
	PIC S9999 NATIONAL SIGN TRAILING SEPARATE	+ 1234	00 31 00 32 00 33 00 34 00 2B
- 1234		00 31 00 32 00 33 00 34 00 2D	
内部 10 進数	PIC S9999 PACKED-DECIMAL PIC S9999 COMP-3	+ 1234	01 23 4C
		- 1234	01 23 4D
	PIC 9999 PACKED-DECIMAL PIC 9999 COMP-3	1234	01 23 4C
内部浮動小数点	COMP-1	+ 1234	44 9A 40 00
		- 1234	C4 9A 40 00
	COMP-2	+ 1234	40 93 48 00 00 00 00 00
		- 1234	C0 93 48 00 00 00 00 00
外部浮動小数点	PIC +9(2).9(2)E+99 DISPLAY	+ 12.34E+02	2B 31 32 2E 33 34 45 2B 30 32
		- 12.34E+02	2D 31 32 2E 33 34 45 2B 30 32
	PIC +9(2).9(2)E+99 NATIONAL	+ 12.34E+02	00 2B 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32
		- 12.34E+02	00 2D 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32

次の表は、zSeries ホスト・データ形式の数値項目の内部表現を示しています。
 CHAR(EBCDIC) および FLOAT(HEX) コンパイラ・オプションが有効であるとし
 ます。

表 6. CHAR(EBCDIC) および FLOAT(HEX) が有効であるときの数値項目の内部表現

数値タイプ	PICTURE および USAGE 節とオプションの SIGN 節	値	内部表現
外部 10 進数	PIC S9999 DISPLAY	+ 1234	F1 F2 F3 C4
		- 1234	F1 F2 F3 D4
		1234	F1 F2 F3 C4
	PIC 9999 DISPLAY	1234	F1 F2 F3 F4
	PIC 9999 NATIONAL	1234	00 31 00 32 00 33 00 34
	PIC S9999 DISPLAY SIGN LEADING	+ 1234	C1 F2 F3 F4
		- 1234	D1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+ 1234	4E F1 F2 F3 F4
		- 1234	60 F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+ 1234	F1 F2 F3 F4 4E
		- 1234	F1 F2 F3 F4 60
	PIC S9999 NATIONAL SIGN LEADING SEPARATE	+ 1234	00 2B 00 31 00 32 00 33 00 34
- 1234		00 2D 00 31 00 32 00 33 00 34	
PIC S9999 NATIONAL SIGN TRAILING SEPARATE	+ 1234	00 31 00 32 00 33 00 34 00 2B	
	- 1234	00 31 00 32 00 33 00 34 00 2D	
内部 10 進数	PIC S9999 PACKED-DECIMAL PIC S9999 COMP-3	+ 1234	01 23 4C
		- 1234	01 23 4D
	PIC 9999 PACKED-DECIMAL PIC 9999 COMP-3	1234	01 23 4C
内部浮動小数点	COMP-1	+ 1234	43 4D 20 00
		- 1234	C3 4D 20 00
	COMP-2	+ 1234	43 4D 20 00 00 00 00 00
		- 1234	C3 4D 20 00 00 00 00 00
外部浮動小数点	PIC +9(2).9(2)E+99 DISPLAY	+ 12.34E+02	4E F1 F2 4B F3 F4 C5 4E F0 F2
		- 12.34E+02	60 F1 F2 4B F3 F4 C5 4E F0 F2
	PIC +9(2).9(2)E+99 NATIONAL	+ 12.34E+02	00 2B 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32
		- 12.34E+02	00 2D 00 31 00 32 00 2E 00 33 00 34 00 45 00 2B 00 30 00 32

データ形式の変換

プログラム内のコードがデータ形式の異なる項目の相互作用を含んでいると、コンパイラーはそれらの項目を一時的に (比較および算術演算の場合) または永続的に (MOVE、COMPUTE、またはその他の算術ステートメントの受け取り側への割り当ての場合) 変換します。

可能であれば、コンパイラーは、直接的な 1 桁ずつの移動ではなく、数値を保持する移動を実行します。

変換には、通常、追加のストレージと処理時間が必要とされます。これは、演算が実行される前に、データが内部作業域に移動され、変換されるためです。さらに、結果を作業域に戻し、再度変換することが必要な場合もあります。

固定小数点データ形式 (外部 10 進数、パック 10 進数、または 2 進数) 間の変換は、ターゲット・フィールドがソース・オペランドのすべての桁を含むことができれば、精度が失われることなく完了します。

固定小数点データ形式と浮動小数点データ形式 (短精度浮動小数点、長精度浮動小数点、または外部浮動小数点) 間の変換では、精度が失われる可能性があります。このような変換は、固定小数点と浮動小数点の両方のオペランドが混在する算術計算時に起こります。

関連参照

『変換および精度』

53 ページの『ゾーンおよびパック 10 進数データのサイン表記』

変換および精度

数値変換によっては精度が低下する場合があるほか、精度が保持されたり、丸めが行われる場合もあります。

固定小数点項目と外部浮動小数点項目は、どちらも 10 進数の特性を持つため、以下の例での固定小数点項目への参照は、特に明記しない限り、外部浮動小数点項目への参照を含みます。

コンパイラーが固定小数点形式を内部浮動小数点形式に変換するときには、基数 10 の固定小数点数は、内部で使用される数体系に変換されます。

コンパイラーが比較のために短精度形式を長精度形式に変換するときには、短精度数値の埋め込みにはゼロが使用されます。

精度が低下する変換

USAGE COMP-1 データ項目が 6 桁を超える固定小数点データ項目に移動されるときには、固定小数点データ項目は有効数字を 6 個だけ受け取り、残りの桁は 0 になります。

精度を保つ変換

6 桁以下の固定小数点データ項目が USAGE COMP-1 データ項目に移動され、その後で固定小数点データ項目に戻される場合は、元の値がリカバリーされます。

USAGE COMP-1 データ項目が 6 桁以上の固定小数点データ項目に移動され、その後で USAGE COMP-1 データ項目に戻される場合は、元の値がリカバリーされます。

15 桁以下の固定小数点データ項目が USAGE COMP-2 データ項目に移動され、その後で固定小数点データ項目に戻される場合は、元の値がリカバリーされます。

USAGE COMP-2 データ項目が 18 桁以上の固定小数点 (外部浮動小数点ではなく) データ項目に移動され、その後で USAGE COMP-2 データ項目に戻される場合は、元の値がリカバリーされます。

丸めを生じさせる変換

USAGE COMP-1 データ項目、USAGE COMP-2 データ項目、外部浮動小数点データ項目、または浮動小数点リテラルが固定小数点データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

USAGE COMP-2 データ項目が USAGE COMP-1 データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

固定小数点データ項目の PICTURE に外部浮動小数点データ項目の PICTURE よりも多くの桁位置が含まれている場合に、固定小数点データ項目が外部浮動小数点データ項目に移動されると、ターゲット・データ項目の低位桁で丸めが起こります。

関連概念

631 ページの『付録 D. 中間結果および算術精度』

ゾーンおよびパック 10 進数データのサイン表記

サイン表記は、ゾーン 10 進数データおよび内部 10 進数データの処理や相互作用に影響します。

X'sd' (ここで *s* はサイン表記であり、*d* は数字を表します) が与えられた場合、SIGN IS SEPARATE 節を持たない ゾーン 10 進数 (USAGE DISPLAY) データの有効なサイン表記は次のとおりです。

正: 0、1、2、3、8、9、A、および B

負: 4、5、6、7、C、D、E、および F

CHAR(NATIVE) コンパイラー・オプションが有効であると、内部生成される符号は、正および符号なしの場合で 3、負の場合で 7 になります。

CHAR(EBCDIC) コンパイラー・オプションが有効であると、内部生成される符号は、正の場合で C、符号なしの場合で F、負の場合で D になります。

X'ds' (*d* は数字を表し、*s* はサイン表記です) が与えられた場合、内部 10 進数 (USAGE PACKED-DECIMAL) データの有効なサイン表記は次のとおりです。

正: A、C、E、および F

負: B および D

内部生成される符号は、正および符号なしの場合で C、負の場合で D になります。

符号なし内部 10 進数のサイン表記は、COBOL for AIX とホスト COBOL とでは異なります。ホスト COBOL は、符号なし内部 10 進数の符号として F を内部生成します。

関連参照

318 ページの『ZWB』

620 ページの『データ表現』

非互換データの検査 (数値のクラス・テスト)

コンパイラーは、データ項目に指定された値を PICTURE および USAGE 節に有効であると見なし、それらの値の妥当性検査を行いません。データ項目を後続の処理で使用する前に、その内容が PICTURE および USAGE 節に適合していることを確認してください。

値がプログラムに渡され、それらの値に対する互換性のないデータ記述を持つ項目に割り当てられることがよくあります。例えば、非数値データが、数値として定義されたフィールドに移動されたり、渡されたりすることがあります。また、符号付き数値が、符号なしとして定義されたフィールドに渡されることもあります。いずれの場合も、受け取り側フィールドには無効なデータが含まれます。項目にそのデータ記述と互換性のない値が与えられると、PROCEDURE DIVISION 内でのその項目への参照が未定義になり、結果が予測できなくなります。

数値のクラス・テストを使用して、データ妥当性検査を行うことができます。以下に例を示します。

```
Linkage Section.  
01 Count-x Pic 999.  
...  
Procedure Division Using Count-x.  
    If Count-x is numeric then display "Data is good"
```

数値のクラス・テストでは、データ項目の内容が、そのデータ項目の PICTURE および USAGE にとって有効な値のセットと照合されます。

算術の実行

いくつかの COBOL 言語機能 (COMPUTE、算術式、数値組み込み関数、数学呼び出し可能サービス、および日付呼び出し可能サービスを含む) のいずれかを使用して、算術を実行できます。どれを選択するかは、特定の必要を機能が満たすかどうかによって違ってきます。

ほとんどの一般的な算術計算の場合、COMPUTE ステートメントが適切です。数値リテラル、数値データ、または算術演算子を使用する必要がある場合は、算術式を使用できます。数値表現が許可されている場合には、数値組み込み関数を使用すれば時間を節約できます。

関連タスク

『COMPUTE およびその他の算術ステートメントの使用』

55 ページの『算術式の使用』

56 ページの『数値組み込み関数の使用』

COMPUTE およびその他の算術ステートメントの使用

ほとんどの算術計算では、ADD、SUBTRACT、MULTIPLY、および DIVIDE ステートメントではなく、COMPUTE ステートメントが使用されます。いくつかの個々の算術ステートメントの代わりに、1 つの COMPUTE ステートメントをコーディングするだけでよいことがよくあります。

COMPUTE ステートメントは、算術式の結果を 1 つ以上のデータ項目に割り当てます。

```
Compute z      = a + b / c ** d - e
Compute x y z = a + b / c ** d - e
```

COMPUTE 以外の算術ステートメントを使用した算術計算の中には、いっそう直感的なものがあります。以下に例を示します。

COMPUTE	等価の算術ステートメント
Compute Increment = Increment + 1	Add 1 to Increment
Compute Balance = Balance - Overdraft	Subtract Overdraft from Balance
Compute IncrementOne = IncrementOne + 1 Compute IncrementTwo = IncrementTwo + 1 Compute IncrementThree = IncrementThree + 1	Add 1 to IncrementOne, IncrementTwo, IncrementThree

さらに、剰余を処理したい除算については、DIVIDE ステートメント (REMAINDER 句を指定した) を使用したい場合もあります。REM 組み込み関数も、剰余を処理する機能を提供します。

算術計算を実行するとき、ゾーン 10 進数データ項目を使用するのと同様に、国別 10 進数データ項目をオペランドとして使用できます。また、表示浮動小数点オペランドを使用するのと同様に、国別浮動小数点データ項目を使用することもできます。

関連概念

60 ページの『固定小数点演算と浮動小数点演算の対比』

631 ページの『付録 D. 中間結果および算術精度』

関連タスク

41 ページの『数値データの定義』

算術式の使用

数値データ項目が許可されているステートメント内の多くの場所で、算術式を使用できます (ただし、どの場所でも使用できるわけではありません)。

例えば、算術式を比較条件の被比較数として使用することができます。

```
If (a + b) > (c - d + 5) Then. . .
```

算術式は、単一の数字リテラル、単一の数値データ項目、または単一の組み込み関数参照で構成することができます。また、これらの項目のいくつかを算術演算子で結合して構成することもできます。

算術演算子は、次の優先順位に従って評価されます。

表 7. 算術演算子の評価の順序

演算子	意味	評価の順序
単項 + または -	代数符号	1 番目
**	指数	2 番目
/ または *	除算または乗算	3 番目
2 項 + または -	加算または減算	最後

優先順位が同じレベルの演算子は、左から右へと評価されます。ただし、演算子とともに括弧を使用して、それらが評価される順序を変更することができます。括弧の中の式は、個々の演算子が評価される前に評価されます。必要であるかどうかにかかわらず、括弧を使用するとプログラムが読みやすくなります。

関連概念

60 ページの『固定小数点演算と浮動小数点演算の対比』

631 ページの『付録 D. 中間結果および算術精度』

数字組み込み関数の使用

数字組み込み関数は、数式を使用できる場所でのみ使用することができます。これらの関数を使用すると、時間の節約に役立ちます。これは、これらの関数が取り扱う多種類の一般的な計算をコーディングする必要がなくなるためです。

数字組み込み関数は符号付き数値を戻し、一時数値データ項目として扱われます。

数字関数は、以下のカテゴリーに分類されます。

整数 整数を戻すもの。

浮動小数点

長精度 (64 ビット) または拡張精度 (128 ビット) の浮動小数点値を戻すもの (これは、デフォルト・オプション ARITH(COMPAT) を使用してコンパイルするか、ARITH(EXTEND) を使用してコンパイルするかによって決まります)。

混合 引数によって、整数、浮動小数点値、または小数部の桁がある固定小数点数を戻すもの。

組み込み関数を使用すると、次の表に概説されているようなさまざまな種類の算術演算を実行することができます。

表 8. 数字組み込み関数

数値処理	日付および時刻	金融	数学	統計
LENGTH	CURRENT-DATE	ANNUITY	ACOS	MEAN
MAX	DATE-OF-INTEGERS	PRESENT-VALUE	ASIN	MEDIAN
MIN	DATE-TO-YYYYMMDD		ATAN	MIDRANGE
NUMVAL	DATEVAL		COS	RANDOM
NUMVAL-C	DAY-OF-INTEGERS		FACTORIAL	RANGE
ORD-MAX	DAY-TO-YYYYDDD		INTEGER	STANDARD-DEVIATION
ORD-MIN	INTEGER-OF-DATE		INTEGER-PART	VARIANCE
	INTEGER-OF-DAY		LOG	
	UNDATE		LOG10	
	WHEN-COMPILED		MOD	
	YEAR-TO-YYYY		REM	
	YEARWINDOW		SIN	
			SQRT	
			SUM	
			TAN	

『例: 数字組み込み関数』

ある関数を別の関数の引数として参照することができます。ネストされた関数は、外側の関数からは独立して評価されます (ただし、コンパイラが混合関数を固定小数点命令と浮動小数点命令のどちらを使用して評価すべきかを判別するときは例外です)。

算術式を数字関数への引数としてネストすることもできます。例えば、次の例には、3 つの関数引数 (a、b、および算術式 (c / d)) がありません。

Compute x = Function Sum(a b (c / d))

ALL 添え字を使用すると、あるテーブル (または配列) のすべてのエレメントを関数の引数として参照することができます。

また、整数タイプの特殊レジスタは、整数の引数を使用できるのであればどこでも引数として使用することができます。

関連概念

60 ページの『固定小数点演算と浮動小数点演算の対比』

631 ページの『付録 D. 中間結果および算術精度』

関連参照

268 ページの『ARITH』

例: 数字組み込み関数

以下の例と付随する説明では、それぞれのカテゴリーごとに組み込み関数を示します。

| 以下の例がゾーン 10 進数データ項目を示している場合、代わりに国別 10 進数項目を使用できます。(ただし、符号付き国別 10 進数項目の場合は、SIGN SEPARATE 節が有効でなければなりません。)

一般数値処理

2 つの価格 (ドル記号付きの英数字項目として以下に示されています) のうちの最大値を見つけ、その値を出力レコードの数値フィールドに入れ、それから出力レコードの長さを判別したいとしましょう。そのためには、NUMVAL-C (英数字または国別リテラルあるいは英数字または国別データ項目の、数値を戻す関数)、および MAX 関数と LENGTH 関数を使用できます。

```
01 X                      Pic 9(2).
01 Price1                 Pic x(8)  Value "$8000".
01 Price2                 Pic x(8)  Value "$2000".
01 Output-Record.
   05 Product-Name       Pic x(20).
   05 Product-Number    Pic 9(9).
   05 Product-Price     Pic 9(6).
. . .
Procedure Division.
  Compute Product-Price =
    Function Max (Function Numval-C(Price1) Function Numval-C(Price2))
  Compute X = Function Length(Output-Record)
```

さらに、Product-Name の内容が大文字になるようにするために、次のステートメントを使用することができます。

```
Move Function Upper-case (Product-Name) to Product-Name
```

日付および時刻

次の例は、今日から 90 日後の満期日を計算する方法を示しています。CURRENT-DATE 関数から戻される最初の 8 文字は、日付を 4 桁の年、2 桁の月、および 2 桁の日という形式 (YYYYMMDD) で表します。この日付がその整数値に変換されます。そのあと、この値に 90 が追加され、整数が YYYYMMDD 形式に再度変換されます。

```
01 YYYYMMDD              Pic 9(8).
01 Integer-Form          Pic S9(9).
. . .
  Move Function Current-Date(1:8) to YYYYMMDD
  Compute Integer-Form = Function Integer-of-Date(YYYYMMDD)
  Add 90 to Integer-Form
  Compute YYYYMMDD = Function Date-of-Integer(Integer-Form)
  Display 'Due Date: ' YYYYMMDD
```

金融

ビジネス投資の判断では、計画された投資の利益率を評価するために、予期される将来の現金流入の現在価格を計算することがしばしば必要になります。将来の特定の時期に受け取ることが期待される金額の現在価格は、今日特定の利率で投資された場合に、累積されてその将来の金額になるであろう金額です。

例えば、計画された \$1,000 の投資で、次の 3 年間にわたり、それぞれ年 1 回の支払いで \$100、\$200、および \$300 の支払いの流れになるとします。次の COBOL ステートメントは、10% の利率でこれらの現金流入の現在の値を計算する方法を示しています。

```
01 Series-Amt1           Pic 9(9)V99      Value 100.
01 Series-Amt2           Pic 9(9)V99      Value 200.
01 Series-Amt3           Pic 9(9)V99      Value 300.
01 Discount-Rate        Pic S9(2)V9(6)    Value .10.
```

```

01 Todays-Value      Pic 9(9)V99.
. . .
  Compute Todays-Value =
    関数
    Present-Value(Discount-Rate Series-Amt1 Series-Amt2 Series-Amt3)

```

ANNUITY 関数は、ローンの元金および利息を返済するために必要な分割払いの支払金（年賦金）の金額を判断することが必要とされるビジネス問題で使用できます。一連の支払いの特徴は、各期間の長さ、期間ごとの金額および利率が一定であるということです。次の例は、\$15,000 のローンを 12% の年利で 3 年間で返済するのに必要な月賦金額を計算する方法を示しています（36 か月払い、1 か月当たりの利率 = .12/12）。

```

01 Loan              Pic 9(9)V99.
01 Payment           Pic 9(9)V99.
01 Interest          Pic 9(9)V99.
01 Number-Periods   Pic 99.
. . .
  Compute Loan = 15000
  Compute Interest = .12
  Compute Number-Periods = 36
  Compute Payment =
    Loan * Function Annuity((Interest / 12) Number-Periods)

```

数学

次の COBOL ステートメントでは、組み込み関数をネストし、算術式を引数として使用し、前の複雑な計算を簡単に行う方法を示しています。

```

Compute Z = Function Log(Function Sqrt (2 * X + 1)) + Function Rem(X 2)

```

ここでは、組み込み関数 REM (REMAINDER 節を指定した DIVIDE ステートメントではなく) が、X を 2 で割った剰余を加数に戻します。

統計

組み込み関数を使用すると、統計情報の計算が簡単になります。さまざまな市民税を分析していて、平均値、中央値、および範囲（最高税額と最低税額の差）を計算したいとします。

```

01 Tax-S             Pic 99v999 value .045.
01 Tax-T             Pic 99v999 value .02.
01 Tax-W             Pic 99v999 value .035.
01 Tax-B             Pic 99v999 value .03.
01 Ave-Tax           Pic 99v999.
01 Median-Tax        Pic 99v999.
01 Tax-Range         Pic 99v999.
. . .
  Compute Ave-Tax     = Function Mean (Tax-S Tax-T Tax-W Tax-B)
  Compute Median-Tax = Function Median (Tax-S Tax-T Tax-W Tax-B)
  Compute Tax-Range  = Function Range (Tax-S Tax-T Tax-W Tax-B)

```

関連タスク

113 ページの『数値への変換 (NUMVAL、NUMVAL-C)』

固定小数点演算と浮動小数点演算の対比

プログラム内の算術計算では (それが算術ステートメント、組み込み関数、式、または相互にネストされたこれらの組み合わせのいずれであっても)、算術計算のコーディング方法によって、浮動小数点演算になるか、固定小数点演算になるかが決まります。

プログラム内の多くのステートメントには、算術計算が伴うことがあります。例えば、以下のそれぞれの COBOL ステートメントには、ある種の算術計算が必要です。

- 一般算術計算

```
compute report-matrix-col = (emp-count ** .5) + 1
add report-matrix-min to report-matrix-max giving report-matrix-tot
```

- 式および関数

```
compute report-matrix-col = function sqrt(emp-count) + 1
compute whole-hours      = function integer-part((average-hours) + 1)
```

- 算術比較

```
if report-matrix-col < function sqrt(emp-count) + 1
if whole-hours      not = function integer-part((average-hours) + 1)
```

浮動小数点計算

通常、算術計算に以下のいずれかの特性がある場合、それは浮動小数点演算で評価されます。

- オペランドまたは結果フィールドが浮動小数点である。

オペランドは、浮動小数点リテラルとしてコーディングするか、あるいは USAGE COMP-1、USAGE COMP-2、または外部浮動小数点 (浮動小数点 PICTURE を指定した USAGE DISPLAY または USAGE NATIONAL) として定義されたデータ項目としてコーディングした場合に浮動小数点になります。

オペランドがネストされた算術式である場合、または数字組み込み関数への参照である場合、そのオペランドは次の条件のいずれかが当てはまるとき、浮動小数点演算になります。

- 算術式内の引数が浮動小数点になる。
- 関数が浮動小数点関数である。
- 関数が 1 つ以上の浮動小数点引数を持つ混合関数である。
- 指数に小数部の桁が含まれている。

指数は、小数部の桁を含むか (小数部の桁を含むリテラルを使用する場合)、小数部の桁を含む PICTURE を項目に与えるか、あるいは結果が小数部の桁を持つ算術式または関数を使用します。

算術式または数字関数は、オペランドまたは引数 (除数および指数を除く) に小数部の桁がある場合には、小数部の桁がある結果をもたらします。

固定小数点計算

通常、算術演算に上記の浮動小数点についての特性がない場合、コンパイラーはそれを固定小数点演算で評価します。すなわち、算術計算が固定小数点として処理されるのは、すべてのオペランドが固定小数点で、結果フィールドが固定小数点と定義されており、しかもどの指数も小数部の桁がある値を表さない場合だけです。また、ネストされた算術式および関数参照も、固定小数点値を表さなければなりません。

算術比較 (比較条件)

関係演算子を使用して数式を比較する場合、数式 (それらがデータ項目、算術式、関数参照、またはこれらの組み合わせのいずれであっても) は、計算全体のコンテキストでは、被比較数です。すなわち、それぞれの属性が互いの計算に影響を与える可能性があり、両方の式が固定小数点で評価されるか、または両方の式が浮動小数点で評価されることとなります。これは、簡略比較にも当てはまります (比較の中で一方の被比較数が明示的に指定されない場合でも)。以下に例を示します。

```
if (a + d) = (b + e) and c
```

このステートメントには、 $(a + d) = (b + e)$ と $(a + d) = c$ の 2 つの比較があります。 $(a + d)$ は、2 番目の比較では明示的に指定されていませんが、その比較の被比較数です。したがって、 c の属性が $(a + d)$ の計算に影響を与える可能性があります。

比較演算 (および比較の中にネストされた算術式の評価) は、一方の被比較数が浮動小数点値であるかまたは結果が浮動小数点値になる場合、コンパイラーによって浮動小数点演算として処理されます。

比較演算 (および比較の中にネストされた算術式の評価) は、両方の被比較数が固定小数点値であるかまたは結果が固定小数点値になる場合、コンパイラーによって固定小数点演算として処理されます。

暗黙の比較 (関係演算子が使用されない) は、単位として扱われません。しかし、2 つの被比較数は、浮動小数点演算または固定小数点演算における評価に関しては別々に扱われます。以下の例では、実際には、それぞれの属性に関係なく評価され、その後で相互に比較される 5 つの算術式があります。

```
evaluate (a + d)
  when (b + e) thru c
  when (f / g) thru (h * i)
  . . .
end-evaluate
```

『例: 固定小数点計算および浮動小数点計算』

関連参照

640 ページの『非算術ステートメントの算術式』

例: 固定小数点計算および浮動小数点計算

次の例は、固定小数点演算と浮動小数点演算を使用して評価されるステートメントを示しています。

従業員表のデータ項目を次のように定義すると仮定します。

```
01 employee-table.
   05 emp-count          pic 9(4).
   05 employee-record occurs 1 to 1000 times
      depending on emp-count.
      10 hours          pic +9(5)e+99.
. . .
01 report-matrix-col    pic 9(3).
01 report-matrix-min    pic 9(3).
01 report-matrix-max    pic 9(3).
01 report-matrix-tot    pic 9(3).
01 average-hours       pic 9(3)v9.
01 whole-hours         pic 9(4).
```

以下のステートメントは、浮動小数点演算を使用して評価されます。

```
compute report-matrix-col = (emp-count ** .5) + 1
compute report-matrix-col = function sqrt(emp-count) + 1
if report-matrix-tot < function sqrt(emp-count) + 1
```

以下のステートメントは、固定小数点演算を使用して評価されます。

```
add report-matrix-min to report-matrix-max giving report-matrix-tot
compute report-matrix-max =
    function max(report-matrix-max report-matrix-tot)
if whole-hours not = function integer-part((average-hours) + 1)
```

通貨記号の使用

多くのプログラムでは金融情報を処理する必要があり、それらの情報を適切な通貨記号で出力表示する必要があります。COBOL 通貨サポート (およびご使用のプリンターやディスプレイ装置に合ったコード・ページ) を使用するなら、プログラムでいくつかの通貨記号を使用できます。

以下の記号の 1 つ以上を使用できます。

- ドル記号 (\$) のような記号
- 複数文字からなる通貨記号 (USD または EUR など)
- 欧州経済通貨同盟 (EMU) によって確立されているユーロ記号

金融情報を表示するための記号を指定するには、それらの記号に関連付けられる PICTURE 文字を指定した CURRENCY SIGN 節を (CONFIGURATION SECTION の SPECIAL-NAMES 段落の中で) 使用してください。次の例で、PICTURE 文字 \$ は、通貨記号として \$US を使用することを示しています。

```
    Currency Sign is "$US" with Picture Symbol "$".
. . .
77 Invoice-Amount      Pic $$,$$9.99.
. . .
    Display "Invoice amount is " Invoice-Amount.
```

この例で、Invoice-Amount に 1500.00 が含まれている場合は、次のように出力されます。

```
Invoice amount is $US1,500.00
```

プログラム内で複数の CURRENCY SIGN 節を使用することにより、複数の通貨記号を表示することができます。

16 進リテラルを使用して通貨記号の値を表すことができます。ソース・プログラムのデータ入力方法では、対象とする文字を簡単に入力できない場合、16 進数リテラルを使用すると役立つことがあります。次の例は、通貨記号として使用される 16 進値 X'80' を示しています。

```
        Currency Sign X'80' with Picture Symbol 'U'.
        . . .
01 Deposit-Amount      Pic UUUUU9.99.
```

キーボード上にユーロ記号に相当する文字がない場合は、それを CURRENCY SIGN 節で 16 進値として指定する必要があります。

ユーロ記号の 16 進値は、コード・ページ 1252 (Latin 1) の場合、X'80' です。

関連参照

277 ページの『CURRENCY』

CURRENCY SIGN 節 (「COBOL for AIX 言語解説書」)

例: 複数の通貨符号

次の例は、ユーロ通貨 (EUR) とスイスのフラン (CHF) の両方で値を表示する方法を示すものです。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. EuroSamp.
Environment Division.
Configuration Section.
Special-Names.
    Currency Sign is "CHF " with Picture Symbol "F"
    Currency Sign is "EUR " with Picture Symbol "U".
Data Division.
Working-Storage Section.
01 Deposit-in-Euro      Pic S9999V99 Value 8000.00.
01 Deposit-in-CHF      Pic S99999V99.
01 Deposit-Report.
    02 Report-in-Franc  Pic -FFFFFF9.99.
    02 Report-in-Euro   Pic -UUUUU9.99.
01 EUR-to-CHF-Conv-Rate Pic 9V99999 Value 1.53893.
. . .
PROCEDURE DIVISION.
Report-Deposit-in-CHF-and-EUR.
    Move Deposit-in-Euro to Report-in-Euro
    Compute Deposit-in-CHF Rounded
        = Deposit-in-Euro * EUR-to-CHF-Conv-Rate
    On Size Error
        Perform Conversion-Error
    Not On Size Error
        Move Deposit-in-CHF to Report-in-Franc
        Display "Deposit in euro = " Report-in-Euro
        Display "Deposit in franc = " Report-in-Franc
    End-Compute
    Goback.
Conversion-Error.
    Display "Conversion error from EUR to CHF"
    Display "Euro value: " Report-in-Euro.
```

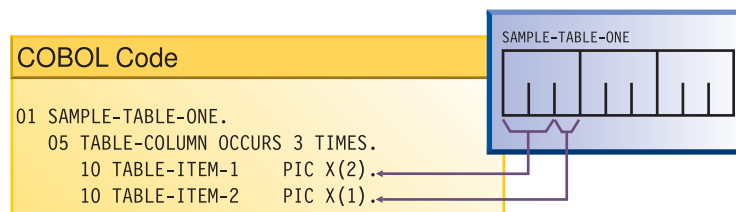
上記の例は、次のような表示出力を作成します。

```
Deposit in euro = EUR 8000.00
Deposit in franc = CHF 12311.44
```

この例で使用されている交換レートは例として使われているだけです。

第 4 章 テーブルの処理

テーブルは、合計額や月平均額のような同じデータ記述を持つデータ項目の集合です。テーブルは、テーブル名と テーブル・エレメント と呼ばれる従属項目から成ります。テーブルは、配列に相当する COBOL 用語です。



上記の例では、SAMPLE-TABLE-ONE が、テーブルを含むグループ項目です。TABLE-COLUMN は、3 回出現する 1 次元テーブルのテーブル・エレメントを指しています。

反復項目を DATA DIVISION に別個の連続する項目として定義するのではなく、DATA DIVISION 記入項目の OCCURS 節を使用してテーブルを定義します。この方法には、次のような利点があります。

- コードは、項目 (テーブル・エレメント) の単位を明確に示します。
- 添え字と指標を使用してテーブル・エレメントを参照することができます。
- データ項目の反復が容易です。

テーブルは、プログラムの処理速度、特にレコードを探索する速度を高めるうえで大切なものです。

関連タスク

- 67 ページの『テーブルのネスト』
- 『テーブルの定義 (OCCURS)』
- 69 ページの『テーブル内の項目の参照』
- 71 ページの『テーブルに値を入れる方法』
- 77 ページの『可変長テーブルの作成 (DEPENDING ON)』
- 80 ページの『テーブルの探索』
- 83 ページの『組み込み関数を使用したテーブル項目の処理』
- 592 ページの『テーブルの効率的処理』

テーブルの定義 (OCCURS)

テーブルをコーディングするには、テーブルにグループ名を与え、 n 回繰り返される従属項目 (テーブル・エレメント) を定義します。

```
01 table-name.  
  05 element-name OCCURS n TIMES.  
    . . . (subordinate items of the table element)
```

上記の例では、table-name は、英数字グループ項目の名前です。テーブル・エレメント定義 (OCCURS 節が組み込まれている) は、テーブルを含むグループ項目に従属しています。OCCURS 節をレベル 01 記述で指定することはできません。

テーブルに入れるのが Unicode (UTF-16) データのみであり、テーブルを含んでいるグループ項目がほとんどの操作で基本カテゴリー国別項目と同様に振る舞うようにさせたい場合には、グループ項目に GROUP-USAGE NATIONAL 節をコーディングします。

```
01 table-nameN Group-Usage National.  
   05 element-nameN OCCURS m TIMES.  
     10 elementN1 Pic nn.  
     10 elementN2 Pic S99 Sign Is Leading, Separate.  
     . . .
```

国別グループに従属する基本項目は、明示的または暗黙的に USAGE NATIONAL として記述する必要があります。また符合付きの従属数値データ項目は、暗黙的または明示的に SIGN IS SEPARATE 節で記述されている必要があります。

2次元から7次元までのテーブルを作成するには、ネストされた OCCURS 節を使用してください。

可変長テーブルを作成するには、OCCURS 節に DEPENDING ON 句をコーディングしてください。

テーブルの1つ以上のキー・フィールドの値に基づいて、テーブル・エレメントが昇順または降順に配列されるよう指定するには、OCCURS 節の ASCENDING または DESCENDING KEY 句 (あるいはその両方) をコーディングしてください。キーの名前は重要度の高い順に指定します。キーは、クラス英字、英数字、DBCS、国別、または数値にすることができます。(USAGE NATIONAL を持っている場合、キーはカテゴリー国別にすることができます。あるいは国別編集、数字編集、国別10進数、または国別浮動小数点の項目にすることもできます。)

テーブルの二分探索 (SEARCH ALL) を行うには、OCCURS 節の ASCENDING または DESCENDING KEY 句をコーディングする必要があります。

83 ページの『例: 二分探索』

関連概念

192 ページの『国別グループ』

関連タスク

67 ページの『テーブルのネスト』

69 ページの『テーブル内の項目の参照』

71 ページの『テーブルに値を入れる方法』

77 ページの『可変長テーブルの作成 (DEPENDING ON)』

194 ページの『国別グループの使用』

82 ページの『二分探索 (SEARCH ALL)』

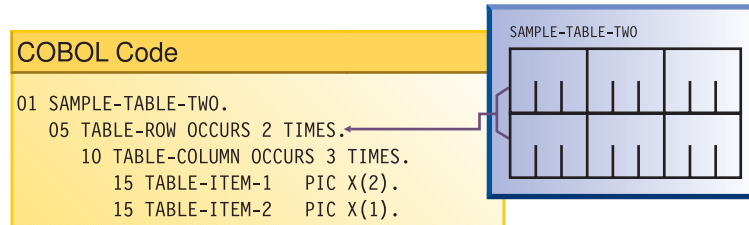
41 ページの『数値データの定義』

関連参照

OCCURS 節 (COBOL for AIX 言語解説書)

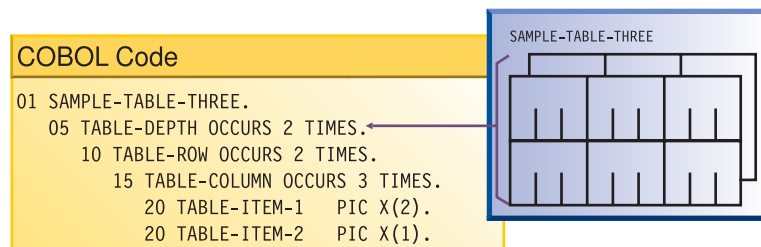
テーブルのネスト

2 次元テーブルを作成するには、ある 1 次元テーブルのそれぞれのオカレンス (出現) の中で別の 1 次元テーブルを定義します。



例えば、上の SAMPLE-TABLE-TWO の TABLE-ROW は、2 回出現する 1 次元テーブルの要素です。TABLE-COLUMN は、2 次元テーブルの要素で、TABLE-ROW のそれぞれのオカレンスで 3 回出現します。

3 次元テーブルを作成するには、ある 1 次元テーブル (それ自体が別の 1 次元テーブルのそれぞれのオカレンスに含まれている) のそれぞれのオカレンスの中で別の 1 次元テーブルを定義します。以下に例を示します。



SAMPLE-TABLE-THREE の TABLE-DEPTH は 1 次元テーブルの要素で、2 回出現します。TABLE-ROW は 2 次元テーブルの要素で、TABLE-DEPTH のそれぞれのオカレンスで 2 回出現します。TABLE-COLUMN は 3 次元テーブルの要素で、TABLE-ROW のそれぞれのオカレンスで 3 回出現します。

2 次元テーブルでは、2 つの添え字が行番号と列番号に対応します。3 次元テーブルでは、3 つの添え字が深さ番号、列番号、および行番号に対応します。

68 ページの『例: 添え字付け』

68 ページの『例: 指標付け』

関連タスク

65 ページの『テーブルの定義 (OCCURS)』

69 ページの『テーブル内の項目の参照』

71 ページの『テーブルに値を入れる方法』

77 ページの『可変長テーブルの作成 (DEPENDING ON)』

80 ページの『テーブルの探索』
83 ページの『組み込み関数を使用したテーブル項目の処理』
592 ページの『テーブルの効率的処理』

関連参照

OCCURS 節 (COBOL for AIX 言語解説書)

例: 添え字付け

次の例は、リテラル添え字を使用している、SAMPLE-TABLE-THREE への有効な参照を示しています。2 番目の例では、スペースは必須です。

```
TABLE-COLUMN (2, 2, 1)  
TABLE-COLUMN (2 2 1)
```

いずれのテーブル参照でも、最初の値 (2) は TABLE-DEPTH 内の 2 番目のオカレンスを参照し、2 つ目の値 (2) は TABLE-ROW 内の 2 番目のオカレンスを参照し、3 つ目の値 (1) は TABLE-COLUMN 内の 1 番目のオカレンスを参照します。

次の SAMPLE-TABLE-TWO への参照では、変数添え字が使用されています。SUB1 と SUB2 が、テーブルの範囲内の正の整数値を含むデータ名であれば、この参照は有効になります。

```
TABLE-COLUMN (SUB1 SUB2)
```

関連タスク

69 ページの『添え字付け』

例: 指標付け

次の例は、指標で参照されるエレメントの変位を計算する方法を示しています。

次の 3 次元テーブル SAMPLE-TABLE-FOUR を考えてみてください。

```
01 SAMPLE-TABLE-FOUR  
   05 TABLE-DEPTH OCCURS 3 TIMES INDEXED BY INX-A.  
      10 TABLE-ROW OCCURS 4 TIMES INDEXED BY INX-B.  
         15 TABLE-COLUMN OCCURS 8 TIMES INDEXED BY INX-C PIC X(8).
```

SAMPLE-TABLE-FOUR に対して次の相対指標付け参照をコーディングするとします。

```
TABLE-COLUMN (INX-A + 1, INX-B + 2, INX-C - 1)
```

この参照によって、TABLE-COLUMN への変位が次のように計算されます。

```
(contents of INX-A) + (256 * 1)  
+ (contents of INX-B) + (64 * 2)  
+ (contents of INX-C) - (8 * 1)
```

この計算は、次のエレメント長に基づいています。

- TABLE-DEPTH のそれぞれのオカレンスは 256 バイトの長さです (4 * 8 * 8)。
- TABLE-ROW のそれぞれのオカレンスは 64 バイトの長さです (8 * 8)。
- TABLE-COLUMN のそれぞれのオカレンスは 8 バイトの長さです。

関連タスク

70 ページの『索引付け』

テーブル内の項目の参照

テーブル・エレメントは集合名を持ちますが、その中の個々の項目は固有のデータ名を持っていません。

項目を参照するには、次の 3 つの方法のいずれかを使用できます。

- テーブル・エレメントのデータ名と一緒に、そのオカレンス番号 (添え字 と呼ばれる) を括弧で囲んで使用する。この手法は、添え字付け と呼ばれます。
- テーブル・エレメントのデータ名と一緒に、項目を位置指定するために (テーブルの先頭からの変位として) テーブルのアドレスに追加される値 (指標 と呼ばれる) を使用する。この手法は、指標付け、または指標名を使用する添え字付け と呼ばれます。
- 添え字と指標の両方を使用する。

関連タスク

『添え字付け』

70 ページの『索引付け』

添え字付け

可能な一番小さい添え字値は 1 であり、これはテーブル・エレメントの最初に現れるものを指します。1 次元テーブルでは、添え字は行番号に対応します。

添え字としてはリテラルまたはデータ名を使用できます。リテラルの添え字を持つデータ項目が固定長である場合は、コンパイラーがそのデータ項目の位置を解決します。

データ名を変数添え字として使用する場合、データ名を基本数値整数として記述する必要があります。最も効率的な形式は、PICTURE サイズが 5 桁よりも少ない COMPUTATIONAL (COMP) です。添え字として使用されるデータ名に添え字を付けることはできません。アプリケーションのために生成されるコードが、実行時に変数添え字の位置を解決します。

リテラルまたは変数の添え字を、指定した整数分だけ増分または減分することができます。以下に例を示します。

```
TABLE-COLUMN (SUB1 - 1, SUB2 + 3)
```

テーブル・エレメント全体ではなく、その一部を変更することができます。これを行うには、変更するサブストリングの文字位置と長さを参照すればよいだけです。以下に例を示します。

```
01 ANY-TABLE.  
   05 TABLE-ELEMENT    PIC X(10)  
      OCCURS 3 TIMES    VALUE "ABCDEFGHJIJ".  
  . . .  
  MOVE "???" TO TABLE-ELEMENT (1) (3 : 2).
```

上の例の MOVE ステートメントは、ストリング「??」を、文字位置 3 から始めて 2 の長さだけ、テーブル・エレメント 1 に移動します。

```
ANY-TABLE
変更前:
ABCDEFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
```

```
ANY-TABLE
変更後:
AB??EFGHIJ
ABCDEFGHIJ
ABCDEFGHIJ
```

68 ページの『例: 添え字付け』

関連タスク

『索引付け』

71 ページの『テーブルに値を入れる方法』

80 ページの『テーブルの探索』

592 ページの『テーブルの効率的処理』

索引付け

指標名を識別する OCCURS 節の INDEXED BY 句を使用して、指標を作成します。

例えば、以下のコードにおける INX-A は指標名です。

```
05 TABLE-ITEM PIC X(8)
   OCCURS 10 INDEXED BY INX-A.
```

コンパイラーは、指標に含まれる値を、オカレンス番号 (添え字) から 1 引いた値にテーブル・エレメントの長さを掛けた値として計算します。したがって、TABLE-ITEM の 5 回目のオカレンスの場合、INX-A に含まれる 2 進値は、 $(5 - 1) * 8$ 、すなわち 32 です。

指標名を使用して別のテーブルを参照できるのは、両方のテーブル記述のテーブル・エレメントの数が同じであり、テーブル・エレメントが同じ長さである場合のみです。

USAGE IS INDEX 節を使用して指標データ項目を作成でき、また任意のテーブルで指標データ項目を使用できます。例えば、以下のコードの INX-B は指標データ項目です。

```
77 INX-B USAGE IS INDEX.
...
   SET INX-A TO 10
   SET INX-B TO INX-A.
   PERFORM VARYING INX-A FROM 1 BY 1 UNTIL INX-A > INX-B
   DISPLAY TABLE-ITEM (INX-A)
...
END-PERFORM.
```

上のテーブル TABLE-ITEM を全探索するのに、指標名 INX-A を使用します。テーブルの最後のエレメントの指標を保持するには、指標データ項目 INX-B を使用します。このタイプのコーディングの利点は、テーブル・エレメントのオフセットの計算が最小限で済み、UNTIL 条件の変換が不要であることです。

SET ステートメントを使用すれば、上のステートメント SET INX-B TO INX-A のように、指標名に保管した値を指標データ項目に割り当てることができます。例えば、レコードを可変長テーブルにロードするとき、読み取られた最後のレコードの指標値を、USAGE IS INDEX として定義されたデータ項目に保管できます。そのあ

と、現行の指標値を最後のレコードの指標値と比較することによって、テーブルの終わりをテストすることができます。この手法は、テーブルを初めから終わりまで検索したり、テーブルを処理したりする場合に有用です。

例えば次のように、基本整数データ項目またはゼロ以外の整数リテラルによって指標名を増分したり、減分したりすることができます。

```
SET INX-A DOWN BY 3
```

整数は出現回数を表します。索引に対して加算または減算される前に、指標値に変換されます。

SET、PERFORM VARYING、または SEARCH ALL ステートメントを使用して、指標名を初期化してください。そのあと、索引名を SEARCH ステートメントまたは関係条件ステートメントでも使用できるようになります。値を変更するには、PERFORM、SEARCH、または SET ステートメントを使用してください。

物理的変位を比較するので、SEARCH および SET ステートメントでのみ、あるいは指標または他の指標データ項目との比較にのみ、指標データ項目を直接使用できません。指標データ項目を添え字または指標として使用することはできません。

68 ページの『例: 指標付け』

関連タスク

69 ページの『添え字付け』

『テーブルに値を入れる方法』

80 ページの『テーブルの探索』

83 ページの『組み込み関数を使用したテーブル項目の処理』

592 ページの『テーブルの効率的処理』

関連参照

266 ページの『ADDR』

INDEXED BY 句 (COBOL for AIX 言語解説書)

INDEX 句 (COBOL for AIX 言語解説書)

SET ステートメント (COBOL for AIX 言語解説書)

テーブルに値を入れる方法

テーブルを定義するときに、テーブルの動的ロード、INITIALIZE ステートメントによるテーブルの初期化、または VALUE 節による値の割り当てによって、値をテーブルに入れることができます。

関連タスク

72 ページの『テーブルの動的なロード』

79 ページの『可変長テーブルのロード』

72 ページの『テーブルの初期化 (INITIALIZE)』

73 ページの『テーブルの定義時の値の割り当て (VALUE)』

80 ページの『可変長テーブルへの値の割り当て』

テーブルの動的なロード

テーブルの初期値がプログラムの実行のたびに異なる場合は、初期値を指定せずにテーブルを定義することができます。代わりに、プログラムでテーブルを参照する前に、変更済みの値をテーブルに動的に読み込むことができます。

テーブルをロードするには、PERFORM ステートメントと添え字付けまたは索引付けのいずれかを使用してください。

データを読み取ってテーブルをロードするときは、データがテーブルに割り振られているスペースを超えないように確認してください。最大項目カウントには、名前付きの値 (リテラルではなく) を使用してください。そうすれば、テーブルをより大きくする場合に、リテラルへのすべての参照を変更する代わりに、1 つの値を変更するだけで済みます。

75 ページの『例: PERFORM と添え字付け』

76 ページの『例: PERFORM および索引付け』

関連参照

VARYING 句を指定した PERFORM (COBOL for AIX 言語解説書)

テーブルの初期化 (INITIALIZE)

1 つ以上の INITIALIZE ステートメントをコーディングすることにより、テーブルをロードできます。

例えば、以下に示す TABLE-ONE という名前のテーブルのそれぞれの基本数値データ項目に値 3 を移動するには、次のステートメントをコーディングできます。

```
INITIALIZE TABLE-ONE REPLACING NUMERIC DATA BY 3.
```

文字「X」を、TABLE-ONE のそれぞれの基本英数字データ項目に移動するには、次のステートメントをコーディングできます。

```
INITIALIZE TABLE-ONE REPLACING ALPHANUMERIC DATA BY "X".
```

INITIALIZE ステートメントを使用してテーブルを初期化するとき、テーブルはグループ項目として (つまり、グループ・セマンティクスで) 処理されます。すなわち、グループ内の基本データ項目が認識されて処理されます。例えば、TABLE-ONE が、以下のように定義された英数字グループであるとしましょう。

```
01 TABLE-ONE.  
  02 Trans-out Occurs 20.  
    05 Trans-code Pic X Value "R".  
    05 Part-number Pic XX Value "13".  
    05 Trans-quan Pic 99 Value 10.  
    05 Price-fields.  
      10 Unit-price Pic 99V Value 50.  
      10 Discount Pic 99V Value 25.  
      10 Sales-Price Pic 999 Value 375.  
      . . .  
      Initialize TABLE-ONE Replacing Numeric Data By 3  
      Alphanumeric Data By "X"
```


以下の表は、上記の INITIALIZE ステートメントの実行前および実行後に 20 個の 12 バイト・エレメント Trans-out(n) のそれぞれが含んでいる内容を示しています。

Trans-out(n) (実行前)	Trans-out(n) (実行後)
R13105025375	XXb030303003 ¹
1. 記号 <i>b</i> は、ブランク・スペースを表します。	

同様に INITIALIZE ステートメントを使用して、国別グループとして定義されたテーブルをロードできます。例えば、上記の TABLE-ONE で GROUP-USAGE NATIONAL 節を指定した場合で、Trans-code および Part-number の PICTURE 節に X ではなく N が指定されている場合、以下のステートメントは、上記の INITIALIZE ステートメントと同じ効果を持つこととなります (ただし、TABLE-ONE のデータは代わりに UTF-16 でエンコードされます)。

```
Initialize TABLE-ONE Replacing Numeric Data By 3
                        National Data By N"X"
```

REPLACING NUMERIC 句は、浮動小数点データ項目も初期化します。

INITIALIZE ステートメントの REPLACING 句を同様に使用して、テーブル内の基本データ項目 ALPHABETIC、DBCS、ALPHANUMERIC-EDITED、NATIONAL-EDITED、および NUMERIC-EDITED のすべてを初期化できます。

INITIALIZE ステートメントでは、値を可変長テーブル (つまり、OCCURS DEPENDING ON 節を使用して定義されたテーブル) に割り当てることはできません。

28 ページの『例: データ項目の初期化』

関連タスク

- 31 ページの『構造の初期化 (INITIALIZE)』
- 『テーブルの定義時の値の割り当て (VALUE)』
- 80 ページの『可変長テーブルへの値の割り当て』
- 96 ページの『テーブルのループ処理』
- 24 ページの『データ項目とグループ項目の使用』
- 194 ページの『国別グループの使用』

関連参照

INITIALIZE ステートメント (COBOL for AIX 言語解説書)

テーブルの定義時の値の割り当て (VALUE)

テーブルに安定値 (日や月など) が入る場合、テーブルの定義時に特定の値を設定できます。

テーブル内の静的値は、次のいずれかの方法で設定します。

- 各テーブル項目を個別に初期化する。
- テーブル全体をグループ・レベルで初期化する。
- 特定のテーブル・エレメントのすべてのオカレンスを同じ値に初期化する。

関連タスク

- 『それぞれのテーブル項目の個別の初期化』
- 『グループ・レベルでのテーブルの初期化』
- 75 ページの『ある特定テーブル・エレメントのすべての出現の初期化』
- 31 ページの『構造の初期化 (INITIALIZE)』

それぞれのテーブル項目の個別の初期化

テーブルが小さい場合は、VALUE 節を使用してそれぞれの項目の値を個別に設定できます。

以下のコード例に示されている手法を使用します。

1. テーブルに入れられる項目を含むレコード (以下の Error-Flag-Table など) を宣言します。
2. 各項目の初期値を VALUE 節で設定します。
3. そのレコードをテーブルに入れるための REDEFINES 記入項目をコーディングします。

```
*****  
***          E R R O R   F L A G   T A B L E          ***  
*****  
01 Error-Flag-Table                               Value Spaces.  
   88 No-Errors                                   Value Spaces.  
     05 Type-Error                               Pic X.  
     05 Shift-Error                             Pic X.  
     05 Home-Code-Error                         Pic X.  
     05 Work-Code-Error                        Pic X.  
     05 Name-Error                             Pic X.  
     05 Initials-Error                         Pic X.  
     05 Duplicate-Error                        Pic X.  
     05 Not-Found-Error                        Pic X.  
01 Filler Redefines Error-Flag-Table.  
   05 Error-Flag Occurs 8 Times  
     Indexed By Flag-Index                       Pic X.
```

上の例で、01 レベルの VALUE 節は、それぞれのテーブル項目を同じ値に初期化します。代わりに、それぞれのテーブル項目に独自の VALUE 節を記述して、その項目を別個の値に初期化することもできます。

もっと大きなテーブルを初期化する場合は、MOVE、PERFORM、または INITIALIZE ステートメントを使用します。

関連タスク

- 31 ページの『構造の初期化 (INITIALIZE)』
- 80 ページの『可変長テーブルへの値の割り当て』

関連参照

- REDEFINES 節 (*COBOL for AIX 言語解説書*)
- OCCURS 節 (*COBOL for AIX 言語解説書*)

グループ・レベルでのテーブルの初期化

英数字または国別グループ・データ項目をコーディングし、VALUE 節でそれにテーブル全体の内容を割り当てます。次に、従属データ項目で、OCCURS 節を使用して個々のテーブル項目を定義します。

以下の例の英数字グループ・データ項目 TABLE-ONE は、TABLE-TWO の 4 個のエレメントそれぞれを初期化する VALUE 節を使用しています。

```
01 TABLE-ONE VALUE "1234".
   05 TABLE-TWO OCCURS 4 TIMES PIC X.
```

以下の例の国別グループ・データ項目 Table-OneN は、従属データ項目 Table-TwoN の 3 個のエレメントそれぞれを初期化する VALUE 節を使用しています (エレメントのそれぞれは暗黙的に USAGE NATIONAL です)。英数字リテラルを使用する VALUE 節 (以下に示されている) または国別リテラルを使用する VALUE 節で国別グループ・データ項目を初期化できることに注意してください。

```
01 Table-OneN Group-Usage National Value "AB12CD34EF56".
   05 Table-TwoN Occurs 3 Times Indexed By MyI.
      10 ElementOneN Pic nn.
      10 ElementTwoN Pic 99.
```

Table-OneN の初期化後に、ElementOneN(1) には、NX"00410042" ('AB' の UTF-16 表現) が入り、国別 10 進数 項目 ElementTwoN(1) には、NX"00310032" ('12' の UTF-16 表現) が入ります。以下同様です。

関連参照

OCCURS 節 (*COBOL for AIX 言語解説書*)

GROUP-USAGE 節 (*COBOL for AIX 言語解説書*)

ある特定テーブル・エレメントのすべての出現の初期化

テーブル・エレメントのデータ記述にある VALUE 節を使用して、そのエレメントのすべてのインスタンスを指定した値に初期化することができます。

```
01 T2.
   05 T-OBJ PIC 9 VALUE 3.
   05 T OCCURS 5 TIMES
      DEPENDING ON T-OBJ.
      10 X PIC XX VALUE "AA".
      10 Y PIC 99 VALUE 19.
      10 Z PIC XX VALUE "BB".
```

例えば、上のコードによって、すべての X エレメント (1 から 5) が AA に初期化され、すべての Y エレメント (1 から 5) が 19 に初期化され、すべての Z エレメント (1 から 5) が BB に初期化されます。その後、T-OBJ が 3 に設定されます。

関連タスク

80 ページの『可変長テーブルへの値の割り当て』

関連参照

OCCURS 節 (*COBOL for AIX 言語解説書*)

例: PERFORM と添え字付け

この例は、設定されたエラー・コードが検出されるまで、添え字付けを使用してエラー・フラグ (error-flag) テーブルを全探索します。エラー・コードが見つかったら、対応するエラー・メッセージが報告書印刷フィールドに移動されます。

```
*****
***          ERROR FLAG TABLE          ***
*****
```

```

01 Error-Flag-Table                Value Spaces.
   88 No-Errors                    Value Spaces.
     05 Type-Error                 Pic X.
     05 Shift-Error                Pic X.
     05 Home-Code-Error            Pic X.
     05 Work-Code-Error            Pic X.
     05 Name-Error                 Pic X.
     05 Initials-Error             Pic X.
     05 Duplicate-Error            Pic X.
     05 Not-Found-Error            Pic X.
01 Filler Redefines Error-Flag-Table.
   05 Error-Flag Occurs 8 Times
       Indexed By Flag-Index       Pic X.
77 Error-on                        Pic X Value "E".
*****
***          E R R O R   M E S S A G E   T A B L E          ***
*****
01 Error-Message-Table.
   05 Filler                       Pic X(25) Value
       "Transaction Type Invalid".
   05 Filler                       Pic X(25) Value
       "Shift Code Invalid".
   05 Filler                       Pic X(25) Value
       "Home Location Code Inval.".
   05 Filler                       Pic X(25) Value
       "Work Location Code Inval.".
   05 Filler                       Pic X(25) Value
       "Last Name - Blanks".
   05 Filler                       Pic X(25) Value
       "Initials - Blanks".
   05 Filler                       Pic X(25) Value
       "Duplicate Record Found".
   05 Filler                       Pic X(25) Value
       "Commuter Record Not Found".
01 Filler Redefines Error-Message-Table.
   05 Error-Message Occurs 8 Times
       Indexed By Message-Index    Pic X(25).
. . .
PROCEDURE DIVISION.
. . .
Perform
  Varying Sub From 1 By 1
  Until No-Errors
  If Error-Flag (Sub) = Error-On
  Move Space To Error-Flag (Sub)
  Move Error-Message (Sub) To Print-Message
  Perform 260-Print-Report
  End-If
End-Perform
. . .

```

例: PERFORM および索引付け

この例は、設定されたエラー・コードが検出されるまで、指標付けを使用してエラー・フラグ (error-flag) テーブルを全探索します。エラー・コードが見つかると、対応するエラー・メッセージが報告書印刷フィールドに移動されます。

```

*****
***          E R R O R   F L A G   T A B L E          ***
*****
01 Error-Flag-Table                Value Spaces.
   88 No-Errors                    Value Spaces.
     05 Type-Error                 Pic X.
     05 Shift-Error                Pic X.
     05 Home-Code-Error            Pic X.

```

```

05 Work-Code-Error          Pic X.
05 Name-Error              Pic X.
05 Initials-Error         Pic X.
05 Duplicate-Error        Pic X.
05 Not-Found-Error        Pic X.
01 Filler Redefines Error-Flag-Table.
05 Error-Flag Occurs 8 Times
    Indexed By Flag-Index      Pic X.
77 Error-on                 Pic X Value "E".
*****
***      E R R O R   M E S S A G E   T A B L E      ***
*****
01 Error-Message-Table.
05 Filler                   Pic X(25) Value
    "Transaction Type Invalid".
05 Filler                   Pic X(25) Value
    "Shift Code Invalid".
05 Filler                   Pic X(25) Value
    "Home Location Code Inval.".
05 Filler                   Pic X(25) Value
    "Work Location Code Inval.".
05 Filler                   Pic X(25) Value
    "Last Name - Blanks".
05 Filler                   Pic X(25) Value
    "Initials - Blanks".
05 Filler                   Pic X(25) Value
    "Duplicate Record Found".
05 Filler                   Pic X(25) Value
    "Commuter Record Not Found".
01 Filler Redefines Error-Message-Table.
05 Error-Message Occurs 8 Times
    Indexed By Message-Index  Pic X(25).

. . .
PROCEDURE DIVISION.
. . .
Set Flag-Index To 1
Perform Until No-Errors
  Search Error-Flag
    When Error-Flag (Flag-Index) = Error-On
      Move Space To Error-Flag (Flag-Index)
      Set Message-Index To Flag-Index
      Move Error-Message (Message-Index) To
        Print-Message
      Perform 260-Print-Report
    End-Search
  End-Perform
. . .

```

可変長テーブルの作成 (DEPENDING ON)

テーブル・エレメントが出現する回数が実行前にわからない場合には、可変長テーブルを定義してください。そのためには、OCCURS DEPENDING ON (ODO) 節を使用します。

```
X OCCURS 1 TO 10 TIMES DEPENDING ON Y
```

上の例で、X は ODO サブジェクトと呼び、Y は ODO オブジェクトと呼びます。

可変長レコードを正しく操作するためには、次の 2 つの要因が影響します。

- レコード長を正しく計算すること

グループ項目の可変部分の長さは、DEPENDING ON 句のオブジェクトと OCCURS 節のサブジェクトの長さとの積です。

- OCCURS DEPENDING ON 節のオブジェクトにおけるデータの PICTURE 節との適合性

ODO オブジェクトの内容がその PICTURE 節と一致していない場合には、プログラムが異常終了することがあります。ODO オブジェクトに、テーブル・エレメントの現在のオカレンス回数を正しく指定するようにしてください。

次の例は、OCCURS DEPENDING ON 節のサブジェクトとオブジェクトの両方を含むグループ項目 (REC-1) を示しています。グループ項目の長さがどのようにして決定されるかは、それがデータを送り出しているのか、データを受け取っているのかによって異なります。

```
WORKING-STORAGE SECTION.  
01 MAIN-AREA.  
  03 REC-1.  
    05 FIELD-1                               PIC 9.  
    05 FIELD-2 OCCURS 1 TO 5 TIMES  
      DEPENDING ON FIELD-1                   PIC X(05).  
01 REC-2.  
  03 REC-2-DATA                               PIC X(50).
```

REC-1 (この場合は送り出し項目) を REC-2 に移動したい場合、REC-1 の長さは、FIELD-1 の現行値を使用して、移動の直前に決定されます。FIELD-1 の内容がその PICTURE 節と一致している場合 (すなわち、FIELD-1 がゾーン 10 進数項目を含んでいる場合)、REC-1 の実際の長さに基づいて移動は続行可能です。それ以外の場合、結果は予測できません。移動を開始する前に、ODO オブジェクトに正しい値が含まれていることを確認してください。

REC-1 (この場合は受け取り項目) に移動を行う場合、REC-1 の長さは、最大のオカレンス回数を使用して決定されます。この例では、FIELD-2 の 5 回のオカレンスと FIELD-1 の合計で 26 バイトの長さになります。この場合、REC-1 を受け取り項目として参照する前に、ODO オブジェクト (FIELD-1) を設定する必要はありません。ただし、移動によって受信フィールドの ODO オブジェクトを有効に設定するために、送信フィールドの ODO オブジェクト (非表示) を 1 から 5 の間の有効な数値に設定しなければなりません。

しかし、REC-1 の後に可変位置グループ (複合 ODO) が続いているような REC-1 (この場合も受け取り項目) に移動を行う場合は、REC-1 の実際の長さは、ODO オブジェクト (FIELD-1) の現行値を使用して、移動の直前に計算されます。次の例では、REC-1 と REC-2 は同じレコードにありますが、REC-2 は REC-1 に従属していないため、可変位置項目です。

```
01 MAIN-AREA  
  03 REC-1.  
    05 FIELD-1                               PIC 9.  
    05 FIELD-3                               PIC 9.  
    05 FIELD-2 OCCURS 1 TO 5 TIMES  
      DEPENDING ON FIELD-1                   PIC X(05).  
  03 REC-2.  
    05 FIELD-4 OCCURS 1 TO 5 TIMES  
      DEPENDING ON FIELD-3                   PIC X(05).
```

コンパイラーは、実際の長さが使用されたことを知らせるメッセージを出します。この場合には、グループ項目を受信フィールドとして使用する前に、ODO オブジェクトの値を設定することが必要となります。

次の例は、ODO オブジェクト (下記の LOCATION-TABLE-LENGTH) がグループの外側にあるときの、可変長テーブルの定義方法を示します。

```
DATA DIVISION.
FILE SECTION.
FD LOCATION-FILE.
01 LOCATION-RECORD.
   05 LOC-CODE          PIC XX.
   05 LOC-DESCRIPTION  PIC X(20).
   05 FILLER           PIC X(58).
WORKING-STORAGE SECTION.
01 FLAGS.
   05 LOCATION-EOF-FLAG PIC X(5) VALUE SPACE.
   88 LOCATION-EOF     VALUE "FALSE".
01 MISC-VALUES.
   05 LOCATION-TABLE-LENGTH PIC 9(3) VALUE ZERO.
   05 LOCATION-TABLE-MAX   PIC 9(3) VALUE 100.
*****
***          L O C A T I O N   T A B L E          ***
***          FILE CONTAINS LOCATION CODES.      ***
*****
01 LOCATION-TABLE.
   05 LOCATION-CODE OCCURS 1 TO 100 TIMES
      DEPENDING ON LOCATION-TABLE-LENGTH PIC X(80).
```

関連概念

643 ページの『付録 E. 複合 OCCURS DEPENDING ON』

関連タスク

80 ページの『可変長テーブルへの値の割り当て』

『可変長テーブルのロード』

646 ページの『エレメントを可変テーブルに追加する際のオーバーレイを防止する』

118 ページの『データ項目の長さの検出』

関連参照

OCCURS DEPENDING ON 節 (COBOL for AIX 言語解説書)

可変長テーブルのロード

do-until 構造 (TEST AFTER ループ) を使用して、可変長テーブルのロードを制御することができます。例えば、次のコードが実行されると、LOCATION-TABLE-LENGTH には、テーブルの最後の項目の添え字が入れられます。

```
DATA DIVISION.
FILE SECTION.
FD LOCATION-FILE.
01 LOCATION-RECORD.
   05 LOC-CODE          PIC XX.
   05 LOC-DESCRIPTION  PIC X(20).
   05 FILLER           PIC X(58).
. . .
WORKING-STORAGE SECTION.
01 FLAGS.
   05 LOCATION-EOF-FLAG PIC X(5) VALUE SPACE.
   88 LOCATION-EOF     VALUE "YES".
01 MISC-VALUES.
   05 LOCATION-TABLE-LENGTH PIC 9(3) VALUE ZERO.
   05 LOCATION-TABLE-MAX   PIC 9(3) VALUE 100.
*****
***          L O C A T I O N   T A B L E          ***
***          FILE CONTAINS LOCATION CODES.      ***
*****
```

```

01 LOCATION-TABLE.
   05 LOCATION-CODE OCCURS 1 TO 100 TIMES
      DEPENDING ON LOCATION-TABLE-LENGTH PIC X(80).
. . .
PROCEDURE DIVISION.
. . .
Perform Test After
   Varying Location-Table-Length From 1 By 1
   Until Location-EOF
   Or Location-Table-Length = Location-Table-Max
Move Location-Record To
   Location-Code (Location-Table-Length)
Read Location-File
   At End Set Location-EOF To True
End-Read
End-Perform

```

可変長テーブルへの値の割り当て

DEPENDING ON 句を持つ OCCURS 節を含んでいる、従属データ項目のある英数字または国別グループ項目に、VALUE 節をコーディングできます。DEPENDING ON 句を含むそれぞれの従属構造は、最大オカレンス回数を使用して初期化されます。

DEPENDING ON 句を使用してテーブル全体を定義する場合、ODO (OCCURS DEPENDING ON) オブジェクトの最大定義値を使用して、全エレメントが初期化されます。

ODO オブジェクトが VALUE 節で初期化される場合、これは必然的に、ODO サブジェクトが初期化された後に初期化されます。

```

01 TABLE-THREE          VALUE "3ABCDE".
   05 X                   PIC 9.
   05 Y OCCURS 5 TIMES
      DEPENDING ON X PIC X.

```

例えば、上記のコードで、ODO サブジェクト Y(1) は「A」に、Y(2) は「B」に、・・・Y(5) は「E」に初期化され、最後に ODO オブジェクト X が 3 に初期化されます。TABLE-THREE への後続の参照 (DISPLAY ステートメントでの参照などは、X とテーブルの最初の 3 つのエレメント

(Y(1) から Y(3)) を参照します。

関連タスク

73 ページの『テーブルの定義時の値の割り当て (VALUE)』

関連参照

OCCURS DEPENDING ON 節 (COBOL for AIX 言語解説書)

テーブルの探索

COBOL は、2 つのテーブルの検索手法 (逐次 および バイナリー) を提供します。

逐次探索を行うには、SEARCH と索引付けを使用します。可変長テーブルの場合は、PERFORM と添え字付けまたは指標付けを使用することができます。

二分探索を行うには、SEARCH ALL と索引付けを使用します。

二分探索の方が、逐次探索よりも効率が相当よくなる可能性があります。逐次探索の場合、比較の数は、 n の回数、すなわちテーブルの項目の数です。二分探索の場合、比較の数は、 n の対数 (基底 2) の回数にすぎません。ただし、二分探索を行うには、テーブル項目をあらかじめソートしておく必要があります。

関連タスク

『逐次探索 (SEARCH)』

82 ページの『二分探索 (SEARCH ALL)』

逐次探索 (SEARCH)

SEARCH ステートメントを使用して、現行指標設定値を開始点として逐次 (順次) 探索を行います。指標設定値を変更するには、SET ステートメントを使用してください。

WHEN 句の条件は、それらが指定された順番に評価されます。

- どの条件も満たされない場合には、指標が次のテーブル・エレメントに対応するように増加され、WHEN 条件が再度評価されます。
- WHEN 条件の 1 つが満たされると、探索は終了します。指標は条件を満たしたテーブル・エレメントを指したままになります。
- テーブル全体が探索され、どの条件も満たされなかった場合には、AT END 命令ステートメントが実行されます (存在する場合)。AT END をコーディングしなかった場合、制御はプログラム内の次のステートメントに渡ります。

それぞれの SEARCH ステートメントでは、テーブルの 1 つのレベル (1 つのテーブル・エレメント) だけを参照することができます。テーブルの複数のレベルを探索するには、ネストされた SEARCH ステートメントを使用してください。ネストされたそれぞれの SEARCH ステートメントは、END-SEARCH で区切らなければなりません。

パフォーマンス: 検出された条件がテーブル内の中間点より後にくる場合には、SET ステートメントを使用してその点より後から探索を開始するように索引を設定することによって、探索を速めることができます。また、最も頻繁に使用されるデータがテーブルの先頭になるようにテーブルを配置すると、逐次探索をより効率的に行うことができます。テーブルが大きくて、事前にソートされている場合には、二分探索の方が効率的です。

『例: 逐次探索』

関連参照

SEARCH ステートメント (COBOL for AIX 言語解説書)

例: 逐次探索

次の例は、3 次元テーブルの最も内部にあるテーブルから特定のストリングを見つける方法を示しています。

テーブルの各次元には独自の指標が割り当てられています (それぞれ 1、4、および 1 に設定されています)。最も内部のテーブル (TABLE-ENTRY3) には昇順キーがあります。

```

01 TABLE-ONE.
   05 TABLE-ENTRY1 OCCURS 10 TIMES
      INDEXED BY TE1-INDEX.
   10 TABLE-ENTRY2 OCCURS 10 TIMES
      INDEXED BY TE2-INDEX.
   15 TABLE-ENTRY3 OCCURS 5 TIMES
      ASCENDING KEY IS KEY1
      INDEXED BY TE3-INDEX.
      20 KEY1                PIC X(5).
      20 KEY2                PIC X(10).
. . .
PROCEDURE DIVISION.
. . .
  SET TE1-INDEX TO 1
  SET TE2-INDEX TO 4
  SET TE3-INDEX TO 1
  MOVE "A1234" TO KEY1 (TE1-INDEX, TE2-INDEX, TE3-INDEX + 2)
  MOVE "AAAAAAAA00" TO KEY2 (TE1-INDEX, TE2-INDEX, TE3-INDEX + 2)
. . .
  SEARCH TABLE-ENTRY3
  AT END
    MOVE 4 TO RETURN-CODE
  WHEN TABLE-ENTRY3(TE1-INDEX, TE2-INDEX, TE3-INDEX)
    = "A1234AAAAAAAA00"
    MOVE 0 TO RETURN-CODE
  END-SEARCH

```

実行後の値:

```

TE1-INDEX = 1
TE2-INDEX = 4
TE3-INDEX points to the TABLE-ENTRY3 item
             that equals "A1234AAAAAAAA00"
RETURN-CODE = 0

```

二分探索 (SEARCH ALL)

SEARCH ALL を使用して二分探索を行う場合、開始する前に指標を設定する必要はありません。指標は常に、OCCURS 節内の最初の指標名と関連付けられるものです。この指標は、探索の効率を最大にするために、実行中に変わります。

SEARCH ALL ステートメントを使用してテーブルを探索するには、テーブルで OCCURS 節の ASCENDING または DESCENDING KEY 句 (あるいはその両方) を指定する必要があります。また ASCENDING および DESCENDING KEY 句で指定されたキーに基づいて既に順序付けられている必要があります。

SEARCH ALL ステートメントの WHEN 句では、テーブルの ASCENDING または DESCENDING KEY 句で指定されているキーをテストできますが、前に現れているすべてのキー (ある場合) をテストする必要があります。テストは等価条件でなければならず、WHEN 句ではキー (テーブルに関連した最初の指標名で添え字が付けられている) かまたはキーに関連した条件名を指定する必要があります。WHEN 条件は、論理連結語として AND のみを使用した複数の単純条件から形成した複合条件であっても構いません。

それぞれのキーとその比較の対象はデータ項目の比較規則に従って、互換性がなければなりません。ただし、キーを国別リテラルまたは ID と比較する場合、キーは国別データ項目にする必要があることに注意してください。

83 ページの『例: 二分探索』

関連タスク

65 ページの『テーブルの定義 (OCCURS)』

関連参照

SEARCH ステートメント (COBOL for AIX 言語解説書)

一般比較条件 (COBOL for AIX 言語解説書)

例: 二分探索

次の例は、テーブルの二分探索をコーディングする方法を示しています。

テーブルにそれぞれが 40 バイトの 90 個の要素が含まれており、3 つのキーがあるとします。1 次キーと 2 次キー (KEY-1 と KEY-2) は昇順ですが、最も重要でないキー (KEY-3) は降順です。

```
01 TABLE-A.  
   05 TABLE-ENTRY OCCURS 90 TIMES  
       ASCENDING KEY-1, KEY-2  
       DESCENDING KEY-3  
       INDEXED BY INDX-1.  
   10 PART-1      PIC 99.  
   10 KEY-1       PIC 9(5).  
   10 PART-2      PIC 9(6).  
   10 KEY-2       PIC 9(4).  
   10 PART-3      PIC 9(18).  
   10 KEY-3       PIC 9(5).
```

このテーブルは、次のステートメントを使用して探索することができます。

```
SEARCH ALL TABLE-ENTRY  
  AT END  
  PERFORM NOENTRY  
  WHEN KEY-1 (INDX-1) = VALUE-1 AND  
        KEY-2 (INDX-1) = VALUE-2 AND  
        KEY-3 (INDX-1) = VALUE-3  
  MOVE PART-1 (INDX-1) TO OUTPUT-AREA  
END-SEARCH
```

3 つのキーのそれぞれが比較対象の値 (それぞれ VALUE-1、VALUE-2、および VALUE-3) と等しい項目が検出された場合、その項目の PART-1 は OUTPUT-AREA へ移動されます。TABLE-A 内のどの項目でも一致するキーが検出されない場合には、NOENTRY ルーチンが実行されます。

組み込み関数を使用したテーブル項目の処理

組み込み関数を使用して、英字、英数字、国別、または数値のテーブル項目を処理できます。(DBCS データ項目は NATIONAL-OF 組み込み関数でのみ処理できます)。テーブル項目のデータ記述は、関数の引数の要件と互換性があるようにする必要があります。

個々のデータを関数引数として参照するには、添え字または指標を使用してください。例えば、Table-One が 3 x 3 の数値項目の配列であるとする、次のようなステートメントを使用して中間要素の平方根を求めることができます。

```
Compute X = Function Sqrt(Table-One(2,2))
```

テーブル内のデータを繰り返し処理することが必要な場合もあります。複数の引数を受け入れる組み込み関数の場合、添え字 ALL を使用して、テーブル内またはテーブルの単一次元内のすべての項目を参照することができます。反復は自動的に処理されるため、コードがより短く、単純になります。

複数の引数を受け入れる関数の場合は、スカラーと配列引数を混在させることができます。

```
Compute Table-Median = Function Median(Arg1 Table-One(ALL))
```

『例: 組み込み関数を使用したテーブルの処理』

関連タスク

37 ページの『組み込み関数の使用 (組み込み関数)』

111 ページの『データ項目の変換 (組み込み関数)』

114 ページの『データ項目の評価 (組み込み関数)』

関連参照

組み込み関数 (*COBOL for AIX 言語解説書*)

例: 組み込み関数を使用したテーブルの処理

以下の例は、ALL 添え字を使用してテーブル内のエレメントの一部または全部に組み込み関数を適用する方法を示しています。

Table-Two が 2 x 3 x 2 の配列であるとする、次のステートメントは、エレメント Table-Two(1,3,1)、Table-Two(1,3,2)、Table-Two(2,3,1)、および Table-Two(2,3,2) の値を合計します。

```
Compute Table-Sum = FUNCTION SUM (Table-Two(ALL, 3, ALL))
```

次の例は、全従業員のさまざまな給与値を計算します。従業員の給与は Employee-Table にエンコードされています。

```
01 Employee-Table.
   05 Emp-Count      Pic s9(4) usage binary.
   05 Emp-Record     Occurs 1 to 500 times
                     depending on Emp-Count.
       10 Emp-Name   Pic x(20).
       10 Emp-Idme   Pic 9(9).
       10 Emp-Salary Pic 9(7)v99.
. . .
Procedure Division.
   Compute Max-Salary = Function Max(Emp-Salary(ALL))
   Compute I          = Function Ord-Max(Emp-Salary(ALL))
   Compute Avg-Salary = Function Mean(Emp-Salary(ALL))
   Compute Salary-Range = Function Range(Emp-Salary(ALL))
   Compute Total-Payroll = Function Sum(Emp-Salary(ALL))
```

第 5 章 プログラム・アクションの選択と反復

COBOL 制御言語を使用すると、論理テストの結果に基づいてプログラム・アクションを選択すること、プログラムおよびデータの選択された部分を繰り返すこと、および 1 つのグループとして実行すべきステートメントを識別することができます。

これらの制御には、IF、EVALUATE、および PERFORM ステートメントと、スイッチおよびフラグの使用が含まれます。

関連タスク

『プログラム・アクションの選択』

94 ページの『プログラム・アクションの繰り返し』

プログラム・アクションの選択

1 つ以上のデータ項目のテストされた値に基づいて、さまざまなプログラム・アクションに備えることができます。

COBOL の IF および EVALUATE ステートメントは、条件式によって 1 つ以上のデータ項目をテストします。

関連タスク

『アクションの選択項目のコーディング』

90 ページの『条件式のコーディング』

関連参照

IF ステートメント (*COBOL for AIX 言語解説書*)

EVALUATE ステートメント (*COBOL for AIX 言語解説書*)

アクションの選択項目のコーディング

IF . . . ELSE は、2 つの処理アクションからいずれかを選択する処理をコーディングする場合に使用します。(THEN という語はオプションです。) 3 つ以上の可能なアクションからいずれかを選択するには、EVALUATE ステートメントを使用します。

```
IF condition-p
  statement-1
ELSE
  statement-2
END-IF
```

2 つの処理選択項目の一方がアクションを取らない場合は、IF ステートメントに ELSE を指定してもしなくてもかまいません。ELSE 節はオプションであるため、IF ステートメントは次のようにコーディングすることができます。

```
IF condition-q
  statement-1
END-IF
```

このコーディングは、簡単な場合に適しています。ロジックが複雑になった場合は、おそらく、ELSE 節を使用する必要があります。例えば、処理選択項目の 1 つだけに対応するアクションがある、ネストされた IF ステートメントがあるとします。その場合は、次のように、ELSE 節と CONTINUE ステートメントを使用して IF ステートメントの NULL ブランチをコーディングすることができます。

```
IF condition-q
  statement-1
ELSE
  CONTINUE
END-IF
```

EVALUATE ステートメントは IF ステートメントの拡張形式であり、これを使用すると、IF ステートメントのネスト（論理エラーやデバッグ問題の一般的な原因となる）を避けることができます。

関連タスク

- 『ネストされた IF ステートメントの使用』
- 87 ページの『EVALUATE ステートメントの使用』
- 90 ページの『条件式のコーディング』

ネストされた IF ステートメントの使用

IF ステートメントが、その可能な分岐の 1 つとして別の IF ステートメントを含んでいるとき、これらの IF ステートメントはネストされている といえます。論理上は、ネストされた IF ステートメントの深さに制限はありません。

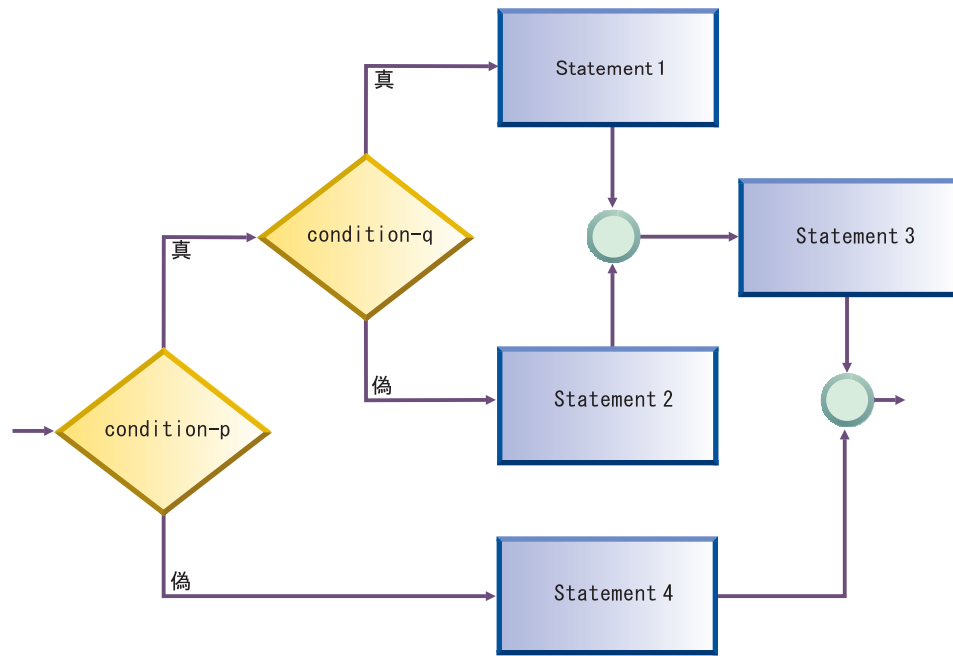
ただし、ネストされた IF ステートメントを多用しないでください。明示範囲終了符号および字下げが役に立つとはいえ、ロジックをたどるのが困難になる可能性があります。プログラムが 2 つを超える値について変数をテストしなければならない場合には、おそらく EVALUATE を使用する方が適切です。

以下に、ネストされた IF ステートメントの疑似コードを示します。

```
IF condition-p
  IF condition-q
    statement-1
  ELSE
    statement-2
  END-IF
  statement-3
ELSE
  statement-4
END-IF
```

上記の疑似コードでは、IF ステートメントと順次構造が外側の IF の 1 つの分岐にネストされています。このような構造では、ネストされた IF を閉じる END-IF が非常に重要になります。ピリオドは外側の IF 構造も終了させるため、ピリオドではなく END-IF を使用してください。

次の図は、上記の疑似コードの論理構造を示しています。



関連タスク

85 ページの『アクションの選択項目のコーディング』

関連参照

明示範囲終了符号 (COBOL for AIX 言語解説書)

EVALUATE ステートメントの使用

ネストされた一連の IF ステートメントではなく、EVALUATE ステートメントを使用して、いくつかの条件をテストし、かつそれぞれについて異なるアクションを指定できます。したがって、EVALUATE ステートメントを使用すると、ケース構造 または デシジョン・テーブルをインプリメントすることができます。

また以下の例に示されているように、EVALUATE ステートメントを使用して、複数の条件で同じ処理が行われるようにすることもできます。

88 ページの『例: THRU 句を使用した EVALUATE』

89 ページの『例: 複数の WHEN 句を使用する EVALUATE』

EVALUATE ステートメントでは、WHEN 句の前のオペランドは選択サブジェクトと呼ばれ、WHEN 句の中のオペランドは選択サブジェクトと呼ばれます。選択サブジェクトは、ID、リテラル、条件式、あるいはワード TRUE または FALSE にすることができます。選択オブジェクトは、ID、リテラル、条件式、算術式、あるいはワード TRUE、FALSE、または ANY にすることができます。

複数の選択サブジェクトを ALSO 句で分離することができます。複数の選択オブジェクトも ALSO 句で分離することができます。それぞれの選択オブジェクト・セット内の選択オブジェクトの数は、次の例に示されているように、選択サブジェクトの数と等しくする必要があります。

89 ページの『例: 複数の条件をテストする EVALUATE』

選択オブジェクト内に現れる ID、リテラル、または算術式は、選択サブジェクト・セット内の対応するオペランドと比較できるよう、有効なオペランドにする必要があります。選択オブジェクト内に現れる条件あるいはワード TRUE または FALSE は、選択サブジェクト・セット内の条件式あるいはワード TRUE または FALSE に対応していなければなりません。(選択オブジェクトとしてワード ANY を使用すると、任意のタイプの選択サブジェクトに対応付けることができます。)

EVALUATE ステートメントの実行は、次のいずれかの条件が発生すると終了します。

- 選択された WHEN 句に関連付けられているステートメントが実行された。
- WHEN OTHER 句に関連付けられているステートメントが実行された。
- いずれの WHEN 条件も満たされない。

WHEN 句は、ソース・プログラムに現れている順序どおりにテストされます。ですから、最高のパフォーマンスが得られるようにこれらの句を順序付ける必要があります。最初に、適合する可能性が最も高い選択オブジェクトを含んでいる WHEN 句をコーディングし、その後、次に可能性の高いものという順にコーディングしてください。例外は WHEN OTHER 句です。これは最後に置かなければなりません。

関連タスク

85 ページの『アクションの選択項目のコーディング』

関連参照

EVALUATE ステートメント (COBOL for AIX 言語解説書)

一般比較条件 (COBOL for AIX 言語解説書)

例: THRU 句を使用した EVALUATE:

この例は、THRU 句をコーディングすることにより、いくつかの条件をある範囲の値でコーディングして同じ処理が行われるようにする方法を示しています。THRU 句内のオペランドは、同じクラスにする必要があります。

この例では、CARPOOL-SIZE は選択サブジェクトであり、1、2、および 3 THRU 6 は選択オブジェクトです。

```
EVALUATE CARPOOL-SIZE
  WHEN 1
    MOVE "SINGLE" TO PRINT-CARPOOL-STATUS
  WHEN 2
    MOVE "COUPLE" TO PRINT-CARPOOL-STATUS
  WHEN 3 THRU 6
    MOVE "SMALL GROUP" TO PRINT-CARPOOL STATUS
  WHEN OTHER
    MOVE "BIG GROUP" TO PRINT-CARPOOL STATUS
END-EVALUATE
```

次のネストされた IF ステートメントも同じロジックを表します。

```
IF CARPOOL-SIZE = 1 THEN
  MOVE "SINGLE" TO PRINT-CARPOOL-STATUS
ELSE
  IF CARPOOL-SIZE = 2 THEN
    MOVE "COUPLE" TO PRINT-CARPOOL-STATUS
  ELSE
    IF CARPOOL-SIZE >= 3 and CARPOOL-SIZE <= 6 THEN
      MOVE "SMALL GROUP" TO PRINT-CARPOOL-STATUS
    ELSE
```



```

        MOVE "BIG GROUP" TO PRINT-CARPOOL-STATUS
      END-IF
    END-IF
  END-IF

```

例: 複数の WHEN 句を使用する EVALUATE:

次の例は、いくつかの条件で同じ処置が行われるようにしなければならない場合は複数の WHEN 句をコーディングできることを示しています。この方法は、THRU 句のみを使用する場合と比べてさらに柔軟性があります。条件が、ある範囲の値に評価される必要もなく、また同じクラスを持つ必要もないからです。

```

EVALUATE MARITAL-CODE
  WHEN "M"
    ADD 2 TO PEOPLE-COUNT
  WHEN "S"
  WHEN "D"
  WHEN "W"
    ADD 1 TO PEOPLE-COUNT
END-EVALUATE

```

次のネストされた IF ステートメントも同じロジックを表します。

```

IF MARITAL-CODE = "M" THEN
  ADD 2 TO PEOPLE-COUNT
ELSE
  IF MARITAL-CODE = "S" OR
    MARITAL-CODE = "D" OR
    MARITAL-CODE = "W" THEN
    ADD 1 TO PEOPLE-COUNT
  END-IF
END-IF

```

例: 複数の条件をテストする EVALUATE:

この例は、ALSO 句を使用して、2 つの選択サブジェクトを分離し (True ALSO True)、それぞれの選択オブジェクト・セット内の対応する 2 つの選択オブジェクトを分離する (例えば、When A + B < 10 Also C = 10) 方法を示しています。

WHEN 句の中の選択オブジェクトはどちらも、関連した処置が実行される前に、TRUE、TRUE 条件を満たす必要があります。両方のオブジェクトが TRUE に評価されなければ、次の WHEN 句が処理されます。

```

Identification Division.
  Program-ID. MiniEval.
Environment Division.
  Configuration Section.
Data Division.
  Working-Storage Section.
  01  Age           Pic 999.
  01  Sex           Pic X.
  01  Description   Pic X(15).
  01  A             Pic 999.
  01  B             Pic 9999.
  01  C             Pic 9999.
  01  D             Pic 9999.
  01  E             Pic 99999.
  01  F             Pic 999999.
Procedure Division.
  PN01.
    Evaluate True Also True
      When Age < 13 Also Sex = "M"
        Move "Young Boy" To Description

```

```

When Age < 13 Also Sex = "F"
  Move "Young Girl" To Description
When Age > 12 And Age < 20 Also Sex = "M"
  Move "Teenage Boy" To Description
When Age > 12 And Age < 20 Also Sex = "F"
  Move "Teenage Girl" To Description
When Age > 19 Also Sex = "M"
  Move "Adult Man" To Description
When Age > 19 Also Sex = "F"
  Move "Adult Woman" To Description
When Other
  Move "Invalid Data" To Description
End-Evaluate
Evaluate True Also True
When A + B < 10 Also C = 10
  Move "Case 1" To Description
When A + B > 50 Also C = ( D + E ) / F
  Move "Case 2" To Description
When Other
  Move "Case Other" To Description
End-Evaluate
Stop Run.

```

条件式のコーディング

IF および EVALUATE ステートメントを使用して、条件式の真理値に従って実行されるプログラム・アクションをコーディングすることができます。

指定できる条件の一部を以下に示します。

- 次のような比較条件
 - 数値比較
 - 英数字比較
 - DBCS 比較
 - 国別比較
- クラス条件 (データ項目が次の条件に当てはまるかどうかのテストなど)
 - IS NUMERIC
 - IS ALPHABETIC
 - IS DBCS
 - IS KANJI
 - IS NOT KANJI
- 条件名条件 (定義した条件変数の値のテスト)
- 符号条件 (数値オペランドが IS POSITIVE、NEGATIVE、または ZERO の条件に当てはまるかどうかのテスト)
- 切り替え状況条件 (SPECIAL-NAMES 段落で名前を付けた UPSI スイッチの状況のテスト)
- 次のような複合条件
 - 否定条件。NOT (A IS EQUAL TO B) など
 - 複合条件 (論理演算子 AND または OR を組み合わせた条件)

関連概念

91 ページの『スイッチおよびフラグ』

関連タスク

- 『スイッチおよびフラグの定義』
- 92 ページの『スイッチとフラグのリセット』
- 54 ページの『非互換データの検査 (数値のクラス・テスト)』
- 202 ページの『国別 (UTF-16) データの比較』
- 207 ページの『有効な DBCS 文字に関するテスト』

関連参照

- 329 ページの『UPSI』
- 一般比較条件 (COBOL for AIX 言語解説書)
- クラス条件 (COBOL for AIX 言語解説書)
- 条件名項目の規則 (COBOL for AIX 言語解説書)
- 符号条件 (COBOL for AIX 言語解説書)
- 複合条件 (COBOL for AIX 言語解説書)

スイッチおよびフラグ

プログラム中のいくつかの決定は、データ項目の値が真か偽か、オンかオフか、はいかいいえかに基づきます。スイッチとして働くレベル 88 項目に意味のある名前 (条件名) を付けて定義して、これらの両方向決定を制御してください。

その他のプログラム決定は、データ項目の特定の値または値の範囲に依存します。フィールドにオンまたはオフ以外の値を与えるために条件名を使用するときには、そのフィールドはフラグと呼ばれるのが普通です。

フラグおよびスイッチを使用すると、コードの変更が容易になります。条件の値を変更する必要がある場合は、そのレベル 88 条件名の値を変更するだけで済みます。

例えば、特定の給与範囲についてフィールドをテストするために、プログラムが条件名を使用するとします。別の給与範囲を検査するようにプログラムを変更しなければならない場合は、DATA DIVISION 内の条件名の値を変更するだけで済みます。PROCEDURE DIVISION で変更を行う必要はありません。

関連タスク

- 『スイッチおよびフラグの定義』
- 92 ページの『スイッチとフラグのリセット』

スイッチおよびフラグの定義

DATA DIVISION では、スイッチまたはフラグとして機能するレベル 88 項目を定義して、それらに分かりやすい名前を与えます。

フラグを持つ 2 つを超える値をテストするには、複数のレベル 88 項目を使用することによって、フィールドに複数の条件名を割り当ててください。

意味のある条件名が選択されており、かつそれらに割り当てられた値が論理値に関連付けられているならば、プログラムを読むときコードを追跡するのが容易になります。

- 92 ページの『例: スイッチ』
- 92 ページの『例: フラグ』

例: スイッチ

以下の例は、レベル 88 項目を使用してプログラム内のさまざまな 2 進値 (オン/オフ) 条件をテストする方法を示しています。

例えば、Transaction-File という名前の入力ファイルのファイル終了 (EOF) 条件をテストするには、データ定義を以下のように記述できます。

```
Working-Storage Section.  
01 Switches.  
    05 Transaction-EOF-Switch Pic X value space.  
    88 Transaction-EOF      value "y".
```

レベル 88 記述では、Transaction-EOF-Switch の値が 'y' なら、Transaction-EOF という名前の条件がオンになることが指定されています。PROCEDURE DIVISION 内で Transaction-EOF を参照することは、Transaction-EOF-Switch = "y" をテストすることと同じ条件を表します。例えば、次のステートメントによって報告書が印刷されるのは、Transaction-EOF-Switch が 'y' に設定されている場合に限られます。

```
If Transaction-EOF Then  
    Perform Print-Report-Summary-Lines
```

例: フラグ

以下の例は、EVALUATE ステートメントと一緒にいくつかのレベル 88 項目を使用して、プログラム内のいくつかある条件のうちどれが真であるかを判別する方法を示しています。

例えば、マスター・ファイルを更新するプログラムを考えてみましょう。更新内容は、トランザクション・ファイルから読み取られます。ファイル内のレコードには、3 つの機能 (追加、変更、または削除) のうちどれを実行するかを指示するフィールドが入っています。入力ファイルのレコード記述で、レベル 88 項目を使用して機能コード用のフィールドをコーディングします。

```
01 Transaction-Input Record  
    05 Transaction-Type      Pic X.  
    88 Add-Transaction      Value "A".  
    88 Change-Transaction   Value "C".  
    88 Delete-Transaction   Value "D".
```

これらの条件名をテストしてどの機能が実行されるかを判別するための、PROCEDURE DIVISION 内のコードは、次のようになります。

```
Evaluate True  
    When Add-Transaction  
        Perform Add-Master-Record-Paragraph  
    When Change-Transaction  
        Perform Update-Existing-Record-Paragraph  
    When Delete-Transaction  
        Perform Delete-Master-Record-Paragraph  
End-Evaluate
```

スイッチとフラグのリセット

プログラムの随所で、スイッチまたはフラグを、それらのデータ記述における元の値にリセットすることが必要になる場合があります。そのためには、SET ステートメントを使用するか、データ項目をスイッチまたはフラグに移動するように定義します。

SET *condition-name* TO TRUE ステートメントを使用すると、スイッチまたはフラグは、データ記述の中で割り当てられた元の値に設定されます。複数の値を持つレベル 88 項目の場合、SET *condition-name* TO TRUE は、最初の値 (次の例では A) を割り当てます。

```
88 Record-is-Active Value "A" "0" "S"
```

SET ステートメントと意味のある条件名を使用すれば、他のプログラマーにもコードが容易に追跡できるようになります。

『例: スイッチをオンに設定する』

『例: スイッチをオフに設定する』

例: スイッチをオンに設定する

以下の例は、値 TRUE をレベル 88 項目に移動する SET ステートメントをコーディングすることでスイッチをオンにする方法を示しています。

例えば、次の例にある SET ステートメントは、ステートメント Move "y" to Transaction-EOF-Switch をコーディングした場合と同じ働きをします。

```
01 Switches
   05 Transaction-EOF-Switch Pic X Value space.
   88 Transaction-EOF      Value "y".
. . .
Procedure Division.
000-Do-Main-Logic.
   Perform 100-Initialize-Paragraph
   Read Update-Transaction-File
   At End Set Transaction-EOF to True
End-Read
```

次の例では、入力レコードのトランザクション・コードに基づいて、出力レコード内のフィールドに値を割り当てる方法を示します。

```
01 Input-Record.
   05 Transaction-Type Pic X(9).
01 Data-Record-Out.
   05 Data-Record-Type Pic X.
   88 Record-Is-Active Value "A".
   88 Record-Is-Suspended Value "S".
   88 Record-Is-Deleted Value "D".
   05 Key-Field Pic X(5).
. . .
Procedure Division.
   Evaluate Transaction-Type of Input-Record
   When "ACTIVE"
     Set Record-Is-Active to TRUE
   When "SUSPENDED"
     Set Record-Is-Suspended to TRUE
   When "DELETED"
     Set Record-Is-Deleted to TRUE
End-Evaluate
```

例: スイッチをオフに設定する

次の例は、値をレベル 88 項目に移動する MOVE ステートメントをコーディングすることでスイッチをオフにする方法を示しています。

例えば、次のコードのように、SWITCH-OFF というデータ項目を使用してオン/オフ・スイッチをオフに設定できます。そうすると、ファイルの終わりに達していないことを示すようにスイッチがリセットされます。

```
01 Switches
   05 Transaction-EOF-Switch      Pic X Value space.
   88 Transaction-EOF            Value "y".
01 SWITCH-OFF                    Pic X Value "n".
. . .
Procedure Division.
. . .
   Move SWITCH-OFF to Transaction-EOF-Switch
```

プログラム・アクションの繰り返し

PERFORM ステートメントを使用すると、指定された回数だけ同じコードを繰り返すか (つまりループ)、判断の結果に基づいてループすることができます。

また、PERFORM ステートメントを使用すると、段落を実行し、その後で次の実行可能ステートメントに暗黙的に制御権を戻すようにすることもできます。実際には、この PERFORM ステートメントは、プログラムの異なる多くの部分から入ることができる閉じたサブルーチンをコーディングするための手段です。

PERFORM ステートメントはインラインまたはライン外にすることができます。

関連タスク

『インラインまたはライン外 PERFORM の選択』

95 ページの『ループのコーディング』

96 ページの『テーブルのループ処理』

97 ページの『複数の段落またはセクションの実行』

関連参照

PERFORM ステートメント (*COBOL for AIX 言語解説書*)

インラインまたはライン外 PERFORM の選択

インライン PERFORM は、プログラムの通常フローで実行される命令ステートメントです。ライン外 PERFORM は、指定された段落への分岐およびその段落からの暗黙の戻りを引き起こします。

インラインまたはライン外のいずれの PERFORM ステートメントをコーディングするかを決定するには、以下の点を考慮してください。

- PERFORM ステートメントを複数の場所で使用しますか。

プログラム内のいくつかの場所で同じコード部分を使用したい場合、ライン外 PERFORM を使用してください。

- どちらのステートメントの配置の方が読みやすいですか。

実行するコードが短い場合は、インライン PERFORM の方が読みやすくなります。ただし、コードがいくつもの画面にわたる場合は、ライン外 PERFORM を使用した

方が、プログラムのロジック・フローは分かりやすくなります。(ただし、構造化プログラミングの各段落は 1 つの論理機能を実行するようにする必要があります。)

- 効率性を優先させますか。

インライン PERFORM の場合は、ライン外 PERFORM で発生する分岐のオーバーヘッドが避けられます。しかし、ライン外 PERFORM コーディングでもコード最適化を利用できるので、効率性を過度に重要視する必要はありません。

1974 COBOL 標準では、PERFORM ステートメントはライン外であり、このため、別の段落への分岐と暗黙の戻りが必要になります。実行された段落が、プログラムのそれ以降の順次フローの中にある場合は、ロジック・フローの中でもう 1 度実行されます。この追加の実行を回避するためには、段落を通常の順次フローの外側 (例えば、GOBACK の後) に置くか、または段落のそばに分岐をコーディングしてください。

インライン PERFORM のサブジェクトは、命令ステートメントです。したがって、インライン PERFORM 内のステートメント (命令ステートメント以外) は、明示範囲終了符号を付けてコーディングしなければなりません。

『例: インライン PERFORM ステートメント』

例: インライン PERFORM ステートメント

この例は、必須の範囲終了符号と必須の END-PERFORM 句を持つインライン PERFORM ステートメントの構造を示しています。

```
Perform 100-Initialize-Paragraph
* The following statement is an inline PERFORM:
  Perform Until Transaction-EOF
    Read Update-Transaction-File Into WS-Transaction-Record
    At End
      Set Transaction-EOF To True
    Not At End
      Perform 200-Edit-Update-Transaction
      If No-Errors
        Perform 300-Update-Commuter-Record
      Else
        Perform 400-Print-Transaction-Errors
* End-If is a required scope terminator
  End-If
  Perform 410-Re-Initialize-Fields
* End-Read is a required scope terminator
  End-Read
End-Perform
```

ループのコーディング

PERFORM . . . TIMES ステートメントは、段落を指定された回数だけ実行する場合に使用します。

```
PERFORM 010-PROCESS-ONE-MONTH 12 TIMES
INSPECT . . .
```

上記の例では、制御が PERFORM ステートメントに達すると、段落 010-PROCESS-ONE-MONTH のコードが 12 回実行されてから、制御が INSPECT ステートメントに移ります。

PERFORM . . . UNTIL ステートメントは、選択した条件が満たされるまで段落を実行する場合に使用します。以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER . . . . UNTIL . . . .  
PERFORM . . . [WITH TEST BEFORE] . . . . UNTIL . . . .
```

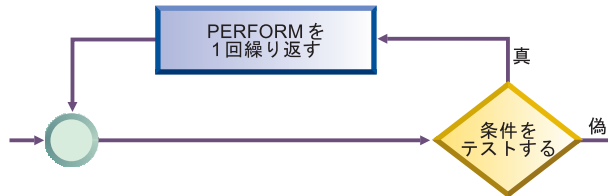
PERFORM . . . WITH TEST AFTER UNTIL ステートメントは、段落を少なくとも 1 回実行し、その後テストしてからそれ以降を実行したい場合に使用します。このステートメントは、do-until 構造と同等です。



次の例では、暗黙の WITH TEST BEFORE 句によって do-while 構造が提供されます。

```
PERFORM 010-PROCESS-ONE-MONTH  
  UNTIL MONTH GREATER THAN 12  
INSPECT . . .
```

制御が PERFORM ステートメントに達すると、条件 MONTH GREATER THAN 12 がテストされます。条件が満たされると、制御が INSPECT ステートメントに移ります。条件が満たされない場合には、010-PROCESS-ONE-MONTH が実行され、条件が再度テストされます。このサイクルは、条件が真になるまで継続されます。(プログラムを読みやすくするために、WITH TEST BEFORE 節をコーディングすることが必要な場合があります。)



テーブルのループ処理

PERFORM . . . VARYING ステートメントを使用すれば、テーブルを初期化できます。この形式の PERFORM ステートメントでは、条件が満たされるまで変数が増加または減少され、テストされます。

そのあと、PERFORM ステートメントを使用して、テーブルを操作するループを制御することができます。以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER . . . . VARYING . . . . UNTIL . . . .  
PERFORM . . . [WITH TEST BEFORE] . . . . VARYING . . . . UNTIL . . . .
```

以下のコードのセクションは、テーブル全体をループ処理して無効データがないか検査する例を示しています。

```
PERFORM TEST AFTER VARYING WS-DATA-IX  
  FROM 1 BY 1 UNTIL WS-DATA-IX = 12  
  IF WS-DATA (WS-DATA-IX) EQUALS SPACES  
  SET SERIOUS-ERROR TO TRUE
```



```
        DISPLAY ELEMENT-NUM-MSG5
      END-IF
    END-PERFORM
  INSPECT . . .
```

上記の PERFORM ステートメントに制御が達すると、WS-DATA-IX は 1 に設定され、PERFORM ステートメントが実行されます。その後、条件 WS-DATA-IX = 12 がテストされます。条件が真である場合には、制御が INSPECT ステートメントに渡ります。条件が偽である場合、WS-DATA-IX が 1 だけ増やされて、PERFORM ステートメントが実行され、条件が再度テストされます。この実行とテストのサイクルは、WS-DATA-IX が 12 になるまで継続されます。

上記のループは、項目 WS-DATA の 12 個のフィールドに関する入力検査を制御します。アプリケーションでは空のフィールドは許可されません。ですから、コードのセクションはループし、必要に応じてエラー・メッセージを発行します。

複数の段落またはセクションの実行

構造化プログラミングでは、通常、単一の段落を実行します。ただし、PERFORM . . . THRU ステートメントをコーディングすれば、段落グループ、単一セクション、またはセクション・グループを実行できます。

PERFORM . . . THRU ステートメントを使用するときには、段落 EXIT ステートメントをコーディングして、一連の段落のエンドポイントを明確に示してください。

関連タスク

83 ページの『組み込み関数を使用したテーブル項目の処理』

第 6 章 スtringの処理

COBOL は、String・データ項目に対して多種類の操作を実行するための言語構造体を提供します。

例えば、次のようなことが可能です。

- データ項目の結合または分割。
- ヌル終了Stringの操作 (文字のカウントや移動など)。
- 通常の位置 (必要があれば、および長さ) によるサブStringへの参照。
- データ項目の計算および置換 (データ項目内に特定文字が現れた回数のカウントなど)。
- データ項目の変換 (大文字または小文字への変更など)。
- データ項目の評価 (データ項目の長さの判別など)。

関連タスク

- 『データ項目の結合 (STRING)』
- 102 ページの『データ項目の分割 (UNSTRING)』
- 105 ページの『ヌル終了Stringの取り扱い』
- 106 ページの『データ項目のサブStringの参照』
- 110 ページの『データ項目の計算および置換 (INSPECT)』
- 111 ページの『データ項目の変換 (組み込み関数)』
- 114 ページの『データ項目の評価 (組み込み関数)』
- 183 ページの『第 10 章 国際環境でのデータの処理』

データ項目の結合 (STRING)

STRING ステートメントは、いくつかのデータ項目またはリテラルの、すべてまたは一部を 1 つのデータ項目に結合する場合に使用します。1 つの STRING ステートメントを、いくつかの MOVE ステートメントの代わりに使用できます。

STRING ステートメントは、示された順序でデータを受信データ項目に転送します。STRING ステートメントでは、以下のものも指定します。

- 送信フィールド・セットごとの区切り文字。検出されると、これにより送信フィールドの転送は停止されます (DELIMITED BY 句)
- (オプション) すべての送信データが処理される前に受信フィールドが満杯になっている場合に実行する処置 (ON OVERFLOW 句)
- (オプション) データの転送先である受信フィールド内の左端文字位置を示す、整数データ項目 (WITH POINTER 句)

受信データ項目を編集項目にしてはなりません。また表示浮動小数点項目や国別浮動小数点項目にしてもなりません。受信データ項目が何を持っているかによって次のような違いが生じます。

- USAGE DISPLAY を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE DISPLAY を持っている必要があり、ステートメント内のそれぞれのリテラルは英数字でなければなりません。
- USAGE NATIONAL を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE NATIONAL を持っている必要があり、ステートメント内のそれぞれのリテラルは国別でなければなりません。
- USAGE DISPLAY-1 を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE DISPLAY-1 を持っている必要があり、ステートメント内のそれぞれのリテラルは DBCS でなければなりません。

STRING ステートメントによってデータが書き込まれる、受信フィールドの特定部分のみが変更されます。

『例: STRING ステートメント』

関連タスク

169 ページの『ストリングの結合および分割におけるエラーの処理』

関連参照

STRING ステートメント (COBOL for AIX 言語解説書)

例: STRING ステートメント

次の例は、レコードから情報を選択して出力行にフォーマットする STRING ステートメントを示しています。

FILE SECTION は以下のレコードを定義します。

```
01 RCD-01.
   05 CUST-INFO.
       10 CUST-NAME    PIC X(15).
       10 CUST-ADDR   PIC X(35).
   05 BILL-INFO.
       10 INV-NO      PIC X(6).
       10 INV-AMT     PIC $$,$$$.$99.
       10 AMT-PAID    PIC $$,$$$.$99.
       10 DATE-PAID   PIC X(8).
       10 BAL-DUE     PIC $$,$$$.$99.
       10 DATE-DUE    PIC X(8).
```

WORKING-STORAGE SECTION は以下の各フィールドを定義します。

```
77 RPT-LINE          PIC X(120).
77 LINE-POS          PIC S9(3).
77 LINE-NO           PIC 9(5) VALUE 1.
77 DEC-POINT         PIC X VALUE ".".
```

レコード RCD-01 には、以下の情報が含まれています (記号 *b* はブランク・スペースを示します)。

```
J.B.bSMITHbbbb
444bSPRINGbST.,bCHICAGO,bILL.bbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
```

PROCEDURE DIVISION では、以下の設定値は STRING ステートメントの前にきます。

- RPT-LINE は SPACES に設定されます。
- LINE-POS (POINTER フィールドとして使用されるデータ項目) は 4 に設定されます。

STRING ステートメントを以下に示します。

```
STRING  
  LINE-NO SPACE CUST-INFO INV-NO SPACE DATE-DUE SPACE  
  DELIMITED BY SIZE  
  BAL-DUE  
  DELIMITED BY DEC-POINT  
  INTO RPT-LINE  
  WITH POINTER LINE-POS.
```

| STRING ステートメントが実行される前、POINTER フィールドの LINE-POS は値 4
| を持っているので、データは受信フィールド RPT-LINE に移動される時、文字位置
| 4 から開始されます。位置 1 から 3 の文字は未変更のままです。

| DELIMITED BY SIZE を指定している送信項目は、その全体が受信フィールドに移動
| されます。BAL-DUE は DEC-POINT で区切られているので、受信フィールドへの
| BAL-DUE の移動は、小数点 (DEC-POINT の値) が検出されると停止します。

STRING の結果

STRING ステートメントが実行されると、次の表に示されるように、項目は RPT-LINE に移動されます。

項目	位置
LINE-NO	4 - 8
スペース	9
CUST-INFO	10 - 59
INV-NO	60 - 65
スペース	66
DATE-DUE	67 - 74
スペース	75
BAL-DUE の小数点より前の部分	76 - 81

STRING ステートメントの実行後、LINE-POS の値は 82 であり、RPT-LINE は以下に示す値を持ちます。

Column

4	10		60	67	76
↓	↓		↓	↓	↓
00001	J.B. SMITH	444 SPRING ST., CHICAGO, ILL.	A14275	10/22/76	\$2,336

データ項目の分割 (UNSTRING)

UNSTRING ステートメントは、送信フィールドを複数の受信フィールドに分割するために使用します。1 つの UNSTRING ステートメントを、いくつかの MOVE ステートメントの代わりに使用できます。

UNSTRING ステートメントでは、以下のものを指定することができます。

- 区切り文字。区切り文字の 1 つが送信フィールドで検出されると、現行受信フィールドは受け取りを停止し、次のフィールド (ある場合) が受け取りを開始します (DELIMITED BY 句)
- 区切り文字用のフィールド。送信フィールドで区切り文字が検出されると、現行受信フィールドは受け取りを停止します (DELIMITER IN 句)
- 現行受信フィールドに入れられた文字の数を保管する整数データ項目 (COUNT IN 句)
- UNSTRING 処理が開始される送信フィールド内の左端文字位置を示す、整数データ項目 (WITH POINTER 句)
- 操作対象の受信フィールドの数の計算値を保管する、整数データ項目 (TALLYING IN 句)
- 送信データ項目の最後に達する前にすべての受信フィールドが満杯になった場合に実行する処置 (ON OVERFLOW 句)

送信データ項目および DELIMITED BY 句の区切り文字は、カテゴリー英字、英数字、英数字編集、DBCS、国別、または国別編集にする必要があります。

受信データ項目は、カテゴリー英字、英数字、数値、DBCS、または国別にすることができます。数値の受信データ項目は、ゾーン 10 進数または国別 10 進数にする必要があります。受信データ項目が何を持っているかによって次のような違いが生じます。

- USAGE DISPLAY を持っている場合、送信項目およびステートメント内のそれぞれの区切り文字項目は USAGE DISPLAY を持っている必要があります、ステートメント内のそれぞれのリテラルは英数字でなければなりません。
- USAGE NATIONAL を持っている場合、送信項目およびステートメント内のそれぞれの区切り文字項目は USAGE NATIONAL を持っている必要があります、ステートメント内のそれぞれのリテラルは国別でなければなりません
- USAGE DISPLAY-1 を持っている場合、送信項目およびステートメント内のそれぞれの区切り文字項目は USAGE DISPLAY-1 を持っている必要があります、ステートメント内のそれぞれのリテラルは DBCS でなければなりません

103 ページの『例: UNSTRING ステートメント』

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

169 ページの『ストリングの結合および分割におけるエラーの処理』

関連参照

UNSTRING ステートメント (COBOL for AIX 言語解説書)

データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)

例: UNSTRING ステートメント

次の例は、選択した情報を入力レコードから転送する UNSTRING ステートメントを示しています。情報の中には、印刷用に編成されているものもあれば、その後の処理用に編成されているものもあります。

FILE SECTION は以下のレコードを定義します。

* Record to be acted on by the UNSTRING statement:

```
01 INV-RCD.
   05 CONTROL-CHARS          PIC XX.
   05 ITEM-INDENT             PIC X(20).
   05 FILLER                  PIC X.
   05 INV-CODE                PIC X(10).
   05 FILLER                  PIC X.
   05 NO-UNITS                PIC 9(6).
   05 FILLER                  PIC X.
   05 PRICE-PER-M            PIC 99999.
   05 FILLER                  PIC X.
   05 RTL-AMT                 PIC 9(6).99.
```

*

* UNSTRING receiving field for printed output:

```
01 DISPLAY-REC.
   05 INV-NO                  PIC X(6).
   05 FILLER                  PIC X VALUE SPACE.
   05 ITEM-NAME               PIC X(20).
   05 FILLER                  PIC X VALUE SPACE.
   05 DISPLAY-DOLS           PIC 9(6).
```

*

* UNSTRING receiving field for further processing:

```
01 WORK-REC.
   05 M-UNITS                 PIC 9(6).
   05 FIELD-A                 PIC 9(6).
   05 WK-PRICE REDEFINES FIELD-A PIC 9999V99.
   05 INV-CLASS               PIC X(3).
```

*

* UNSTRING statement control fields:

```
77 DBY-1                     PIC X.
77 CTR-1                      PIC S9(3).
77 CTR-2                      PIC S9(3).
77 CTR-3                      PIC S9(3).
77 CTR-4                      PIC S9(3).
77 DLTR-1                     PIC X.
77 DLTR-2                     PIC X.
77 CHAR-CT                    PIC S9(3).
77 FLDS-FILLED                PIC S9(3).
```

PROCEDURE DIVISION では、以下の設定値は UNSTRING ステートメントの前にきます。

- ピリオド (.) は、DBY-1 内では区切り文字として配置されます。
- CHAR-CT (POINTER フィールド) は 3 に設定します。
- 値ゼロ (0) を FLDS-FILLED (TALLYING フィールド) に入れます。
- データは読み取られてレコード INV-RCD に入れられます。このレコードのフォーマットを以下のとおりです。

Column						
1	10	20	30	40	50	60
↓	↓	↓	↓	↓	↓	↓
ZYFOUR-PENNY-NAILS			707890/BBA	475120	00122	000379.50

UNSTRING ステートメントを以下に示します。

```
* Move subfields of INV-RCD to the subfields of DISPLAY-REC
* and WORK-REC:
  UNSTRING INV-RCD
    DELIMITED BY ALL SPACES OR "/" OR DBY-1
    INTO ITEM-NAME      COUNT IN CTR-1
      INV-NO            DELIMITER IN DLTR-1  COUNT IN CTR-2
      INV-CLASS
      M-UNITS          COUNT IN CTR-3
      FIELD-A
      DISPLAY-DOLS DELIMITER IN DLTR-2  COUNT IN CTR-4
    WITH POINTER CHAR-CT
    TALLYING IN  FLDS-FILLED
    ON OVERFLOW GO TO UNSTRING-COMPLETE.
```

UNSTRING ステートメントの実行前には POINTER フィールドである CHAR-CT の値は 3 であるので、INV-RCD 内の CONTROL-CHARS フィールドの 2 つの文字位置は無視されます。

UNSTRING の結果

この UNSTRING ステートメントを実行すると、以下のステップで処理が行われます。

- INV-RCD の桁 3 から 18 (FOUR-PENNY-NAILS) が ITEM-NAME に入れられ、区域内で左寄せされ、未使用の 4 つの文字位置にスペースが埋め込まれます。値 16 が CTR-1 に入れられます。
- ALL SPACES が区切り文字としてコーディングされているので、桁 19 から 23 の 5 つの連続スペース文字は 1 つの区切り文字とみなされます。
- 桁 24 から 29 (707890) が INV-NO に入れられます。区切り文字のスラッシュ (/) が DLTR-1 に入れられ、値 6 が CTR-2 に入れられます。
- 桁 31 から 33 (BBA) が INV-CLASS に入れられます。区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 34 のスペースは迂回されます。
- 桁 35 から 40 (475120) が M-UNITS に入れられます。値 6 が CTR-3 に入れられます。区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 41 のスペースは迂回されます。
- 桁 42 から 46 (00122) が FIELD-A に入れられ、領域内で右寄せされます。高位桁位置にはゼロ (0) が埋め込まれます。区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 47 のスペースは迂回されます。
- 桁 48 から 53 (000379) が DISPLAY-DOLS に入れられます。DBY-1 内のピリオド (.) 区切り文字が DLTR-2 に入れられ、値 6 が CTR-4 に入れられます。
- すべての受信フィールドに対して操作が行われたが、INV-RCD の 2 文字が検査されなかったため、ON OVERFLOW ステートメントが実行されます。UNSTRING ステートメントの実行は完了です。

UNSTRING ステートメントの実行後、フィールドには以下の値が入っています。

フィールド	値
DISPLAY-REC	707890 FOUR-PENNY-NAILS 000379
WORK-REC	475120000122BBA
CHAR-CT (POINTER フィールド)	55
FLDS-FILLED (TALLYING フィールド)	6

ヌル終了ストリングの取り扱い

さまざまな手段を使って、ヌル終了ストリング (例えば、C プログラムとの間でやり取りされるストリング) を構成し取り扱うことができます。

例えば、次のようなことが可能です。

- ヌル終了リテラル定数 (Z". . . ")。
- INSPECT ステートメントを使用して、ヌル終了ストリング内の文字数をカウントする。

```
MOVE 0 TO char-count
INSPECT source-field TALLYING char-count
                        FOR CHARACTERS
                        BEFORE X"00"
```

- UNSTRING ステートメントを使用して、ヌル終了ストリング内の文字をターゲット・フィールドに移動し、文字カウントを得る。

```
WORKING-STORAGE SECTION.
01 source-field          PIC X(1001).
01 char-count           COMP-5 PIC 9(4).
01 target-area.
   02 individual-char OCCURS 1 TO 1000 TIMES DEPENDING ON char-count
                        PIC X.
```

```
. . .
PROCEDURE DIVISION.
  UNSTRING source-field DELIMITED BY X"00"
                        INTO target-area
                        COUNT IN char-count

  ON OVERFLOW
    DISPLAY "source not null terminated or target too short"
  END-UNSTRING
```

- SEARCH ステートメントを使用して、後続ヌルまたはスペース文字を見つける。検査するストリングを単一文字からなるテーブルとして定義してください。
- ループのフィールドの各文字を検査する (PERFORM)。フィールドの各文字は、source-field (I:1) のような参照修飾子を使用して検査することができます。

106 ページの『例: ヌル終了ストリング』

関連タスク

522 ページの『ヌル終了ストリングの処理』

関連参照

英数字リテラル (COBOL for AIX 言語解説書)

例: ヌル終了ストリング

以下の例は、ヌル終了ストリングを処理できるいくつかの方法を示しています。

```
01 L pic X(20) value z'ab'.
01 M pic X(20) value z'cd'.
01 N pic X(20).
01 N-Length pic 99 value zero.
01 Y pic X(13) value 'Hello, World!'.
. . .
* Display null-terminated string:
  Inspect N tallying N-length
  for characters before initial x'00'
  Display 'N: ' N(1:N-Length) ' Length: ' N-Length
. . .
* Move null-terminated string to alphanumeric, strip null:
  Unstring N delimited by X'00' into X
. . .
* Create null-terminated string:
  String Y delimited by size
  X'00' delimited by size
  into N.
. . .
* Concatenate two null-terminated strings to produce another:
  String L delimited by x'00'
  M delimited by x'00'
  X'00' delimited by size
  into N.
```

データ項目のサブストリングの参照

参照修飾子を使用することにより、USAGE DISPLAY、DISPLAY-1、または NATIONAL を持つデータ項目のサブストリングを参照します。参照修飾子を使用すると、組み込み関数によって戻される英数字または国別文字ストリングのサブストリングを参照することもできます。

以下の例は、参照修飾子を使用して、Customer-Record という名前のデータ項目の 20 文字のサブストリングを参照する方法を示しています。

```
Move Customer-Record(1:20) to Orig-Customer-Name
```

データ項目の直後に括弧で囲んだ参照修飾子をコーディングします。例に示されるように、参照修飾子は、次の順で、コロンの分離された 2 つの値を含むことができます。

1. サブストリングを開始したい文字の序数位置 (左からの)
2. (オプション) 望ましいサブストリングの長さ (文字位置の数)

USAGE DISPLAY を持つ項目の参照修飾子の位置および長さは、1 バイト文字で表されます。USAGE DISPLAY-1 または NATIONAL を持つ項目の参照修飾子の位置および長さは、それぞれ DBCS 文字位置および国別文字位置で表されます。

参照修飾子の長さを省略すると (先頭文字の序数位置とその後のコロンのみをコーディングすると)、サブストリングは項目の最後まで延長されます。可能であれば、より単純でエラーになりにくいコーディング手法として、長さを省略してください。

参照修飾子を使用すると、英数字グループ、英数字編集データ項目、数字編集データ項目、表示浮動小数点データ項目、およびゾーン 10 進数データ項目を含め、USAGE DISPLAY データ項目のサブストリングを参照できます。これらのデータ項目のいずれかを参照修飾した場合、結果はカテゴリ-英数字になります。英字データ項目を参照修飾した場合、結果はカテゴリ-英字になります。

参照修飾子を使用すると、国別グループ、国別編集データ項目、数字編集データ項目、国別浮動小数点データ項目、および国別 10 進数データ項目を含め、USAGE NATIONAL データ項目のサブストリングを参照することができます。これらのデータ項目のいずれかを参照修飾した場合、結果はカテゴリ-国別になります。例えば、次のように国別 10 進数データ項目を定義するとしましょう。

```
01 NATL-DEC-ITEM Usage National Pic 999 Value 123.
```

NATL-DEC-ITEM はカテゴリ-数値なので、NATL-DEC-ITEM を算術式で使用できません。しかし、NATL-DEC-ITEM(2:1) (国別文字 2、16 進数表記では NX"0032") はカテゴリ-国別なので、これを算術式で使用することはできません。

参照修飾子を使用すると、可変長項目を含め、テーブル項目のサブストリングを参照することができます。テーブル記入項目のサブストリングを参照するには、参照修飾子の前に添え字式をコーディングします。例えば、PRODUCT-TABLE は、正しくコーディングされた文字ストリング・テーブルであると想定しましょう。D をテーブル内の 2 番目のストリングの 4 文字目に移動するために、次のステートメントをコーディングできます。

```
MOVE 'D' to PRODUCT-TABLE (2), (4:1)
```

参照修飾子の中の 2 つの値の一方または両方を、変数または算術式としてコーディングできます。

109 ページの『例: 参照修飾子としての演算式』

数字関数 ID は、算術式を使用できる場所ならどこでも使用できるので、左端文字位置または長さ (あるいはその両方) として、数字関数 ID を参照修飾子の中でコーディングできます。

109 ページの『例: 参照修飾子としての組み込み関数』

参照修飾子の中のそれぞれの数値は少なくとも 1 の値でなければなりません。サブストリングの最後を越えて参照することがないよう、2 つの数値の合計が、データ項目の全長を 2 文字位置以上超えるようなことがあってはなりません。

左端の文字位置または長さ値が固定小数点の非整数の場合には、整数を作成するために切り捨てが行われます。浮動小数点の非整数の場合には、整数を作成するための丸めが行われます。

以下のオプションを使用すると、範囲外の参照修飾子が検出され、実行時メッセージによって違反が示されます。

- SSRANGE コンパイラー・オプション
- CHECK ランタイム・オプション

関連概念

『参照修飾子』

188 ページの『Unicode および言語文字のエンコード』

関連タスク

69 ページの『テーブル内の項目の参照』

関連参照

309 ページの『SSRANGE』

参照変更 (COBOL for AIX 言語解説書)

関数定義 (COBOL for AIX 言語解説書)

参照修飾子

参照修飾子を使用すると、データ項目のサブストリングを容易に参照できます。

例えば、システムから現在の時刻を取り出して、その値を拡張形式で表示したいとします。現在の時刻は、ACCEPT ステートメントを使用して取り出すことができます。このステートメントは、次の形式で、時、分、秒、および 100 分の 1 秒を戻します。

```
HHMMSSss
```

しかし、現在の時刻を次の形式で表示したいとします。

```
HH:MM:SS
```

参照修飾子を使用しない場合は、両方の形式についてのデータ項目を定義しなければなりません。さらに、1 つの形式を別の形式に変換するためのコードも書く必要があります。

参照修飾子を使用する場合は、TIME エlementを記述するサブフィールドに名前を指定する必要はありません。必要なデータ定義は、システムによって戻される時刻用のデータ定義だけです。以下に例を示します。

```
01 REFMOD-TIME-ITEM PIC X(8).
```

次のコードは、時刻値を取り出して、拡張します。

```
ACCEPT REFMOD-TIME-ITEM FROM TIME.  
DISPLAY "CURRENT TIME IS: "  
* Retrieve the portion of the time value that corresponds to  
* the number of hours:  
REFMOD-TIME-ITEM (1:2)  
": "  
* Retrieve the portion of the time value that corresponds to  
* the number of minutes:  
REFMOD-TIME-ITEM (3:2)  
": "  
* Retrieve the portion of the time value that corresponds to  
* the number of seconds:  
REFMOD-TIME-ITEM (5:2)
```

109 ページの『例: 参照修飾子としての演算式』

109 ページの『例: 参照修飾子としての組み込み関数』

関連タスク

35 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』

106 ページの『データ項目のサブストリングの参照』
189 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

参照変更 (COBOL for AIX 言語解説書)

例: 参照修飾子としての演算式

あるフィールドに右揃えされたいいくつかの文字が入っている場合に、それらの文字を別のフィールドに移動し、右ではなく左に揃えたいとします。これは、参照修飾子と INSPECT ステートメントを使用して行うことができます。

プログラムに次のデータが入っているとします。

```
01 LEFTY      PIC X(30).  
01 RIGHTY     PIC X(30) JUSTIFIED RIGHT.  
01 I          PIC 9(9)  USAGE BINARY.
```

プログラムは、先行するスペースの数をカウントし、参照修飾子内の算術式を使用して、右揃えされた文字を別のフィールドに移動し、左寄せします。

```
MOVE SPACES TO LEFTY  
MOVE ZERO TO I  
INSPECT RIGHTY  
    TALLYING I FOR LEADING SPACE.  
IF I IS LESS THAN LENGTH OF RIGHTY THEN  
    MOVE RIGHTY ( I + 1 : LENGTH OF RIGHTY - I ) TO LEFTY  
END-IF
```

MOVE ステートメントは、RIGHTY の文字を、I + 1 で計算された位置から、LENGTH OF RIGHTY - I で計算された長さだけ、フィールド LEFTY に移動します。

例: 参照修飾子としての組み込み関数

コンパイル時にサブストリングの左端位置またはその長さを知らない場合、参照修飾子の中で組み込み関数を使用できます。

例えば、以下のコード・フラグメントにより、Customer-Record のサブストリングがデータ項目 WS-name に移動されます。サブストリングは、実行時に決定されます。

```
05 WS-name      Pic x(20).  
05 Left-posn    Pic 99.  
05 I            Pic 99.  
...  
Move Customer-Record(Function Min(Left-posn I):Function Length(WS-name)) to WS-name
```

整数関数を使わなければならない位置で非整数関数を使用したい場合、INTEGER または INTEGER-PART 関数を使用して結果を整数に変換できます。以下に例を示します。

```
Move Customer-Record(Function Integer(Function Sqrt(I)): ) to WS-name
```

関連参照

INTEGER (COBOL for AIX 言語解説書)

INTEGER-PART (COBOL for AIX 言語解説書)

データ項目の計算および置換 (INSPECT)

INSPECT ステートメントを使用して、データ項目内の文字または文字グループを検査し、必要に応じてそれらを置換します。

INSPECT ステートメントを使用して以下のタスクを行います。

- データ項目内に特定文字が現れる回数をカウントします (TALLYING 句)。
- データ項目またはデータ項目内の選択部分に、指定された文字 (スペース、アスタリスク、またはゼロなど) を充てんします (REPLACING 句)。
- データ項目内に特定文字または文字ストリングがあればそれらすべてを、指定された置換文字に変換します (CONVERTING 句)。

検査する項目として、以下のデータ項目の 1 つを指定できます。

- USAGE DISPLAY、USAGE DISPLAY-1、または USAGE NATIONAL として明示的または暗黙的に記述された基本項目
- 英数字グループ項目または国別グループ項目

検査する項目に何が指定されているかによって次のような違いが生じます。

- USAGE DISPLAY が指定されている場合、ステートメント内のそれぞれの ID は (TALLYING カウント・フィールドを除いて) USAGE DISPLAY が指定されている必要があります、ステートメント内のそれぞれのリテラルは英数字でなければなりません。
- USAGE NATIONAL が指定されている場合、ステートメント内のそれぞれの ID は (TALLYING カウント・フィールドを除いて) USAGE NATIONAL が指定されている必要があります、ステートメント内のそれぞれのリテラルは国別でなければなりません。
- USAGE DISPLAY-1 が指定されている場合、ステートメント内のそれぞれの ID は (TALLYING カウント・フィールドを除いて) USAGE DISPLAY-1 が指定されている必要があります、ステートメント内のそれぞれのリテラルは DBCS リテラルでなければなりません。

『例: INSPECT ステートメント』

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連参照

INSPECT ステートメント (COBOL for AIX 言語解説書)

例: INSPECT ステートメント

以下の例では、文字を検査して置き換える INSPECT ステートメントの使用法をいくつか示します。

次の例では、INSPECT ステートメントは、データ項目 DATA-2 内の文字を検査して置き換えます。データ項目内に現れる先行ゼロ (0) の数は、累算されて COUNTR に入れます。文字 C の最初のインスタンスの後に続く文字 A の最初のインスタンスは、文字 2 に置き換えられます。

```

77 COUNTR          PIC 9  VALUE ZERO.
01 DATA-2        PIC X(11).
. . .
INSPECT DATA-2
  TALLYING COUNTR FOR LEADING "0"
  REPLACING FIRST "A" BY "2" AFTER INITIAL "C"

```

DATA-2 (実行前)	COUNTR (実行後)	DATA-2 (実行後)
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATHAM0000	0	CH2THAM0000

次の例では、INSPECT ステートメントは、データ項目 DATA-3 内の文字を検査して置き換えます。引用符 (") の最初のインスタンスより前にある各文字は、文字 0 で置き換えられます。

```

77 COUNTR          PIC 9  VALUE ZERO.
01 DATA-3        PIC X(8).
. . .
INSPECT DATA-3
  REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE

```

DATA-3 (実行前)	COUNTR (実行後)	DATA-3 (実行後)
456"ABEL	0	000"ABEL
ANDES"12	0	00000"12
"TAS BR	0	"TAS BR

次の例では、AFTER 句と BEFORE 句を指定した INSPECT CONVERTING を使用して、データ項目 DATA-4 内の文字を検査して置き換えます。文字 / の最初のインスタンスより後にあり、文字 ? (もしあれば) の最初のインスタンスより前にあるすべての文字が、小文字から大文字に変換されます。

```

01 DATA-4        PIC X(11).
. . .
INSPECT DATA-4
  CONVERTING
    "abcdefghijklmnopqrstuvwxyz" TO
    "ABCDEFGHIJKLMNopRSTUVWXYZ"
  AFTER INITIAL "/"
  BEFORE INITIAL "?"

```

DATA-4 (実行前)	DATA-4 (実行後)
a/five/?six	a/FIVE/?six
r/Rexx/RRRr	r/REXX/RRRR
zfour?inspe	zfour?inspe

データ項目の変換 (組み込み関数)

組み込み関数を使用して、文字ストリング・データ項目を他のいくつかのフォーマットに (例: 大文字または小文字に、逆順に、数字に、あるコード・ページから別のコード・ページに) 変換できます。

NATIONAL-OF および DISPLAY-OF 組み込み関数を使用して、国別 (Unicode) ストリングとの間で変換を行うことができます。

INSPECT ステートメントを使用して文字を変換することもできます。

110 ページの『例: INSPECT ステートメント』

関連タスク

『大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)』

『逆順への変換 (REVERSE)』

113 ページの『数値への変換 (NUMVAL、NUMVAL-C)』

114 ページの『あるコード・ページから別のコード・ページへの変換』

大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)

UPPER-CASE および LOWER-CASE 組み込み関数を使用すれば、英数字、英字、または国別ストリングの大/小文字を容易に変更できます。

```
01 Item-1 Pic x(30) Value "Hello World!".
01 Item-2 Pic x(30).
. . .
  Display Item-1
  Display Function Upper-case(Item-1)
  Display Function Lower-case(Item-1)
  Move Function Upper-case(Item-1) to Item-2
  Display Item-2
```

上記のコードは、次のメッセージをシステムの論理出力装置に表示します。

```
Hello World!
HELLO WORLD!
hello world!
HELLO WORLD!
```

DISPLAY ステートメントは、Item-1 の実際の内容は変更せず、文字の表示方法にのみ影響を与えます。しかし、MOVE ステートメントでは、Item-2 の内容が大文字に置き換わります。

変換では、現在のロケールで定義された大/小文字マッピングを使用します。関数の結果の長さは、引数の長さとは異なることがあります。

関連タスク

35 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』

37 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

逆順への変換 (REVERSE)

REVERSE 組み込み関数を使用して、ストリング内の文字の順序を反転させることができます。

```
Move Function Reverse(Orig-cust-name) To Orig-cust-name
```

例えば、上記ステートメントは、Orig-cust-name 中の文字の順序を逆にします。開始値が JOHNSONbbb であれば、ステートメントの実行後の値は bbbNOSNHQJ になります (b はブランク・スペースを表します)。

関連概念

188 ページの『Unicode および言語文字のエンコード』

数値への変換 (NUMVAL、NUMVAL-C)

NUMVAL および NUMVAL-C 関数は、文字ストリング (英数字または国別リテラル、あるいはクラス英数字またはクラス国別データ項目) を数値に変換します。これらの関数を使用して、数値的に処理できるよう、フリー・フォーマット文字表現の数値を数値形式に変換します。

```
01 R      Pic x(20) Value "- 1234.5678".
01 S      Pic x(20) Value " $12,345.67CR".
01 Total  Usage is Comp-1.
. . .
Compute Total = Function Numval(R) + Function Numval-C(S)
```

NUMVAL-C は、上の例で示されているように、引数に通貨記号またはコンマ (またはその両方) が含まれているときに使用します。文字ストリングの前または後ろに代数符号を入れることができ、その符号が処理されます。引数は、デフォルト・オプション ARITH(COMPAT) (互換モード) でコンパイルするときは 18 桁を超えてはならず、ARITH(EXTEND) (拡張モード) でコンパイルするときは 31 桁を超えてはなりません (桁数には編集記号は含みません)。

NUMVAL および NUMVAL-C は、互換モードでは長精度 (64 ビット) 浮動小数点値を戻し、拡張モードでは拡張精度 (128 ビット) 浮動小数点値を戻します。これらの関数への参照は、数値データ項目への参照を表します。

最大でも、正確に長精度浮動小数点に変換できるのは 15 桁の 10 進数字までです (変換および精度に関する以下の関連参照で説明されています)。NUMVAL または NUMVAL-C の引数が 15 桁を超える場合、ARITH(EXTEND) コンパイラー・オプションを指定して、引数の値を正確に表現できる拡張精度関数結果が戻されるようにすることをお勧めします。

NUMVAL または NUMVAL-C を使用する場合は、数値データを固定形式で静的に宣言したり、入力データを正確な方法で入力したりする必要はありません。例えば、次のように入力される数値を定義するとします。

```
01 X      Pic S999V99 leading sign is separate.
. . .
Accept X from Console
```

アプリケーションのユーザーは、PICTURE 節で定義されているとおりに正確に数値を入力しなければなりません。以下に例を示します。

```
+001.23
-300.00
```

しかし、NUMVAL 関数を使用する場合は、次のようにコーディングすることができます。

```
01 A      Pic x(10).
01 B      Pic S999V99.
. . .
Accept A from Console
Compute B = Function Numval(A)
```

入力は次のようにすることができます。

1.23
-300

関連概念

45 ページの『数値データの形式』
51 ページの『データ形式の変換』
188 ページの『Unicode および言語文字のエンコード』

関連タスク

197 ページの『国別 (Unicode) 表現との変換』

関連参照

52 ページの『変換および精度』
268 ページの『ARITH』

あるコード・ページから別のコード・ページへの変換

DISPLAY-OF 組み込み関数と NATIONAL-OF 組み込み関数をネストすると、任意のコード・ページを別の任意のコード・ページに簡単に変換できます。

例えば、次のコードでは、EBCDIC のストリングを ASCII のストリングに変換しています。

```
77 EBCDIC-CCSID PIC 9(4) BINARY VALUE 1140.  
77 ASCII-CCSID PIC 9(4) BINARY VALUE 819.  
77 Input-EBCDIC PIC X(80).  
77 ASCII-Output PIC X(80).  
.  
.  
.  
* Convert EBCDIC to ASCII  
  Move Function Display-of  
    (Function National-of (Input-EBCDIC EBCDIC-CCSID),  
      ASCII-CCSID)  
  to ASCII-output
```

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

197 ページの『国別 (Unicode) 表現との変換』

データ項目の評価 (組み込み関数)

組み込み関数を使用して、照合シーケンス内の文字の序数位置を判別したり、一連のものから最大項目または最小項目を検出したり、データ項目の長さを検出したり、あるいはプログラムがコンパイルされた時間を判別したりできます。

以下の組み込み関数を使用します。

- CHAR および ORD。これらは、プログラム内で使用される照合シーケンスを基準にして、整数および単一の英字または英数字を評価するためのものです。
- MAX、MIN、ORD-MAX、および ORD-MIN。これらは、USAGE NATIONAL データ項目を含む一連のデータ項目から最大項目または最小項目を検出するためのものです。
- LENGTH。これは、データ項目の長さ (USAGE NATIONAL データ項目を含む) を調べるためのものです。

- WHEN-COMPILED。これは、プログラムがコンパイルされた日時を調べるためのものです。

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

『照合シーケンスに関する単一文字の評価』

『最大または最小データ項目の検出』

118 ページの『データ項目の長さの検出』

118 ページの『コンパイルの日付の検出』

照合シーケンスに関する単一文字の評価

照合シーケンス内のある特定の文字 (英字または英数字) の序数位置を検出するには、引数として文字を持つ ORD 関数を使用します。ORD はその序数位置を表す整数を返します。

以下のように、データ項目の 1 文字のサブストリングを ORD への引数として使用することができます。

```
IF Function Ord(Customer-record(1:1)) IS > 194 THEN . . .
```

ある文字の照合シーケンスにおける序数位置がわかっていて、それに対応する文字を検索する場合には、CHAR でその整数の序数位置を引数として使用することができます。CHAR は、要求された文字を返します。以下に例を示します。

```
INITIALIZE Customer-Name REPLACING ALPHABETIC BY Function Char(65)
```

関連参照

CHAR (COBOL for AIX 言語解説書)

ORD (COBOL for AIX 言語解説書)

最大または最小データ項目の検出

2 つ以上の英数字、英字、または国別データ項目のうち、どれが最大値を持つかを判別する場合には、MAX または ORD-MAX 組み込み関数を使用します。どの項目が最小値を持つかを判別するには、MIN または ORD-MIN を使用します。これらの関数は、照合シーケンスにしたがって評価します。

数値項目 (USAGE NATIONAL のあるものを含む) を比較するには、MAX、ORD-MAX、MIN、または ORD-MIN を使用します。これらの組み込み関数を使用すると、引数の代数值が比較されます。

MAX および MIN 関数は、指定された引数の 1 つの内容を返します。例えば、プログラムに以下のデータ定義が含まれているとします。

```
05 Arg1 Pic x(10) Value "THOMASSON ".  
05 Arg2 Pic x(10) Value "THOMAS ".  
05 Arg3 Pic x(10) Value "VALLEJO ".
```

次のステートメントは、VALLEJObbb を Customer-record の最初の 10 文字位置に割り当てます (ここで b はブランク・スペースを表します)。

Move Function Max(Arg1 Arg2 Arg3) To Customer-record(1:10)

代わりに MIN を使用すると、THOMAS**bbbb** が割り当てられます。

ORD-MAX および ORD-MIN 関数は、提供した引数のリストの中で最大値または最小値を持つ引数の (左からカウントした) 序数位置を表す整数を返します。上記の例で ORD-MAX 関数を使用すると、数字関数への参照が有効場所がないので、コンパイラーはエラー・メッセージを出します。以下に、ORD-MAX の有効な使用例を示します。

Compute x = Function Ord-max(Arg1 Arg2 Arg3)

上記のステートメントは、前の例と同じ引数を使用された場合、整数 3 を x に割り当てます。代わりに ORD-MIN を使用した場合には、整数 2 が返されます。Arg1、Arg2、および Arg3 が配列 (テーブル) の連続エレメントであったなら、上の例はもっと現実的なものになると思われます。

任意の引数に国別項目を指定する場合、すべての引数をクラス国別と指定する必要があります。

関連タスク

54 ページの『算術の実行』

83 ページの『組み込み関数を使用したテーブル項目の処理』

『英数字または国別関数によって戻される可変長結果』

関連参照

MAX (COBOL for AIX 言語解説書)

MIN (COBOL for AIX 言語解説書)

ORD-MAX (COBOL for AIX 言語解説書)

ORD-MIN (COBOL for AIX 言語解説書)

英数字または国別関数によって戻される可変長結果

英数字または国別関数の結果は、関数引数によって、長さや値が異なることがあります。

次の例では、R3 に移動されるデータの量および COMPUTE ステートメントの結果は、R1 および R2 の値とサイズによって異なります。

```
01 R1    Pic x(10) value "e".
01 R2    Pic x(05) value "f".
01 R3    Pic x(20) value spaces.
01 L     Pic 99.
. . .
      Move Function Max(R1 R2) to R3
      Compute L = Function Length(Function Max(R1 R2))
```

このコードの結果は次のようになります。

- R2 は R1 より大きいものと評価されます。
- スtring 'f**bbbb**' が R3 に移動されます (ここで *b* はブランク・スペースを表します)。(R3 内の残りの文字位置にはスペースが埋められます。)
- L は値 5 と評価されます。

R1 が「e」ではなく「g」を含んでいたなら、コードの結果は以下のようになります。

- R1 は、R2 より大きいと評価されます。
- スtring `gbbbbbbbb` が R3 に移動されます。(R3 内の残りの文字位置にはスペースが埋められます。)
- 値 10 が L に割り当てられます。

プログラムが関数引数として国別データを使用する場合、同様に関数結果の長さおよび値が変化します。例えば、以下のコードは上のフラグメントと等しいですが、英数字データではなく国別データを使用します。

```
01 R1    Pic n(10) national value "e".
01 R2    Pic n(05) national value "f".
01 R3    Pic n(20) national value spaces.
01 L     Pic 99    national.
. . .
      Move Function Max(R1 R2) to R3
      Compute L = Function Length(Function Max(R1 R2))
```

このコードの結果は以下のようになります。これらは、国別文字についてのものであることを除けば、最初の結果のセットと似ています。

- R2 は R1 より大きいものと評価されます。
- String NX"0066 0020 0020 0020 0020" (国別文字 `fbbbb` と同等のもの。ここで、*b* はブランク・スペース) は、ここでは読みやすくするためスペースが挿入された 16 進表記で示されており、R3 に移動されます。R3 内の空白文字位置には、国別スペースが埋め込まれます。
- L は値 5 (R2 の国別文字位置の長さ) と評価されます。

英数字または国別の関数からの可変長出力を取り扱うことがあります。それに応じてプログラムを設計しなければなりません。例えば、書き込むレコードによって長さが異なる可能性があるときには、可変長レコード・ファイルの使用を考慮することが必要になります。

```
File Section.
FD Output-File Recording Mode V.
01 Short-Customer-Record Pic X(50).
01 Long-Customer-Record  Pic X(70).
Working-Storage Section.
01 R1    Pic x(50).
01 R2    Pic x(70).
. . .
      If R1 > R2
          Write Short-Customer-Record from R1
      Else
          Write Long-Customer-Record from R2
      End-if
```

関連タスク

115 ページの『最大または最小データ項目の検出』

54 ページの『算術の実行』

関連参照

MAX (COBOL for AIX 言語解説書)

データ項目の長さの検出

LENGTH 関数を多くのコンテキスト (テーブルおよび数値データを含む) で使用して、項目の長さを判別することができます。例えば、LENGTH 関数を呼び出して、英数字または国別リテラルの長さ、あるいは DBCS 以外の不特定タイプのデータ項目の長さを判別できます。

LENGTH 関数は、国別項目 (リテラル、あるいは USAGE NATIONAL を持つ任意の項目 (国別グループ項目を含む)) の長さを、引数の長さ (国別文字位置の数) に等しい整数として戻します。これは、他の任意のデータ項目の長さを、引数の長さ (英数字位置の数) に等しい整数として戻します。

次の COBOL ステートメントは、データ項目を、顧客名を入れるレコードのフィールドに移動する例を示しています。

```
Move Customer-name To Customer-record(1:Function Length(Customer-name))
```

LENGTH OF 特殊レジスターを使用することもできます。この特殊レジスターは国別データの場合でも、長さをバイト単位で戻します。Function Length(Customer-name) または LENGTH OF Customer-name のどちらをコーディングしても、英数字項目の場合は同じ結果 (Customer-name のバイト単位の長さ) が戻されます。

LENGTH 関数は、算術式が許可される場所でしか使用できません。しかし、LENGTH OF 特殊レジスターはさまざまなコンテキストで使用することができます。例えば、整数引数を受け入れる組み込み関数に対する引数として LENGTH OF 特殊レジスターを使用できます。(組み込み関数を LENGTH OF 特殊レジスターに対するオペランドとして使用することはできません。) LENGTH OF 特殊レジスターは、CALL ステートメントのパラメーターとして使用することもできます。

関連タスク

54 ページの『算術の実行』

77 ページの『可変長テーブルの作成 (DEPENDING ON)』

83 ページの『組み込み関数を使用したテーブル項目の処理』

関連参照

266 ページの『ADDR』

LENGTH (COBOL for AIX 言語解説書)

LENGTH OF (COBOL for AIX 言語解説書)

コンパイルの日付の検出

WHEN-COMPILED 組み込み関数を使用して、プログラムがいつコンパイルされたのかを知ることができます。21 文字の結果は、コンパイル時の 4 桁の年、月、日、および時刻 (時間、分、秒、および百分の 1 秒)、およびグリニッジ標準時との差 (時間と分) を示します。

最初の 16 桁の形式は次のとおりです。

```
YYYYMMDDhhmmsshh
```

代わりに WHEN-COMPILED 特殊レジスターを使用して、次のフォーマットで、コンパイルの日時を知ることができます。

MM/DD/YYhh.mm.ss

WHEN-COMPILED 特殊レジスターは、2桁の年のみをサポートし、時刻を秒までしか扱いません。この特殊レジスターは、MOVE ステートメントの送信フィールドとしてのみ使用できます。

関連参照

WHEN-COMPILED (*COBOL for AIX* 言語解説書)

第 7 章 ファイルの処理

ファイルからのデータ読み取りとファイルへのデータ書き込みは、大半の COBOL プログラムで重要となります。プログラムは情報を検索し、それを要求どおりに処理した後、結果を書き込みます。

ただし、処理を行う前に、各ファイルを識別してそれらの物理構造を記述し、順次、相対、索引付き、行順次のいずれでファイルが編成されているかを示す必要があります。ファイルを識別するには、ファイルとそのファイル・システムに名前を付ける必要があります。また、処理が正しく行われたことを確認するために、後で検査が可能なファイル状況フィールドを設定することも可能です。

ファイルの処理で実行できる主なタスクでは、まずファイルをオープンして読み取り、(ファイル編成およびアクセスのタイプに応じて) レコードの追加、置換、または削除を行います。

関連概念

124 ページの『ファイル・システム』

関連タスク

『ファイルの識別』

132 ページの『ファイル編成およびアクセス・モードの指定』

136 ページの『ファイル・オープン時のエラーからの保護』

136 ページの『ファイル状況フィールドの設定』

137 ページの『ファイル構造の詳細記述』

137 ページの『ファイルの入出力ステートメントのコーディング』

148 ページの『COBOL アプリケーションで SFS ファイルにアクセスできるようにする』

ファイルの識別

ENVIRONMENT DIVISION の INPUT-OUTPUT SECTION の FILE-CONTROL 段落では、以下に示されているような、使用するファイルの名前を指定する必要があります。また、オプションでファイル・システムの名前を指定できます。

```
SELECT file ASSIGN TO FileSystemID-Filename
```

file は、ファイルを参照するためにプログラム内部で使用される名前です。この名前は、必ずしも、システムに認識されているファイルの実際の名前でなくても構いません。

FileSystemID は、次のファイル・システムのいずれかを識別します。

VSAM 仮想記憶アクセス方式システム。 VSAM を VSA と省略できます。

VSAM ファイルまたは Encina® SFS ファイルにアクセスしたい場合は、VSAM ファイル・システムを指定します。

STL 標準言語ファイル・システム

RSD レコード順次区切りファイル・システム

ファイル・システムの指定がない場合は、ランタイム・オプション FILESYS を使用してファイル・システムが選択されます。FILESYS ランタイム・オプションのデフォルトは VSAM です。

Filename は、アクセスする物理ファイルの名前です。別の方法として、環境変数を使用して実行時にファイル名を指定することも可能です。

関連概念

124 ページの『ファイル・システム』

関連タスク

『VSAM ファイルの識別』

『Encina SFS ファイルの識別』

123 ページの『STL ファイルの識別』

123 ページの『RSD ファイルの識別』

関連参照

125 ページの『VSAM ファイル・システム』

125 ページの『SFS ファイル・システム』

127 ページの『STL ファイル・システム』

129 ページの『RSD ファイル・システム』

228 ページの『ランタイム環境変数』

328 ページの『FILESYS』

VSAM ファイルの識別

VSAM ファイルに VSAM ファイル・システムを使用する場合は、以下に示すように、SELECT 文節を使用してファイルを識別します。

```
SELECT file1 ASSIGN USING 'VSA-MyFile'
```

ランタイム・オプション FILESYS(VSA) が有効な場合は、次のようにファイルを識別します。

```
SELECT file1 ASSIGN TO 'MyFile'
```

環境変数 *MYFILE* (例: export *MYFILE*=VSA-*MYFILE*) を定義している場合は、次のようにファイルを識別します。

```
SELECT file1 ASSIGN TO MYFILE
```

Encina SFS ファイルの識別

SFS ファイルに VSAM ファイル・システムを使用する場合は、以下に示すように、SELECT 文節を使用してファイルを識別します。

```
SELECT file1 ASSIGN TO Myfile
```

Myfile (COBOL *assignment-name*) はランタイム環境変数です。export を次の形式で使用して、実行時に環境変数に値を割り当てます。

```
export Myfile="/./myEnc/mySFS/mySFSfil(/./myEnc/mySFS/  
mySFSfil;myaltfil1,./myEnc/mySFS/mySFSfil;myaltfil2)"
```

このコマンドは、以下の情報を提供します。

- `././myEnc/mySFS/` は、Encina サーバーの完全修飾名です。
- `mySFSfil` はベース SFS ファイルです。
- `././myEnc/mySFS/mySFSfil` は、完全修飾ベース・ファイル名です。
- `myaltfil1` および `myaltfil2` は、代替索引ファイルです。

代替索引ファイルの場合、ファイル名は、その完全修飾ベース・ファイル名と、その後セミコロン (;) および代替索引ファイル名が続く形式にする必要があります (`././myEnc/mySFS/mySFSfil;myaltfil1`)。

`export` コマンドでは、代替索引ファイルの指定と指定の間のコンマは必須です。

関連参照

125 ページの『SFS ファイル・システム』

STL ファイルの識別

ファイルに標準言語ファイル・システムを使用する場合は、以下に示すように、`SELECT` 文節を使用してファイルを識別します。

```
SELECT file1 ASSIGN USING 'STL-MyFile'
```

ランタイム・オプション `FILESYS(STL)` が有効な場合は、次のようにファイルを識別します。

```
SELECT file1 ASSIGN TO 'MyFile'
```

環境変数 `MYFILE` (例: `export MYFILE=STL-MYFILE`) を定義している場合は、次のようにファイルを識別します。

```
SELECT file1 ASSIGN TO MYFILE
```

RSD ファイルの識別

ファイルにレコード順次区切りファイル・システムを使用する場合は、以下に示すように、`SELECT` 文節を使用してファイルを識別します。

```
SELECT file1 ASSIGN USING 'RSD-MyFile'
```

ランタイム・オプション `FILESYS(RSD)` が有効な場合は、次のようにファイルを識別します。

```
SELECT file1 ASSIGN TO 'MyFile'
```

環境変数 `MYFILE` (例: `export MYFILE=RSD-MYFILE`) を定義している場合は、次のようにファイルを識別します。

```
SELECT file1 ASSIGN TO MYFILE
```

ファイル・システム

順次、相対、または索引付きで編成されたレコード単位ファイルへは、ファイル・システムを通してアクセスします。ファイル・システム関数を使用すると、これらのタイプのファイルのレコードを作成および操作することができます。

COBOL for AIX では、次のファイル・システムをサポートしています。

- VSAM ファイル・システム。VSAM ファイルまたは Encina SFS ファイルへのアクセスを可能にします。(Encina SFS は CICS で使用するファイル・システムです。) VSAM ファイル・システムは COBOL for AIX とともに提供されますが、Encina SFS は、CICS の一部として提供されており、CICS とは別個に注文することもできます。
- STL (標準言語) ファイル・システムは、基本的なローカル・ファイル機能を備えています。COBOL for AIX に備わっており、順次ファイル、相対ファイル、および索引付きファイルに対応しています。
- RSD (レコード順次区切り) ファイル・システムでは、他の言語で書かれたプログラムとデータを共用することができます。RSD ファイルは順次のみですが、レコード内のあらゆる COBOL データ型に対応しています。レコード内のテキスト・データは、大半のファイル・エディターで編集することができます。

大半のプログラムは、どのファイル・システムでも同じように動作します。ただし、あるファイル・システムで書き込まれたファイルを、別のファイル・システムで読み取ることはできません。

ファイル・システムを選択するには、割り当て名の環境変数を設定するか、FILESYS ランタイム・オプションを使用します。どのファイル・システムでも、COBOL ステートメントを使用して COBOL ファイルの読み取りと書き込みを行うことができます。

関連概念

130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』

関連タスク

121 ページの『ファイルの識別』

関連参照

125 ページの『VSAM ファイル・システム』

125 ページの『SFS ファイル・システム』

127 ページの『STL ファイル・システム』

129 ページの『RSD ファイル・システム』

131 ページの『ファイル・システム要約』

328 ページの『FILESYS』

627 ページの『付録 C. zSeries ホスト・データ形式についての考慮事項』

VSAM ファイル・システム

VSAM ファイル・システムは、VSAM ファイルと SFS ファイルをサポートします。このファイル・システムは、順次ファイル、索引付きファイル、および相対ファイルをサポートします。サポートされないファイル編成タイプは LINE SEQUENTIAL のみです。

OPEN OUTPUT ステートメントと WRITE ステートメントを使用して VSAM ファイルを作成する場合、ネイティブ・バイト・ストリーム・ファイル・システムに複数のファイルが作成されます。

- ファイル属性は、接尾部が .DDMEA となっているファイルに保管されます。
- 1 次索引は、ファイル名の先頭に文字 @ が付けられたファイルに保管されます。
- 代替索引は、接尾部が .@01 や .@02 などになっているファイルに保管されます。

このようなローカル VSAM 管理ファイルに、他の非 VSAM ソフトウェアでアクセスしないでください。

制約事項: 複数のスレッドとともに使用する場合、VSAM ファイル・システムは安全ではありません。

VSAM ファイル・システムは 標準 COBOL 85 に準拠しています。

関連概念

130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』

132 ページの『ファイル編成およびアクセス・モード』

関連タスク

122 ページの『VSAM ファイルの識別』

関連参照

131 ページの『ファイル・システム要約』

『SFS ファイル・システム』

VSAM ファイル・システム 応答メッセージ

SFS ファイル・システム

Encina SFS (Structured File Server) ファイル・システムは、順次、相対、および索引付き COBOL ファイルをサポートするレコード単位のファイル・システムです。このファイル・システムには、ファイルへの順次、ランダム、または動的アクセスに必要な、基本的な機能が備わっています。

SFS ファイルは、読み取り、挿入、更新、削除など、標準のファイル操作によって処理することができます。

制限: 64 ビット COBOL for AIX プログラムで SFS ファイルを処理することはできません。

SFS ファイルにアクセスするには、VSAM ファイル・システムを指定します。詳しくは、ファイルの識別に関する関連タスクを参照してください。

各 SFS ファイルには、ファイル内でのレコードの物理的順序を定義する 1 次索引が 1 つあり、任意の数の副次索引を格納することができます。副次索引は、レコードにアクセスできる代替シーケンスを提供します。

SFS ファイルのフルネームは、SFS ファイルが常駐する SFS サーバーの名前と、その後そのファイル自体の名前を続けて指定します。例えば、サーバー名が `./encina/sfs/my_server` でファイル名が `myfile` の場合、ファイルのフルネームは `./encina/sfs/my_server/myfile` になります。(この書式の名前を、完全修飾 ファイル名と呼びます。)

SFS ファイル・システムは、非階層型です。つまり、サーバー名の後に指定できるのは、ディレクトリー名ではなく、個々のファイル名のみです。

SFS ファイル・システムは、入力順 (順次)、相対、クラスター化 (索引付き) として知られている、3 つのタイプのファイル編成をサポートします。

SFS ファイル内のすべてのデータは、SFS サーバーによって管理されます。SFS には、システム管理ツール、`sfsadmin` を備えています。これは、ファイルおよび索引の作成や SFS サーバー上で使用可能なボリュームの判別などの管理機能を、コマンド行インターフェースを使用して実行するためのツールです。詳しくは、下記の関連参照にある Encina の資料を参照してください。

SFS ファイルへの COBOL アクセスは非トランザクション・アクセスです。すなわち、SFS ファイルに対する各操作は、全体として実行されるか、全く実行されないかのいずれかです (アトミック と呼ばれることがあります)。SFS システム障害が発生すると、COBOL アプリケーションが完了したファイルの操作の結果が SFS ファイルに反映されない場合があります。

SFS ファイル・システムを使用すると、PL/I プログラムと共用するファイルの読み取りと書き込みを容易に行うことができます。

関連概念

132 ページの『ファイル編成およびアクセス・モード』

関連タスク

121 ページの『ファイルの識別』

122 ページの『Encina SFS ファイルの識別』

148 ページの『COBOL アプリケーションで SFS ファイルにアクセスできるようにする』

151 ページの『SFS パフォーマンスの向上』

238 ページの『SFS ファイルのためのコンパイル』

Encina SFS Programming Guide

関連参照

131 ページの『ファイル・システム要約』

Encina Administration Guide Volume 2: Server Administration

STL ファイル・システム

STL ファイル・システム (標準言語ファイル・システム) は、順次ファイル、索引付きファイル、および相対ファイルに対応しています。これは、ファイルにアクセスするための基本のファイル機能を提供します。

STL ファイル・システムは 標準 COBOL 85 に準拠しており、パフォーマンスが高く、AIX システムと Windows ベースのシステムの間で容易に移植できることも特長です。LINE SEQUENTIAL 編成は、STL ファイル・システムによってサポートされない唯一のファイル編成タイプです。

STL ファイル・システムは、スレッドとともに使用しても安全です。ただし、複数のスレッドが同時にファイルのレベル 01 のレコードにアクセスしないようにしてください。複数のスレッドが同一の STL ファイルを操作することは可能ですが、`pthread_cond_wait` 呼び出しまたは同等のを使用して、アクティブ・スレッド上でのファイル・アクセスが完了するまで、他のすべてのスレッドを待機させるようにする必要があります。

関連概念

130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』

132 ページの『ファイル編成およびアクセス・モード』

関連タスク

123 ページの『STL ファイルの識別』

関連参照

『STL ファイル・システムの戻りコード』

131 ページの『ファイル・システム要約』

STL ファイル・システムの戻りコード

STL ファイルに対する入出力操作の後に、複数あるエラー・コードのうちの 1 つが発生する可能性があります。

FILE STATUS *data-name-1 data-name-8*

FILE-CONTROL 段落で、STL ファイルに関する、上述の文節をコーディングするとします。ファイルに対する入出力操作の後、*data-name-1* には使用するファイル・システムに依存しない状況コードが含まれ、*data-name-8* には次の表に示す STL ファイル・システムの戻りコードのいずれかが含まれます。

表9. STL ファイル・システムの戻りコード

コード	意味	注
0	正常終了	入出力操作が正しく完了しました。
1	無効な操作	この戻りコードは、通常は戻されません。これはファイル・システム内のエラーを示します。
2	入出力エラー	オペレーティング・システムの I/O ルーチンへの呼び出しからエラー・コードが戻されました。
3	ファイルがオープンしない	ファイルに対して (OPEN 以外の) 操作を試行しましたが、ファイルがオープンしません。

表9. STL ファイル・システムの戻りコード (続き)

コード	意味	注
4	キー値が検出されない	キーを使用してレコードを読み取ろうとしましたが、キーがファイル内にありません。
5	キー値の重複	重複できないキーを 2 回使用しようとした。
6	無効なキー番号	この戻りコードは、通常は戻されません。これはファイル・システム内のエラーを示します。
7	別のキー番号	この戻りコードは、通常は戻されません。これはファイル・システム内のエラーを示します。
8	操作に対する無効なフラグ	この戻りコードは、通常は戻されません。これはファイル・システム内のエラーを示します。
9	ファイルの終わり	ファイルの終わりを検出しました。これはエラーではありません。
10	I/O GET 操作の前に入出力操作が必要	この操作は現行レコードを検索しますが、現行レコードはまだ定義されていません。
11	スペース取得ルーチンから戻されたエラー	オペレーティング・システムがメモリー不足を示しています。
12	重複したキーの受け入れ	当該操作で指定したキーが重複しています。
13	順次アクセスおよびキー・シーケンスの不良	順次アクセスが指定されましたが、レコードが順序どおりになっていません。
14	レコード長 < 最大キー	レコード長で許可しているスペースが足りないため、すべてのキーを格納できません。
15	ファイルへのアクセス拒否	オペレーティング・システムがファイルにアクセスできないことを報告しました。当該ファイルが存在しないか、またはオペレーティング・システムがファイルにアクセスするための適切なアクセス権がユーザー側にありません。
16	ファイルがすでに存在する	新規ファイルをオープンしようとしたが、そのファイルがすでに存在することをオペレーティング・システムが報告しました。
17	(予約済み)	
18	ファイル・ロック	ファイルをオープンしようとしたが、そのファイルはすでに排他モードでオープンしています。
19	ファイル・テーブルがいっぱい	ファイル・テーブルがいっぱいであることをオペレーティング・システムが報告しました。
20	ハンドル・テーブルがいっぱい	これ以上ファイル・ハンドルを割り振れないことをオペレーティング・システムが報告しました。
21	タイトルが STL でない	STL ファイル・システムによって読み取りのためにオープンされたファイルには、ファイルの特定のオフセット位置に「STL」を含むヘッダー・レコードが入っていないなければなりません。
22	作成用の indexcount 引数が不良	この戻りコードは、通常は戻されません。これはファイル・システム内のエラーを示します。

表9. STL ファイル・システムの戻りコード (続き)

コード	意味	注
23	索引または相対レコード > 64 KB	索引および相対レコードの長さは 64 KB に制限されています。
24	オープン中の既存ファイルのファイル・ヘッダーまたはデータ内で検出されたエラー	STL ファイルはヘッダーで始まります。ヘッダーまたは関連データの値が矛盾しています。
25	順次ファイルの索引付きオープン	順次ファイルを索引付きファイルまたは相対ファイルとしてオープンしようとしていました。

次の表に、アダプター・オープン・ルーチンで検出されるエラーに対する戻りコードを示します。

表10. STL ファイル・システムのアダプター・オープン・ルーチン戻りコード

コード	意味	注
1000	索引付きまたは相対ファイルの順次オープン	索引付きファイルまたは相対ファイルを順次ファイルとしてオープンしようとしていました。
1001	索引付きファイルの相対オープン	相対ファイルを索引付きファイルとしてオープンしようとしていました。
1002	順次ファイルの索引付きオープン	索引付きファイルを順次ファイルとしてオープンしようとしていました。
1003	ファイルが存在しない	当該ファイルが存在しないことをオペレーティング・システムが報告しました。
1004	キー数が異なる	キー数が異なるファイルをオープンしようとしていました。
1005	レコード長が異なる	レコード長が異なるファイルをオープンしようとしていました。
1006	レコード・タイプが異なる	レコード・タイプが異なるファイルをオープンしようとしていました。
1007	キー位置または長さが異なる	キー位置または長さが異なるファイルをオープンしようとしていました。

関連タスク

174 ページの『ファイル状況キーの使用』

関連参照

127 ページの『STL ファイル・システム』

FILE STATUS 節 (COBOL for AIX 言語解説書)

RSD ファイル・システム

RSD (レコード順次区切り) ファイル・システムは、順次ファイルに対応していません。RSD ファイルは、ブラウズ、編集、コピー、削除、印刷などの標準的なシステム・ファイル・ユーティリティー機能を使用して処理することができます。

RSD ファイルは優れたパフォーマンスを提供します。また、AIX システムと Windows ベースのシステムとの間でファイルを容易に移植でき、異なる言語で書かれたプログラムやアプリケーションの間でファイルを共用することができます。

RSD ファイルは、固定長レコード内のあらゆる COBOL データ型に対応しています。書き込まれた各レコードの後には、改行制御文字が続きます。読み取り操作の場合は、ファイル・レコード域にレコード長を示すデータが格納されます。このレコード域にデータが格納される前にファイル・マークが検出された場合は、レコード域にスペース文字が埋め込まれます。

RSD ファイルをテキスト・エディターで編集する場合は、各レコードの長さを必ず維持してください。例えば、ファイル *infile* を編集するときに、各レコードを必ず 80 文字にしたい場合は、各レコードの列 80 にスペースを埋め込み、次のコマンドを発行して結果をファイル *outfile* に書き込みます。

```
awk '{printf("%*-80s\n", $0)}' infile > outfile
```

RSD ファイルに対して AFTER ADVANCING または BEFORE ADVANCING 句を持つ WRITE ステートメントを試行すると、このステートメントは失敗し、ファイル状況キーが 30 に設定されます。

RSD ファイル・システムは、スレッドとともに使用しても安全です。ただし、複数のスレッドが同時にファイルのレベル 01 のレコードにアクセスしないようにしてください。複数のスレッドが同一の RSD ファイルを操作することは可能ですが、`pthread_cond_wait` 呼び出しなどを使用して、アクティブ・スレッド上でファイル・アクセスが完了するまで、1 つのスレッドを除くすべてのスレッドを待機させるようにする必要があります。

関連概念

132 ページの『ファイル編成およびアクセス・モード』

関連タスク

123 ページの『RSD ファイルの識別』

2 GB を超える STL ファイルおよび VSAM ファイル

IBM COBOL for AIX のバージョン 3 以降では、以前のバージョンの COBOL for AIX にあった STL ファイルと VSAM ファイルの最大サイズ 2 GB の制限はなくなりました。

デフォルトでは、OPEN OUTPUT ステートメントを使用して作成される STL ファイルと VSAM ファイルは、大容量形式と呼ばれるメタデータ (内部制御情報) を持ちます。このメタデータにより、増やされたファイル・サイズがサポートされます。一般に、この機能は COBOL アプリケーション・プログラムに対して透過的です。増やされたのは最大ファイル・サイズのみであることに注意してください。個々のレコードの最大サイズは増やされていません。

最新のランタイム・ライブラリーを使用して COBOL for AIX バージョン 2 のプログラムを実行する場合は、COBOL for AIX バージョン 2 のプログラムでも大容量形式ファイル进行处理できます。COBOL for AIX バージョン 2 のランタイム・ライブラリーを使用して実行される COBOL for AIX バージョン 2 のプログラムでは、大容量形式ファイル进行处理できません。

バージョン 3 より前の COBOL for AIX のプログラムによって作成された STL ファイルと VSAM ファイルは、小容量形式 メタデータを持ちます。小容量形式ファイルは OPEN INPUT ステートメント、OPEN I-O ステートメント、または OPEN EXTEND ステートメントでオープンできますが、ファイルは 2 GB の最大サイズに制限されたままです。

COBOL for AIX バージョン 2 との互換性を保つために、COBOL for AIX プログラムの実行時にランタイム・オプション FILEMODE(SMALL) を指定して、小容量形式の STL ファイルと VSAM ファイルを作成するように、そのプログラムに対して指示できます。それらのファイルは、バージョン 2 (またはそれ以降) のランタイム・ライブラリーを使用して実行されるバージョン 2 のプログラムで処理できます。

小容量形式の STL ファイルと VSAM ファイルは PL/I プログラムと共用できません。

関連タスク

121 ページの『ファイルの識別』

関連参照

125 ページの『VSAM ファイル・システム』

127 ページの『STL ファイル・システム』

327 ページの『FILEMODE』

コンパイラ限界値 (COBOL for AIX 言語解説書)

ファイル・システム要約

次の表に、どのタイプの COBOL プログラムがどのタイプのファイルを処理できるのかを示します。X は、列の上部に示されたプログラムのタイプがファイルを処理できることを表します。

表 11. ファイル・システム要約

ファイル・システムおよび形式	2 GB を超えるファイル・サイズにできるか?	32 ビット COBOL for AIX プログラム	64 ビット COBOL for AIX プログラム	最新ランタイム・ライブラリーを使用するバージョン 2 COBOL for AIX プログラム	バージョン 2 ランタイム・ライブラリーを使用するバージョン 2 COBOL for AIX プログラム
STL 小容量形式	いいえ	X	X	X	X
STL 大容量形式 ¹	はい	X	X	X ²	
VSAM 小容量形式	いいえ	X	X	X	X
VSAM 大容量形式 ¹	はい	X	X	X ²	
行順次	はい	X	X	X	X
RSD	はい	X	X	X	X
Encina SFS	COBOL 制限なし ³	X		X	X

表 11. ファイル・システム要約 (続き)

ファイル・システム および形式	2 GB を超える ファイル・サイ ズにできるか?	32 ビット COBOL for AIX プログラム	64 ビット COBOL for AIX プログラム	最新ランタイム・ ライブラリーを使 用するバージョン 2 COBOL for AIX プログラム	バージョン 2 ラ ンタイム・ライブ ラリーを使用する バージョン 2 COBOL for AIX プログラム
<p>1. ランタイム・オプション FILEMODE(LARGE) を有効にした状態で作成されたファイル。ファイル・サイズは、最大レコード数に最大レコード・サイズを掛けた値に制限されます。(詳しくは、下記のコンパイラ限界値に関する関連参照を参照してください。)</p> <p>2. これらのプログラムでは、既存の大容量形式ファイルを OPEN INPUT、OPEN I-O、または OPEN EXTEND でオープンでき、これらのファイルを処理できます。ただし、これらのプログラムが OPEN OUTPUT を使用してファイルを作成する場合、そのファイルの形式は小容量形式になります。</p> <p>3. 制限と要件について詳しくは、Encina の資料を参照してください。</p>					

関連概念

130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』

関連参照

266 ページの『ADDR』

327 ページの『FILEMODE』

Encina Administration Guide Volume 2: Server Administration

コンパイラ限界値 (COBOL for AIX 言語解説書)

ファイル編成およびアクセス・モードの指定

FILE-CONTROL 段落では、以下に示すように、ファイルの物理構造とアクセス・モードを定義する必要があります。

```
FILE-CONTROL.
  SELECT file ASSIGN TO FileSystemID-Filename
  ORGANIZATION IS org ACCESS MODE IS access.
```

org に対しては、SEQUENTIAL (デフォルト)、LINE SEQUENTIAL、INDEXED、RELATIVE のいずれかを選択することができます。

access に対しては、SEQUENTIAL (デフォルト)、RANDOM、DYNAMIC のいずれかを選択することができます。

順次ファイルおよび行順次ファイルは、順次にアクセスする必要があります。索引付きファイルと相対ファイルの場合は 3 つのアクセス・モードをすべて使用することができます。

ファイル編成およびアクセス・モード

ファイル編成は、順次、行順次、索引付き、相対のいずれかの形式で行うことができます。アクセス・モードは、ファイルの編成方法ではなく、COBOL でのファイルの読み取り方法と書き込み方法を定義します。

プログラムの設計時には、ファイル編成とアクセス・モードを決定する必要があります。

次の表は、COBOL ファイルのファイル編成とアクセス・モードをまとめたものです。

表 12. ファイル編成およびアクセス・モード

ファイル編成	レコードの順序	レコードは削除または置換可能か	アクセス・モード
順次	レコードが書き込まれた順序	レコードは削除できないが、そのスペースを同じ長さのレコードに再利用することが可能	順次のみ
行順次	レコードが書き込まれた順序	いいえ	順次のみ
索引付き	キー・フィールドによる照合シーケンス	はい	順次、ランダム、または動的
相対	相対レコード番号の順序	はい	順次、ランダム、または動的

ファイル管理システムは、入出力装置からの入出力要求とレコード検索を処理します。

関連概念

- 『順次ファイルの編成』
- 134 ページの『行順次ファイル編成』
- 134 ページの『索引付きファイル編成』
- 134 ページの『相対ファイル編成』
- 135 ページの『順次アクセス』
- 135 ページの『ランダム・アクセス』
- 135 ページの『動的アクセス』

関連タスク

- 132 ページの『ファイル編成およびアクセス・モードの指定』

順次ファイルの編成

順次ファイルに含まれるレコードは、それらが入力された順序で編成されています。レコードの順序は固定されます。

順次ファイル内のレコードの読み取りや書き込みは、順次形式でのみ行うことができます。

順次ファイルにレコードを挿入したら、そのレコードの長さを調整したり、削除することはできません。ただし、長さが変わらなければ、レコードを更新 (REWRITE) することはできます。新しいレコードはファイルの終わりに追加されます。

ファイル内のレコードの順序が重要でない場合は、レコード数の多少にかかわらず、順次編成を選択することをお勧めします。順次出力は、レポートを印刷する際にも便利です。

関連概念

135 ページの『順次アクセス』

関連参照

141 ページの『順次ファイルに有効な COBOL ステートメント』

行順次ファイル編成

行順次ファイルは順次ファイルと似ていますが、レコード内にデータとして文字しか含めることができない点が異なります。行順次ファイルは、オペレーティング・システムのネイティブ・バイト・ストリーム・ファイルでサポートされています。

ADVANCING 句を持つ WRITE ステートメントで作成される行順次ファイルは、プリンターまたはディスクに送信することができます。

関連概念

133 ページの『順次ファイルの編成』

関連参照

141 ページの『行順次ファイルに有効な COBOL ステートメント』

索引付きファイル編成

索引付きファイルには、レコード・キーによって順序指定されたレコードが含まれています。レコード・キーは、レコードを固有に識別し、それがアクセスされる順序を他のレコードとの関連で判別します。

各レコードには、レコード・キーを含むフィールドがあります。レコードのレコード・キーは、例えば、従業員番号や送り状番号にすることができます。

また、索引付きファイルでは、代替索引 (レコードの別の論理配置を使用してファイルにアクセスできるレコード・キー) を使用することも可能です。例えば、従業員番号ではなく、従業員の部門を介してファイルにアクセスすることができます。

索引付きファイルに対して使用できるレコード伝送 (アクセス) モードは、順次モード、ランダム・モード、または動的モードです。索引付きファイルの順次読み取りまたは書き込みの順序は、キー値の順序になります。

EBCDIC についての考慮事項: 照合シーケンスが変更された場合と同様に、索引付きファイルがローカル EBCDIC ファイルの場合は、EBCDIC キーが COBOL プログラムの外部にあるものとして認識されません。例えば、外部ソート・プログラムは、EBCDIC にも対応していない限り、期待どおりの順序でレコードをソートすることはありません。

関連参照

142 ページの『索引付きファイルおよび相対ファイルに有効な COBOL ステートメント』

相対ファイル編成

相対レコード・ファイルには、相対キー、すなわちファイルの先頭から相対したレコードの位置を表すレコード番号によって順序指定されたレコードが含まれています。

例えば、ファイルの最初のレコードの相対レコード番号は 1 で、10 番目のレコードの相対レコード番号は 10 というようになっています。レコードは、固定長でも可変長でも構いません。

相対ファイルについて認められるレコード伝送モードは、順次、ランダム、または動的です。相対ファイルの順次読み取りまたは書き込みの順序は、相対レコード番号の順序です。

関連参照

142 ページの『索引付きファイルおよび相対ファイルに有効な COBOL ステートメント』

順次アクセス

順次アクセスの場合は、FILE-CONTROL 段落で ACCESS IS SEQUENTIAL をコーディングします。

索引付きファイルのレコードは、ファイル位置標識の現在位置から始まって、選択されたキー・フィールド (基本または代替) の順にアクセスされます。

相対ファイルのレコードは、相対レコード番号の順にアクセスされます。

関連概念

『ランダム・アクセス』

『動的アクセス』

関連参照

140 ページの『ファイル位置標識』

ランダム・アクセス

ランダム・アクセスの場合は、FILE-CONTROL 段落で ACCESS IS RANDOM をコーディングします。

索引付きファイルのレコードは、キー・フィールドに入れられた値 (基本、代替、相対) に従ってアクセスされます。代替索引は 1 つ以上存在する可能性があります。

相対ファイルのレコードは、相対キーに入れられた値に従ってアクセスされます。

関連概念

『順次アクセス』

『動的アクセス』

動的アクセス

動的アクセスの場合は、FILE-CONTROL 段落で ACCESS IS DYNAMIC をコーディングします。

動的アクセスでは、同じプログラム内での順次アクセスとランダム・アクセスの混在がサポートされています。動的アクセスでは、順次処理とランダム処理の両方を実行する 1 つの COBOL ファイル定義を使用することで、あるレコードへは順次にアクセスし、別のレコードへはキーによってアクセスすることができます。

例えば、従業員レコードの索引付きファイルがあり、従業員の時間給がレコード・キーを形成しているものとします。また、プログラムでは、時間給が \$12.00 から \$18.00 の従業員と、時間給が \$25.00 以上の従業員について処理するものとします。この情報にアクセスするには、1200 のキーに基づいて (ランダム検索 READ を使用して) 最初のレコードをランダムに検索します。次に、給与フィールドが 1800 を超えるまで、(READ NEXT を使用して) 順次読み取りを行います。今度は 2500 のキーに基づいて、ランダム読み取りに切り替えます。このランダム読み取りの後、ファイルの終わりに到達するまで、順次読み取りを行います。

関連概念

135 ページの『順次アクセス』

135 ページの『ランダム・アクセス』

ファイル・オープン時のエラーからの保護

存在しないファイルのオープンや読み取りをプログラムが試みると、通常はエラーが発生します。

ただし、存在しないファイルのオープンが意味をなす場合もあります。このような場合は、SELECT とともにオプション・キーワード OPTIONAL を使用します。

```
SELECT OPTIONAL file ASSIGN TO filename
```

ファイル状況フィールドの設定

ファイル状況キーは、FILE-CONTROL 段落の FILE STATUS 文節および DATA DIVISION のデータ定義を使用して設定します。

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    ..  
    FILE STATUS IS file-status  
WORKING-STORAGE SECTION.  
01 file-status PIC 99.
```

ファイル状況キー *file-status* は、2 文字のカテゴリ英数字またはカテゴリ国別項目として、あるいは 2 桁のゾーン 10 進数 (USAGE DISPLAY) (上記に示すとおり) または国別 10 進数 (USAGE NATIONAL) 項目として指定します。

制限: FILE STATUS 文節で参照されるデータ項目を可変的な場所に置くことはできません。例えば、可変長テーブルの後に置くことはできません。

関連タスク

174 ページの『ファイル状況キーの使用』

関連参照

FILE STATUS 節 (*COBOL for AIX* 言語解説書)

ファイル構造の詳細記述

DATA DIVISION の FILE SECTION では、キーワード FD と、FILE-CONTROL 段落内の対応する SELECT 文節で使用したファイル名を使用して、ファイル記述を開始します。

```
DATA DIVISION.  
FILE SECTION.  
FD filename  
01 recordname  
   nn . . . fieldlength & type  
   nn . . . fieldlength & type  
   . . .
```

上記の例では、*filename* は、OPEN、READ、および CLOSE ステートメントでも使用されるファイル名です。

recordname は、WRITE および REWRITE ステートメントで使用されるレコードの名前です。1 つのファイルに対して複数のレコードを指定することができます。

fieldlength はフィールドの論理長です。*type* はフィールドの形式を指定します。この方法でレコード記述項目をレベル 01 以上に分ける場合、各エLEMENTはレコードのフィールドに対して正確にマップしなければなりません。

関連参照

データ関係 (COBOL for AIX 言語解説書)

レベル番号 (COBOL for AIX 言語解説書)

PICTURE 節 (COBOL for AIX 言語解説書)

USAGE 節 (COBOL for AIX 言語解説書)

ファイルの入出カステートメントのコーディング

ENVIRONMENT DIVISION および DATA DIVISION でファイルの識別と記述を行った後、プログラムの PROCEDURE DIVISION でファイル・レコードを処理することができます。

COBOL プログラムは、使用するファイルのタイプ (順次、行順次、索引付き、相対のいずれか) に従ってコーディングします。入力および出力のコーディングを行う際の一般的なフォーマット (後述の例を参照) には、ファイルのオープン、読み取り、情報の書き込み、およびクローズが含まれます。

138 ページの『例: COBOL でのファイルのコーディング』

関連タスク

121 ページの『ファイルの識別』

132 ページの『ファイル編成およびアクセス・モードの指定』

136 ページの『ファイル・オープン時のエラーからの保護』

136 ページの『ファイル状況フィールドの設定』

『ファイル構造の詳細記述』

140 ページの『ファイルのオープン』

143 ページの『ファイルからのレコードの読み取り』

145 ページの『ファイルへのレコードの追加』

- 145 ページの『ファイル内のレコードの置換』
- 146 ページの『ファイルからのレコードの削除』

関連参照

- 140 ページの『ファイル位置標識』
- 144 ページの『ファイルへのレコード書き込み時に使用するステートメント』
- 147 ページの『ファイルの更新に使用する PROCEDURE DIVISION ステートメント』

例: COBOL でのファイルのコーディング

入出力コーディングの一般形式を、以下の例に示します。コードの後、ユーザーが指定した情報 (例内の小文字のテキスト) について説明します。

```
IDENTIFICATION DIVISION.
. . .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT filename ASSIGN TO assignment-name (1) (2)
    ORGANIZATION IS org ACCESS MODE IS access (3) (4)
    FILE STATUS IS file-status (5)
. . .
DATA DIVISION.
FILE SECTION.
FD filename
01 recordname (6)
   nn . . . fieldlength & type (7) (8)
   nn . . . fieldlength & type
. . .
WORKING-STORAGE SECTION.
01 file-status PIC 99.
. . .
PROCEDURE DIVISION.
    OPEN iomode filename (9)
. . .
    READ filename
. . .
    WRITE recordname
. . .
    CLOSE filename
STOP RUN.
```

(1) *filename*

任意の有効な COBOL 名。SELECT 文節および FD 項目、および OPEN、READ、START、DELETE、および CLOSE の各ステートメントでは、同じファイル名を使用する必要があります。この名前は、必ずしも、システムに認識されているファイルの実際の名前でなくても構いません。各ファイルには、独自の SELECT 節、FD 項目、および入出力ステートメントが必要です。WRITE および REWRITE では、ファイルに対して定義されたレコードを指定します。

(2) *assignment-name*

ASSIGN TO *assignment-name* をコーディングして、システムに認識されているターゲット・ファイル・システム ID とファイル名を直接指定するか、または環境変数を使用して値を間接的に設定することができます。

OPEN の際にシステム・ファイル名を識別したい場合は、ASSIGN USING *data-name* を指定できます。当該ファイルに対して OPEN ステートメントが

実行されるときには、*data-name* の値が使用されます。また、オプションにより、ファイル・システムのタイプを識別することでシステム・ファイルの識別を処理することも可能です。

次の例は、*inventory-file* を動的に (MOVE ステートメントにより) ファイル */user/inventory/parts* と関連付ける方法を示しています。

```
SELECT inventory-file ASSIGN USING a-file . . .
. . .
FD inventory-file . . .
. . .
77 a-file PIC X(25) VALUE SPACES.
. . .
    MOVE "/user/inventory/parts" TO a-file
    OPEN INPUT inventory-file
```

次の例は、*inventory-file* を (MOVE ステートメントにより) 索引ファイル *parts* と動的に関連付け、代替索引ファイル *altpart1* および *altpart2* を Encina サーバーの完全修飾名 (この例では *./Encina/sfs*) と関連付ける方法を示しています。

```
SELECT inventory-file ASSIGN USING a-file . . .
ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS FILESYSFILE-KEY
    ALTERNATE RECORD KEY IS ALTKEY1
    ALTERNATE RECORD KEY IS ALTKEY2. . . .
. . .
FILE SECTION.
FD inventory-file . . .
. . .
WORKING-STORAGE SECTION.
01 a-file PIC X(80) VALUE ' ' . . . .
. . .
    MOVE "/./Encina/sfs/parts(./Encina/sfs/parts;altpart1,./Encina/sfs/parts;altpart2)" TO a-file
    OPEN INPUT inventory-file
```

(3) *org*

編成 (SEQUENTIAL、LINE SEQUENTIAL、INDEXED、または RELATIVE) を示します。この文節を省略した場合は、デフォルトの ORGANIZATION SEQUENTIAL が使用されます。

(4) *access*

アクセス・モード (SEQUENTIAL、RANDOM、または DYNAMIC) を示します。この文節を省略した場合は、デフォルトの ACCESS SEQUENTIAL が使用されます。

(5) *file-status*

| COBOL ファイル状況キー。 ファイル状況キーを、2 文字の英数字または
| 国別データ項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進
| 数項目として指定してください。

(6) *recordname*

WRITE および REWRITE ステートメントで使用されるレコードの名前。1 つのファイルに対して複数のレコードを指定することができます。

(7) *fieldlength*

フィールドの論理長。

(8) *type*

ファイルのレコード形式と一致していなければなりません。レコード記述項目をレベル 01 以上に分ける場合は、各エレメントはレコードのフィールドに対して正確にマップします。

(9) *iomode*

オープン・モードを指定します。ファイルから読み取りだけを行う場合は、INPUT をコーディングします。ファイルへの書き込みだけを行う場合は、OUTPUT (新規ファイルのオープンまたは既存ファイル上での書き込み) または EXTEND (ファイル末尾へのレコードの追加) をコーディングします。読み取りと書き込みの両方を行う場合は、I-O をコーディングします。

制限: 行順次ファイルの場合、I-O は OPEN の有効なパラメーターにはなりません。

ファイル位置標識

ファイル位置標識は、順次 COBOL 要求の場合にアクセスされる次のレコードを示します。

ファイル位置標識は、プログラマーがプログラム内に設定するものではありません。この標識は、成功した OPEN、START、READ、READ NEXT、および READ PREVIOUS ステートメントによって設定されます。その後の READ、READ NEXT、または READ PREVIOUS 要求は、設定されたファイル位置標識の位置を使用した後、それを更新します。

ファイル位置標識は、出力ステートメント WRITE、REWRITE、または DELETE によって使用されたり、影響を受けたりすることはありません。ファイル位置標識は、ランダム処理では意味がありません。

ファイルのオープン

WRITE、START、READ、REWRITE、または DELETE ステートメントを使用してファイル内のレコードを処理するためには、前もってそのファイルを OPEN ステートメントでオープンしておかなければなりません。

PROCEDURE DIVISION.

```
      . . .  
      OPEN iomode filename
```

上記の例では、*iomode* はオープン・モードを指定します。ファイルから読み取りだけを行う場合は、オープン・モードに対して INPUT をコーディングします。ファイルへの書き込みだけを行う場合は、オープン・モードに対して OUTPUT (新規ファイルのオープンまたは既存ファイル上での書き込み) または EXTEND (ファイル末尾へのレコードの追加) をコーディングします。

すでにレコードが入っているファイルをオープンするには、OPEN INPUT、OPEN I-O (行順次ファイルの場合は無効)、または OPEN EXTEND を使用してください。

レコードが格納されている SFS ファイルに対して OPEN OUTPUT を使用すると、COBOL ランタイムがそのファイルを削除し、その後で COBOL で提供される属性

を持つファイルを作成します。SFS ファイルを削除したくないバックアップは、OPEN I-O とコーディングしてファイルをオープンします。

順次ファイル、行順次ファイル、または相対ファイルを EXTEND としてオープンする場合には、ファイル内の最後の既存レコードの後に、追加されたレコードが置かれます。索引付きファイルを EXTEND としてオープンする場合には、追加するそれぞれのレコードが、ファイル内の最高位レコードよりも大きいレコード・キーを持っていなければなりません。

関連概念

- 130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』
- 132 ページの『ファイル編成およびアクセス・モード』

関連タスク

- 136 ページの『ファイル・オープン時のエラーからの保護』

関連参照

- 『順次ファイルに有効な COBOL ステートメント』
- 『行順次ファイルに有効な COBOL ステートメント』
- 142 ページの『索引付きファイルおよび相対ファイルに有効な COBOL ステートメント』
- OPEN ステートメント (*COBOL for AIX 言語解説書*)

順次ファイルに有効な COBOL ステートメント

次の表に、順次ファイルに対して使用可能な入出力ステートメントの組み合わせを示します。「X」のマークは、当該列の最上部に記載されているオープン・モードで、当該ステートメントが使用できることを示しています。

表 13. 順次ファイルに有効な COBOL ステートメント

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X	X	X
	WRITE		X		X
	START				
	READ	X		X	
	REWRITE			X	
	DELETE				
	CLOSE	X	X	X	X

関連概念

- 133 ページの『順次ファイルの編成』
- 135 ページの『順次アクセス』

行順次ファイルに有効な COBOL ステートメント

次の表に、行順次ファイルに対して使用可能な入出力ステートメントの組み合わせを示します。「X」のマークは、当該列の最上部に記載されているオープン・モードで、当該ステートメントが使用できることを示しています。

表 14. 行順次ファイルに有効な COBOL ステートメント

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X		X
	WRITE		X		X
	START				
	READ	X			
	REWRITE				
	DELETE				
	CLOSE	X	X		X

関連概念

134 ページの『行順次ファイル編成』

135 ページの『順次アクセス』

索引付きファイルおよび相対ファイルに有効な COBOL ステートメント

次の表に、索引付きファイルおよび相対ファイルに対して使用可能な入出力ステートメントの組み合わせを示します。「X」のマークは、当該列の最上部に記載されているオープン・モードで、当該ステートメントが使用できることを示しています。

表 15. 索引付きファイルおよび相対ファイルに有効な COBOL ステートメント

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
順次	OPEN	X	X	X	X
	WRITE		X		X
	START	X		X	
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	X
ランダム	OPEN	X	X	X	
	WRITE		X	X	
	START				
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	

表 15. 索引付きファイルおよび相対ファイルに有効な COBOL ステートメント (続き)

アクセス・モード	COBOL ステートメント	OPEN INPUT	OPEN OUTPUT	OPEN I-O	OPEN EXTEND
動的	OPEN	X	X	X	
	WRITE		X	X	
	START	X		X	
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	

関連概念

- 134 ページの『索引付きファイル編成』
- 134 ページの『相対ファイル編成』
- 135 ページの『順次アクセス』
- 135 ページの『ランダム・アクセス』
- 135 ページの『動的アクセス』

ファイルからのレコードの読み取り

READ ステートメントを使用して、ファイルからレコードを取り出します。レコードを読み取るには、OPEN INPUT または OPEN I-O (OPEN I-O は行順次ファイルでは無効) を使用してファイルをオープンする必要があります。各 READ の後で、ファイル状況キーを検査してください。

順次ファイルおよび行順次ファイルの中のレコードは、それらが書き込まれたシーケンスでしか検索することができません。

索引付きおよび相対レコード・ファイルの中のレコードは、順次 (索引付きファイルの場合は使用中のキーの昇順に従い、相対ファイルの場合は相対レコード位置に従う)、ランダム、または動的に検索することができます。

動的アクセスの場合、順次検索には READ NEXT と READ PREVIOUS を使用し、(キーによる) ランダム検索には READ を使用することによって、特定のレコードの直接読み取りと、レコードの順次読み取りを切り替えることができます。

特定のレコードから順次に読み取りを行いたい場合には、READ NEXT または READ PREVIOUS ステートメントの前に START ステートメントを使用して、ファイル位置標識を、特定のレコードを指すように設定してください。START の後に READ NEXT をコーディングすると、次のレコードが読み取られ、ファイル位置標識は次のレコードにリセットされます。START の後に READ PREVIOUS をコーディングすると、前のレコードが読み取られ、ファイル位置標識は前のレコードにリセットされます。ファイル位置標識は、START を使用してランダムに移動することができますが、すべての読み取りはそのポイントから順次に行われることとなります。

レコードの順次読み取りを続行することも、START ステートメントを使用してファイル位置標識を移動することも可能です。以下に例を示します。

```
START file-name KEY IS EQUAL TO ALTERNATE-RECORD-KEY
```

重複が存在する代替索引に基づいて、索引付きファイルに対して直接 READ が実行されると、その代替キー値を持つファイル (基本クラスター) の最初のレコードのみが検索されます。同じ代替キーを持つデータ・セット・レコードのそれぞれを検索するためには、一連の READ NEXT ステートメントが必要です。同じ代替キーを持つ読み取り対象レコードが複数ある場合は、ファイル状況値 02 が戻されます。そのキー値を持つ最後のレコードが読み取られると、値 00 が戻されます。

関連概念

- 135 ページの『順次アクセス』
- 135 ページの『ランダム・アクセス』
- 135 ページの『動的アクセス』
- 132 ページの『ファイル編成およびアクセス・モード』

関連タスク

- 140 ページの『ファイルのオープン』
- 174 ページの『ファイル状況キーの使用』

関連参照

- 140 ページの『ファイル位置標識』
- FILE STATUS 節 (COBOL for AIX 言語解説書)

ファイルへのレコード書き込み時に使用するステートメント

次の表に、ファイルの作成または拡張時に使用できる COBOL ステートメントを示します。

表 16. ファイルへのレコード書き込み時に使用するステートメント

除算	順次	行順次	索引付き	相対
ENVIRONMENT	SELECT ASSIGN FILE STATUS ACCESS MODE	SELECT ASSIGN ORGANIZATION IS LINE SEQUENTIAL FILE STATUS ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY ALTERNATE RECORD KEY FILE STATUS ACCESS MODE	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS ACCESS MODE
DATA	FD 記入項目	FD 記入項目	FD 記入項目	FD 記入項目
PROCEDURE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE

関連概念

- 132 ページの『ファイル編成およびアクセス・モード』

関連タスク

- 132 ページの『ファイル編成およびアクセス・モードの指定』
- 140 ページの『ファイルのオープン』

136 ページの『ファイル状況フィールドの設定』
『ファイルへのレコードの追加』

関連参照

147 ページの『ファイルの更新に使用する PROCEDURE DIVISION ステートメント』

ファイルへのレコードの追加

COBOL の WRITE ステートメントは、既存のレコードを置き換えずに、ファイルにレコードを追加します。追加されるレコードは、ファイルの定義時に設定された最大レコード・サイズを超えてはなりません。それぞれの WRITE ステートメントの後で、ファイル状況キーを検査してください。

レコードを順次に追加する場合: OUTPUT または EXTEND としてオープンされたファイルの終わりにレコードを順次に追加するには、ACCESS IS SEQUENTIAL を使用し、WRITE ステートメントをコーディングしてください。

順次ファイルおよび行順次ファイルは、必ず順次に書き込まれます。

索引付きファイルの場合、新規のレコードは昇順キー配列で書き込む必要があります。ファイルが EXTEND としてオープンされている場合、追加されるレコードのレコード・キーは、ファイルがオープンされた時点のファイルの最高位基本レコード・キーよりも大きくなければなりません。

相対ファイルの場合、レコードは正しい順序になっていなければなりません。SELECT 節に RELATIVE KEY データ項目を組み込むと、書き込まれるレコードの相対レコード番号がそのデータ項目に入られます。

レコードをランダムまたは動的に追加する場合: ACCESS IS RANDOM または ACCESS IS DYNAMIC をコーディングしているレコードを索引付きデータ・セットに書き込むとき、レコードは任意の順序で書き込むことができます。

関連概念

130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』
132 ページの『ファイル編成およびアクセス・モード』

関連タスク

132 ページの『ファイル編成およびアクセス・モードの指定』
174 ページの『ファイル状況キーの使用』

関連参照

144 ページの『ファイルへのレコード書き込み時に使用するステートメント』
147 ページの『ファイルの更新に使用する PROCEDURE DIVISION ステートメント』
FILE STATUS 節 (COBOL for AIX 言語解説書)

ファイル内のレコードの置換

ファイル内のレコードを置換するには、I-O として開いたファイルに対して REWRITE を使用してください。ファイルが I-O としてオープンされなかった場合、

レコードは置換されず、状況キーが 49 に設定されます。それぞれの REWRITE ステートメントの後で、ファイル状況キーを検査してください。

順次ファイルの場合、置換レコードの長さは元のレコードの長さと同じでなければなりません。索引ファイルまたは可変長相対ファイルの場合、置換するレコードの長さを変更することができます。

レコードをランダムまたは動的に置換するには、まずレコードを READ する必要はありません。そうではなく、次のように置換対象のレコードを検索します。

- 索引付きファイルの場合、レコード・キーを RECORD KEY データ項目に移動してから、REWRITE を発行します。
- 相対ファイルの場合、相対レコード番号を RELATIVE KEY データ項目に移動してから、REWRITE を発行します。

圧縮フォーマットで割り振った拡張フォーマットのデータ・セットを、I-O としてオープンすることはできません。

関連概念

132 ページの『ファイル編成およびアクセス・モード』

関連タスク

140 ページの『ファイルのオープン』

174 ページの『ファイル状況キーの使用』

関連参照

FILE STATUS 節 (COBOL for AIX 言語解説書)

ファイルからのレコードの削除

既存のレコードを索引付きまたは相対ファイルから削除するには、ファイル I-O を開き、DELETE ステートメントを使用します。順次または行順次ファイルには DELETE を使用できません。

ACCESS IS SEQUENTIAL の場合は、削除するレコードを COBOL プログラムがまず読み取る必要があります。DELETE ステートメントは、読み取られたレコードを除去します。DELETE の前の READ が成功しなかった場合には、削除は行われず、ファイル状況キー値が 92 に設定されます。

ACCESS IS RANDOM または ACCESS IS DYNAMIC の場合は、削除するレコードを COBOL プログラムが読み取る必要はありません。レコードを削除するには、レコードのキーを RECORD KEY データ項目に移動してから、DELETE を発行します。

それぞれの DELETE ステートメントの後で、ファイル状況キーを検査してください。

関連概念

132 ページの『ファイル編成およびアクセス・モード』

関連タスク

140 ページの『ファイルのオープン』

143 ページの『ファイルからのレコードの読み取り』

174 ページの『ファイル状況キーの使用』

関連参照

FILE STATUS 節 (COBOL for AIX 言語解説書)

ファイルの更新に使用する PROCEDURE DIVISION ステートメント

次の表に、順次ファイル、行順次ファイル、索引付きファイル、および相対ファイルの PROCEDURE DIVISION で使用できるステートメントを示します。

表 17. ファイルの更新に使用する PROCEDURE DIVISION ステートメント

アクセス方式	順次	行順次	索引付き	相対
ACCESS IS SEQUENTIAL	OPEN EXTEND WRITE CLOSE または OPEN I-0 READ REWRITE CLOSE	OPEN EXTEND WRITE CLOSE	OPEN EXTEND WRITE CLOSE または OPEN I-0 READ REWRITE DELETE CLOSE	OPEN EXTEND WRITE CLOSE または OPEN I-0 READ REWRITE DELETE CLOSE
ACCESS IS RANDOM	適用されない	適用されない	OPEN I-0 READ WRITE REWRITE DELETE CLOSE	OPEN I-0 READ WRITE REWRITE DELETE CLOSE
ACCESS IS DYNAMIC (順次処理)	適用されない	適用されない	OPEN I-0 READ NEXT READ PREVIOUS START CLOSE	OPEN I-0 READ NEXT READ PREVIOUS START CLOSE
ACCESS IS DYNAMIC (ランダム処理)	適用されない	適用されない	OPEN I-0 READ WRITE REWRITE DELETE CLOSE	OPEN I-0 READ WRITE REWRITE DELETE CLOSE

関連概念

132 ページの『ファイル編成およびアクセス・モード』

関連タスク

140 ページの『ファイルのオープン』

143 ページの『ファイルからのレコードの読み取り』

- 145 ページの『ファイルへのレコードの追加』
- 145 ページの『ファイル内のレコードの置換』
- 146 ページの『ファイルからのレコードの削除』

関連参照

- 144 ページの『ファイルへのレコード書き込み時に使用するステートメント』

COBOL アプリケーションで SFS ファイルにアクセスできるようにする

AIX で実行される COBOL アプリケーションから Encina SFS ファイルにアクセスするには、コンパイルおよびリンクに関するガイドラインや、ファイル・システム、Encina SFS サーバー、および SFS ファイルを識別するためのガイドラインに従う必要があります。

1. `cob2_r` コマンドを使用して、アプリケーション内の COBOL プログラムをコンパイルおよびリンクします。
2. VSAM ファイル・システムを使用します。(FILESYS(VSA) は FILESYS ランタイム・オプションのデフォルトです。)
3. アプリケーションがアクセスする Encina SFS サーバーが稼働中であることを確認します。
4. `ENCINA_CDS_ROOT` 環境変数を、Encina SFS サーバーの完全修飾名に設定します。

以下に例を示します。

```
export ENCINA_CDS_ROOT=./cics/sfs/test1
```

5. (オプション) アプリケーションで 1 つ以上の SFS ファイルを作成し、`encinaSfsVol` 以外の名前を持つ SFS データ・ボリューム上にファイルを割り当てたい場合は、次の名前のいずれかか両方を指定することができます。
 - SFS ファイルが作成される SFS データ・ボリュームの名前。このためには、ランタイム環境変数 `ENCINA_SFS_DATA_VOLUME` に値を割り当てます。SFS サーバーに対してデータ・ボリュームが定義されている必要があります。SFS サーバーに対してどのデータ・ボリュームが使用可能かわからない場合は、コマンド `sfsadmin list lvols` を発行します。

デフォルトでは、SFS ファイルは `encinaSfsVol` という名前のデータ・ボリューム上に作成されます。
 - 代替索引ファイルが存在する場合は、それが作成される SFS データ・ボリュームの名前。このためには、ランタイム環境変数 `ENCINA_SFS_INDEX_VOLUME` に値を割り当てます。SFS サーバーに対してデータ・ボリュームが定義されている必要があります。

デフォルトでは、代替索引ファイルは対応するベース・ファイルと同じボリューム上に作成されます。
6. 完全修飾名を使用して各 SFS ファイルを識別します。
 - Encina SFS サーバー名はファイル名の前に付ける必要があります。
 - 代替索引ファイル名は、ベース・ファイル名で始まり、その後セミコロン (;) と代替索引名を付ける必要があります。

例えば、`./cics/sfs/test1` が Encina SFS サーバーで、VSAM04A が代替索引 VSAM04A1 を含む SFS ファイルの場合、VSAM04A を識別するには次の `export` コマンドを発行します。

```
export VSAM04A="./cics/sfs/test1/VSAM04A(/cics/sfs/test1/VSAM04A;VSAM04A1)"
```

SFS サーバーおよびファイルの完全修飾名については、Encina SFS ファイルの識別に関する関連タスクを参照してください。

『例: SFS ファイルへのアクセス』

関連タスク

122 ページの『Encina SFS ファイルの識別』
238 ページの『SFS ファイルのためのコンパイル』
151 ページの『SFS パフォーマンスの向上』

Encina SFS Programming Guide

関連参照

125 ページの『SFS ファイル・システム』
228 ページの『ランタイム環境変数』
328 ページの『FILESYS』

Encina Administration Guide Volume 2: Server Administration

例: SFS ファイルへのアクセス

下記の例は、コーディング可能な COBOL ファイルの記述を示しています。さらに、2 つの SFS ファイルを作成してそれらにアクセスするために発行できる `sfsadmin` コマンドと `export` コマンドも示しています。

VSAM04 は、代替索引を含まない索引付きファイルです。VSAM04A は、1 つの代替索引 VSAM04A1 を含む索引付きファイルです。

COBOL ファイル記述

```
...
input-output section.
file-control.
    select VSAM04-file
        assign to VSAM04
        access dynamic
        organization is indexed
        record key is VSAM04-rec-num
        file status is VSAM04-status.

    select VSAM04A-file
        assign to VSAM04A
        access dynamic
        organization is indexed
        record key is VSAM04A-rec-num
        alternate record key is VSAM04A-date-time
        file status is VSAM04A-status.

data division.
file section.
FD VSAM04-file
    data record is VSAM04-record.
01 VSAM04-record.
    05 VSAM04-rec-num                pic x(10).
```

```

05 VSAM04-rec-data          pic x(70).
FD VSAM04A-file
  data record is VSAM04A-record.
01 VSAM04A-record.
  05 VSAM04A-rec-num        pic x(10).
  05 VSAM04A-date-time.
    07 VSAM04A-date-yyyyymmdd pic 9(8).
    07 VSAM04A-time-hhmmsshh  pic 9(8).
    07 VSAM04A-date-time-counter pic 9(8).
  05 VSAM04A-rec-data      pic x(1000).

```

sfsadmin コマンド

sfsadmin create clusteredfile コマンドを発行してそれぞれの索引付きファイルを作成し、sfsadmin add index コマンドを発行して代替索引を追加します。

```

sfsadmin create clusteredfile VSAM04 2 ¥
PrimaryKey byteArray 10 ¥
DATA byteArray 70 ¥
primaryIndex -unique 1 PrimaryKey sfs_SFS_SERV
#
sfsadmin create clusteredfile VSAM04A 3 ¥
PrimaryKey byteArray 10 ¥
AltKey1 byteArray 24 ¥
DATA byteArray 1000 ¥
primaryIndex -unique 1 PrimaryKey sfs_SFS_SERV
#
sfsadmin add index VSAM04A VSAM04A1 1 AltKey1

```

上記の最初の sfsadmin create clusteredfile コマンドで示されているように、次の項目を指定する必要があります。

- 新規索引付きファイルの名前 (この例では VSAM04)
- レコード当たりのフィールド数 (2)
- 各フィールドの記述 (PrimaryKey および DATA で、それぞれがバイト配列)
- 1 次索引の名前 (primaryIndex)
- -unique オプション
- 1 次索引内のフィールド数 (1)
- 1 次索引内のフィールドの名前 (PrimaryKey)
- フィールドの保管先のデータ・ボリュームの名前 (sfs_SFS_SERV)

Encina SFS では、デフォルトでクラスター・ファイルの 1 次索引に重複キーを使用できます。ただし、COBOL では 1 次索引のキー値がファイル内で固有でなければならないため、上記のように -unique オプションを指定する必要があります。

上記の sfsadmin add index コマンドで示されているように、次の項目を指定する必要があります。

- 代替索引の追加先のファイルの名前 (この例では VSAM04A)
- 新規索引の名前 (VSAM04A1)
- 新規索引でキーとして使用するフィールドの数 (1)
- 新規索引内のフィールドの名前 (AltKey1)

コマンド sfsadmin create clusteredfile および sfsadmin add index の構文について詳しくは、以下の関連参照を参照してください。

export コマンド

SFS ファイルを処理するプログラムを実行する前に、以下の export コマンドを発行して、ファイルにアクセスする Encina SFS サーバー (././cics/sfs/test1) と、ファイルを保管するデータ・ボリューム (sfs_SFS_SERV) を指定します。

```
# Set environment variables required by the VSAM file system
# for SFS files:
export ENCINA_CDS_ROOT=././cics/sfs/test1
export ENCINA_SFS_DATA_VOLUME=sfs_SFS_SERV
export ENCINA_SFS_INDEX_VOLUME=sfs_SFS_SERV
# Set environment variable to access SFS file VSAM04:
export VSAM04=././cics/sfs/test1/VSAM04
# Set environment variable to access SFS file VSAM04A and the
# alternate index VSAM04A1:
export VSAM04A="././cics/sfs/test1/VSAM04A(././cics/sfs/test1/VSAM04A;VSAM04A1)"
```

関連参照

Encina Administration Guide Volume 2: Server Administration

SFS パフォーマンスの向上

SFS ファイルにアクセスするアプリケーションのパフォーマンスを向上させる方法が 2 つあります。1 つは、クライアント・マシンでクライアント側のキャッシュを使用するという方法です。もう 1 つは、SFS ファイルに変更内容を保存する頻度を減らすという方法です。

関連タスク

- 122 ページの『Encina SFS ファイルの識別』
- 『クライアント側のキャッシュの使用可能化』
- 153 ページの『変更の保存回数の削減』

Encina SFS Programming Guide

関連参照

- 125 ページの『SFS ファイル・システム』

クライアント側のキャッシュの使用可能化

デフォルトでは、SFS ファイル内のレコードは SFS サーバーとの間で読み書きされ、レコードにアクセスするたびにリモート・プロシージャ・コール (RPC) が必要です。ただし、クライアント側のキャッシュを有効にすると、レコードへのアクセスに必要な時間が短くなるため、パフォーマンスを向上させることができます。

クライアント側のキャッシュを使用している場合、レコードはクライアント・マシンのローカル・メモリー (キャッシュ) に保管され、1 回の RPC でサーバーに送信 (フラッシュ) されます。読み取りキャッシュと挿入キャッシュという 2 種類のキャッシュのいずれか、または両方とも指定することができます。

- 読み取りキャッシュを使用可能にすると、ファイルの最初の順次読み取りにより、現行レコードおよび隣接レコードの数がサーバーから読み取られ、読み取りキャッシュ内に入れられます。以降の順次読み取り、更新、および削除は、サーバー上ではなく読み取りキャッシュ内で行われます。
- 挿入キャッシュを使用可能にすると、挿入操作 (読み取り、更新、または削除ではなく) は、サーバー上ではなく挿入キャッシュ内で行われます。

アプリケーション内のすべての SFS ファイルに対してクライアント側のキャッシュを使用可能にするには、アプリケーションを実行する前に ENCINA_VSAM_CACHE 環境変数を設定します。ENCINA_VSAM_CACHE 環境変数を設定する場合の構文図については、ランタイム環境変数に関する以下の関連参照を参照してください。

キャッシュ・サイズとして単一の値をコーディングして、読み取りキャッシュと挿入キャッシュに同じページ数を使用するよう指定したり、値をコロン (:) で区切って読み取りおよび挿入キャッシュに対して異なる値をコーディングすることができます。サイズの単位はページ数として表現します。読み取りキャッシュ、挿入キャッシュ、またはその両方のサイズとしてゼロをコーディングすると、そのタイプのキャッシュは使用不可になります。例えば、次のコマンドは、アプリケーション内の各 SFS ファイルの読み取りキャッシュのサイズを 16 ページに設定し、挿入キャッシュのサイズを 64 ページに設定します。

```
export ENCINA_VSAM_CACHE=16:64
```

また、クライアント側のキャッシュをより柔軟なものにするために、次のフラグのいずれかまたは両方を指定することもできます。

- 非コミット・データ (新規または変更済みだが、サーバーに送信されなかったレコードで、ダーティー・レコードとも言う) の読み取りを許可するには、ALLOW_DIRTY_READS を指定します。

このフラグは、アクセスするファイルがロックされている必要があるという読み取りキャッシュの制限を取り除きます。

- 任意の挿入のキャッシュを許可するには、INSERTS_DESPITE_UNIQUE_INDICES を指定します。

このフラグは、クラスター・ファイルのすべてのアクティブな索引と、入力順および相対ファイルのアクティブな代替索引が、重複を許可している必要があるという、挿入キャッシュの制限を取り除きます。

例えば、次のコマンドは最大の柔軟性を可能にします。

```
export ENCINA_VSAM_CACHE=16:64:ALLOW_DIRTY_READS,INSERTS_DESPITE_UNIQUE_INDICES
```

特定のファイルのそれぞれにクライアント側のキャッシュを設定するには、キャッシュ方針を変更したい各ファイルの OPEN ステートメントの前に、ENCINA_VSAM_CACHE を設定する putenv() 呼び出しをコーディングします。プログラムの中では、putenv() 呼び出しで行う環境変数の設定は、export コマンドで行う環境変数の設定に優先します。

234 ページの『例: 環境変数の設定とアクセス』

関連タスク

225 ページの『環境変数の設定』

関連参照

125 ページの『SFS ファイル・システム』

228 ページの『ランタイム環境変数』

変更の保存回数の削減

通常、RPC は、クライアント側のキャッシュを使用しない (つまり、SFS の強制操作機能が使用可能になっている) SFS ファイルに対して書き込み操作または更新操作を行うたびに実行されます。入出力操作の結果として発生するファイルの変更はすべて、制御がアプリケーションに返される前にディスクにコミットされます。

この動作は、ファイルが閉じられるまで SFS ファイル上の入出力操作の結果がディスクにコミットされないように変更することができます (つまり、遅延書き込み方針を使用します)。SFS ファイルに対する変更が、毎回即時に保存されるわけではなく、アプリケーションを高速で実行することができます。

アプリケーション内のすべての SFS ファイルに対するデフォルトのコミット動作を変更するには、アプリケーションを実行する前に、ENCINA_VSAM_AUTO_FLUSH 環境変数を OFF に設定します。

```
export ENCINA_VSAM_AUTO_FLUSH=OFF
```

特定のファイルのそれぞれにフラッシュ値を設定するには、フラッシュ値を変更したい各ファイルの OPEN ステートメントの前に、ENCINA_VSAM_FLUSH を設定する putenv() 呼び出しをコーディングします。プログラムの中では、putenv() 呼び出しで行う環境変数の設定は、export コマンドで行う環境変数の設定に優先します。

234 ページの『例: 環境変数の設定とアクセス』

SFS ファイルに対してクライアント側のキャッシュが有効な場合 (つまり、環境変数 ENCINA_VSAM_CACHE がゼロ以外の有効な値に設定されている場合)、そのファイルの ENCINA_VSAM_AUTO_FLUSH は無視されます。そのファイルに対して強制操作は使用できません。

関連タスク

225 ページの『環境変数の設定』

関連参照

125 ページの『SFS ファイル・システム』

228 ページの『ランタイム環境変数』

第 8 章 ファイルのソートおよびマージ

SORT または MERGE ステートメントを使用すると、レコードを特定のシーケンスで並べることができます。同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。

SORT ステートメント

(ファイルまたは内部プロシージャから) 順序付けられていない入力を受け入れ、要求されたシーケンスで出力を (ファイルまたは内部プロシージャに) 作成します。ソートの前に、レコードを追加、削除、または変更することができます。

MERGE ステートメント

2 つ以上の順序付けられたファイルからのレコードを比較し、それらを順序正しく結合します。マージの前に、レコードを追加、削除、または変更することができます。

プログラムにいくつかのソート操作およびマージ操作を含めても構いません。また、同じ操作を何度も実行しても構いませんし、異なる操作を実行しても構いません。ただし、1 つの操作が終了してからでなければ、別の操作を開始することはできません。

ソートまたはマージで行う手順は一般的に次のようになります。

1. ソートまたはマージに使用するソート・ファイルまたはマージ・ファイルを記述する。
2. ソートまたはマージする入力を記述する。レコードをソート前に処理したい場合には、入力プロシージャをコーディングしてください。
3. ソートまたはマージからの出力を記述する。レコードをソートまたはマージした後処理したい場合には、出力プロシージャをコーディングしてください。
4. ソートまたはマージを要求する。
5. ソートまたはマージ操作が成功したかどうかを判別する。

関連概念

156 ページの『ソートおよびマージ・プロセス』

関連タスク

156 ページの『ソートまたはマージ・ファイルの記述』

157 ページの『ソートまたはマージへの入力の記述』

159 ページの『ソートまたはマージからの出力の記述』

161 ページの『ソートまたはマージの要求』

164 ページの『ソートまたはマージの成否の判断』

168 ページの『ソートまたはマージ操作の途中停止』

関連参照

SORT ステートメント (COBOL for AIX 言語解説書)

MERGE ステートメント (COBOL for AIX 言語解説書)

ソートおよびマージ・プロセス

ファイルのソート時に、ファイル内のレコードはすべて、それぞれのレコード内の 1 つ以上のフィールドの内容 (キー) に従って順序付けられます。レコードは、各キーの昇順または降順にソートすることができます。

複数のキーがある場合は、レコードはまず最初の (基本) キーの内容に従ってソートされ、次に 2 番目のキーの内容に従ってソートされる、というようになります。

ファイルをソートするには、COBOL の SORT ステートメントを使用します。

複数のファイルのマージ時には (これらのファイルはソート済みでなければなりません)、レコードは、各レコード内の 1 つ以上のキーの内容に従って結合され、順序付けされます。レコードは、各キーの昇順または降順に順序付けすることができます。ソートの場合と同様、レコードはまず最初の (基本) キーの内容に従って順序付けされ、次に 2 番目のキーの内容に従って順序付けされる、というようになります。

MERGE . . . USING を使用して、順序付けられた 1 つのファイルとして結合したい複数のファイルの名前を指定します。マージ操作では、入力ファイルのレコード内のキーを比較し、順序付けられたレコードを 1 つずつ、出力プロシージャの RETURN ステートメントに、または GIVING 句で指定されたファイルに渡します。

関連タスク

162 ページの『ソートまたはマージ基準の設定』

関連参照

SORT ステートメント (COBOL for AIX 言語解説書)

MERGE ステートメント (COBOL for AIX 言語解説書)

ソートまたはマージ・ファイルの記述

ソートまたはマージに使用するソート・ファイルを記述してください。

WORKING-STORAGE または LOCAL-STORAGE からのデータ項目のみをソートまたはマージする場合でも、SELECT 節および SD 項目が必要です。

次のようにコーディングします。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つ以上の SELECT 節をコーディングし、ソート・ファイルの名前を指定します。以下に例を示します。

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT Sort-Work-1 ASSIGN TO SortFile.
```

Sort-Work-1 は、プログラム内のファイルの名前です。この名前を使用してファイルを参照してください。

2. そのソート・ファイルを、DATA DIVISION の FILE SECTION の SD 記入項目で記述します。それぞれの SD 記入項目がレコード記述を含んでいなければなりません。以下に例を示します。

```

DATA DIVISION.
FILE SECTION.
SD  Sort-Work-1
    RECORD CONTAINS 100 CHARACTERS.
01  SORT-WORK-1-AREA.
    05  SORT-KEY-1    PIC  X(10).
    05  SORT-KEY-2    PIC  X(10).
    05  FILLER        PIC  X(80).

```

SD 記入項目で記述するファイルは、ソートまたはマージ操作に使用される作業ファイルです。このファイルに入力または出力操作を実行することはできません。

関連参照

13 ページの『FILE SECTION 記入項目』

ソートまたはマージへの入力の記述

ソートまたはマージ用の入力ファイルは、以下の手順に従って記述してください。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つ以上の SELECT 節をコーディングし、入力ファイルの名前を指定します。以下に例を示します。

```

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT Input-File ASSIGN TO InFile.

```

Input-File は、プログラム内のファイルの名前です。この名前を使用してファイルを参照してください。

2. その入力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。以下に例を示します。

```

DATA DIVISION.
FILE SECTION.
FD  Input-File
    RECORD CONTAINS 100 CHARACTERS.
01  Input-Record    PIC  X(100).

```

関連タスク

158 ページの『入力プロシージャのコーディング』

161 ページの『ソートまたはマージの要求』

関連参照

13 ページの『FILE SECTION 記入項目』

例: SORT 用のソート・ファイルおよび入力ファイルの記述

次の例は、ソート作業ファイルおよび入力ファイルを記述するのに必要な ENVIRONMENT DIVISION および DATA DIVISION の記入項目を示しています。

```

ID Division.
Program-ID.  Smp1Sort.
Environment Division.
Input-Output Section.
File-Control.
*
* Assign name for a working file is treated as documentation.
*
    Select Sort-Work-1 Assign To SortFile.

```

```

        Select Sort-Work-2 Assign To SortFile.
        Select Input-File Assign To InFile.
    . . .
Data Division.
File Section.
SD Sort-Work-1
   Record Contains 100 Characters.
01 Sort-Work-1-Area.
   05 Sort-Key-1    Pic X(10).
   05 Sort-Key-2    Pic X(10).
   05 Filler        Pic X(80).
SD Sort-Work-2
   Record Contains 30 Characters.
01 Sort-Work-2-Area.
   05 Sort-Key      Pic X(5).
   05 Filler        Pic X(25).
FD Input-File
   Record Contains 100 Characters.
01 Input-Record    Pic X(100).
. . .
Working-Storage Section.
01 EOS-Sw          Pic X.
01 Filler.
   05 Table-Entry Occurs 100 Times
       Indexed By X1    Pic X(30).
. . .

```

関連タスク

161 ページの『ソートまたはマージの要求』

入力プロシージャーのコーディング

入力ファイルのレコードを、それらがソート・プログラムに解放される前に処理する場合には、SORT ステートメントの INPUT PROCEDURE 句を使用してください。

入力プロシージャーを使用して、以下のことを行うことができます。

- データ項目を WORKING-STORAGE または LOCAL-STORAGE からソート・ファイルに解放する。
- プログラム内の別な場所で既に読み取られているレコードを解放する。
- 入力レコードからレコードを読み取り、それらを選択または処理し、それらをソート・ファイルに解放する

それぞれの入力プロシージャーは、段落またはセクションのいずれかで構成されなければなりません。例えば、WORKING-STORAGE または LOCAL-STORAGE の表からのレコードをソート・ファイル SORT-WORK-2 に解放するには、次のようにコーディングすることができます。

```

        SORT SORT-WORK-2
          ON ASCENDING KEY SORT-KEY
          INPUT PROCEDURE 600-SORT3-INPUT-PROC
    . . .
600-SORT3-INPUT-PROC SECTION.
  PERFORM WITH TEST AFTER
    VARYING X1 FROM 1 BY 1 UNTIL X1 = 100
    RELEASE SORT-WORK-2-AREA FROM TABLE-ENTRY (X1)
  END-PERFORM.

```

レコードをソート・プログラムに転送するためには、すべての入力プロシージャに少なくとも 1 つの RELEASE または RELEASE FROM ステートメントが含まれていなければなりません。例えば、X から A を解放するには、次のようにコーディングできます。

```
MOVE X TO A.
RELEASE A.
```

あるいは、次のようにコーディングできます。

```
RELEASE A FROM X.
```

次の表では、RELEASE ステートメントと RELEASE FROM ステートメントを比較しています。

RELEASE	RELEASE FROM
<pre>MOVE EXT-RECORD TO SORT-EXT-RECORD PERFORM RELEASE-SORT-RECORD . . . RELEASE-SORT-RECORD. RELEASE SORT-RECORD</pre>	<pre>PERFORM RELEASE-SORT-RECORD . . . RELEASE-SORT-RECORD. RELEASE SORT-RECORD FROM SORT-EXT-RECORD</pre>

関連参照

160 ページの『入出力プロシージャに関する制約事項』
RELEASE ステートメント (COBOL for AIX 言語解説書)

ソートまたはマージからの出力の記述

ソートまたはマージからの出力がファイルである場合は、以下の手順に従ってファイルを記述しなければなりません。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に SELECT 節をコーディングし、出力ファイルの名前を指定します。以下に例を示します。

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT Output-File ASSIGN TO OutFile.
```

Output-File は、プログラム内のファイルの名前です。この名前を使用してファイルを参照してください。

2. その出力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。以下に例を示します。

```
DATA DIVISION.
FILE SECTION.
FD Output-File
  RECORD CONTAINS 100 CHARACTERS.
01 Output-Record PIC X(100).
```

関連タスク

160 ページの『出力プロシージャのコーディング』
161 ページの『ソートまたはマージの要求』

関連参照

13 ページの『FILE SECTION 記入項目』

出力プロシージャーのコーディング

ソート済みのレコードをソート作業ファイルから別のファイルに書きこむ前に、それらの選択、編集、またはそれ以外の変更を行うには、SORT ステートメントの OUTPUT PROCEDURE 句を使用してください。

それぞれの出力プロシージャーは、セクションまたは段落のいずれかで構成されなければなりません。また、出力プロシージャーには、以下の両方を含めなければなりません。

- 少なくとも 1 つ以上の RETURN ステートメントまたは INTO 句を含んだ 1 つの RETURN ステートメント
- レコードの処理に必要なステートメント。レコードは、RETURN ステートメントによって一度に 1 つずつ使用可能になります。

RETURN ステートメントによって、ソート済みの各レコードが出力プロシージャーから使用可能になります。(ソート・ファイルに対する RETURN ステートメントは、入力ファイルに対する READ ステートメントに似ています。)

RETURN ステートメントとともに AT END および END-RETURN 句を使用することができます。AT END 句の命令ステートメントは、ソート・ファイルからすべてのレコードが戻された後で実行されます。END-RETURN 明示範囲終了符号は、RETURN ステートメントの有効範囲を区切る役割をします。

RETURN ではなく RETURN INTO を使用すると、レコードは WORKING-STORAGE、LOCAL-STORAGE、または出力域に戻されます。

関連参照

『入出力プロシージャーに関する制約事項』

RETURN ステートメント (COBOL for AIX 言語解説書)

入出力プロシージャーに関する制約事項

SORT によって呼び出されるそれぞれの入出力プロシージャー、および MERGE によって呼び出されるそれぞれの出力プロシージャーには、以下の制約事項が適用されます。

- プロシージャーに SORT または MERGE ステートメントを含めてはなりません。
- プロシージャーの中で ALTER、GO TO、および PERFORM ステートメントを使用することによって、入力または出力プロシージャーの外側にあるプロシージャー名を参照することができます。しかし、GO TO または PERFORM ステートメントの後で、その入力または出力プロシージャーに制御権を戻さなければなりません。
- PROCEDURE DIVISION のその他の部分に、入力または出力プロシージャーの内部への制御権の移動を記述してはなりません (ただし、宣言セクションからの制御権の戻りは例外です)。

- 入力または出力プロシージャの中から、プログラムを呼び出すことができます。ただし、呼び出されるプログラムから SORT または MERGE ステートメントを出すことはできません。呼び出されるプログラムは呼び出し元に戻る必要があります。
- SORT または MERGE 操作時には、SD データ項目が使用されます。出力プロシージャの中で、最初の RETURN が実行される前に、このデータ項目を使用してはなりません。最初の RETURN ステートメントの前に、データをこのレコード域に移動すると、戻される最初のレコードが上書きされます。

関連タスク

158 ページの『入力プロシージャのコーディング』

160 ページの『出力プロシージャのコーディング』

ソートまたはマージの要求

事前処理を行わずに 1 つの入力ファイル (MERGE の場合は複数のファイル) からレコードを読み取るには、SORT . . . USING または MERGE . . . USING を使用し、さらに SELECT 文節で宣言した入力ファイルの名前も使用します。

ソート済みまたはマージ済みレコードを、これ以上処理せずに、ソート・プログラムまたはマージ・プログラムから別のファイルへ転送するには、SORT . . . GIVING または MERGE . . . GIVING、および SELECT 節で宣言された出力ファイルの名前を使用してください。以下に例を示します。

```
SORT Sort-Work-1
  ON ASCENDING KEY Sort-Key-1
  USING Input-File
  GIVING Output-File.
```

SORT . . . USING または MERGE . . . USING の場合、コンパイラーは入力プロシージャを生成します。このプロシージャは、ファイルをオープンし、レコードを読み取り、レコードをソート・プログラムまたはマージ・プログラムに渡し、ファイルをクローズします。SORT または MERGE ステートメントが実行を開始するとき、ファイルが開いた状態にしないでください。SORT . . . GIVING または MERGE . . . GIVING の場合、コンパイラーは出力プロシージャを生成します。このプロシージャは、ファイルをオープンし、レコードを返し、レコードを書き込み、ファイルをクローズします。SORT または MERGE ステートメントが実行を開始するとき、ファイルが開いた状態にしないでください。

157 ページの『例: SORT 用のソート・ファイルおよび入力ファイルの記述』

ソート・レコードがソートされる前にソート・レコードに対して入力プロシージャが実行されるようにする場合は、SORT . . . INPUT PROCEDURE を使用します。ソート済みレコードに対して出力プロシージャが実行されるようにする場合は、SORT . . . OUTPUT PROCEDURE を使用します。以下に例を示します。

```
SORT Sort-Work-1
  ON ASCENDING KEY Sort-Key-1
  INPUT PROCEDURE EditInputRecords
  OUTPUT PROCEDURE FormatData.
```

163 ページの『例: 入出力プロシージャを使用したソート』

制約事項: MERGE ステートメントで入力プロシージャーを使用することはできません。マージ操作への入力ソースは、既にソート済みのファイルの集合でなければなりません。ただし、マージ済みレコードに対して出力プロシージャーが実行されるようにする場合は、MERGE . . . OUTPUT PROCEDURE を使用します。以下に例を示します。

```
MERGE Merge-Work
      ON ASCENDING KEY Merge-Key
      USING Input-File-1 Input-File-2 Input-File-3
      OUTPUT PROCEDURE ProcessOutput.
```

FILE SECTION では、SD 記入項目の *Merge-Work*、および FD 記入項目の入力ファイルを定義する必要があります。

関連参照

SORT ステートメント (*COBOL for AIX 言語解説書*)

MERGE ステートメント (*COBOL for AIX 言語解説書*)

ソートまたはマージ基準の設定

ソートまたはマージ基準を設定するには、操作の実行対象のキーを定義します。

以下の手順を実行します。

1. ソートまたはマージするファイルのレコード記述の中で、キー (複数の場合もある) を定義します。

制約事項: キーは可変位置にすることはできません。

2. SORT または MERGE ステートメントの中で、ASCENDING 句または DESCENDING KEY 句 (あるいはその両方) をコーディングすることにより、順序付けに使用するキー・フィールドを指定してください。複数のキーをコーディングする場合、一部を昇順にし、残りを降順にすることができます。

キーの名前は重要度の高い順に指定します。左端のキーが基本キーです。次のキーが 2 次キー、というようになります。

SORT および MERGE キーは、クラス英字、英数字、国別 (コンパイラー・オプション NCOLLSEQ(BIN) が有効化されている場合)、数値 (ただし、USAGE NATIONAL の数値ではない) にすることができます。USAGE NATIONAL を持っている場合、キーはカテゴリー国別にするか、あるいは国別編集または数字編集データ項目にすることができます。キーを、国別 10 進数データ項目にしたり、国別浮動小数点データ項目にすることはできません。

国別キーの照合順序は、キーのバイナリーの順序によって決まります。キーとして国別データ項目を指定する場合は、SORT または MERGE ステートメントの COLLATING SEQUENCE 句はいずれもそのキーに適用されません。

同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。プログラムはいくつのソート操作およびマージ操作でも実行することができます。ただし、1 つの操作が終了してからでなければ、次の操作を開始することはできません。

関連タスク

216 ページの『ロケール付きの照合シーケンスの制御』

関連参照

296 ページの『NCOLLSEQ』

SORT ステートメント (*COBOL for AIX 言語解説書*)

MERGE ステートメント (*COBOL for AIX 言語解説書*)

代替照合シーケンスの選択

レコードのソートまたはマージは、1 バイト文字キーで指定した照合シーケンスで行うことができます。OBJECT-COMPUTER 段落で PROGRAM COLLATING SEQUENCE 節をコーディングしない限り、デフォルトの照合シーケンスは、コンパイル時に有効化されているロケール・設定で指定された照合シーケンスです。

デフォルト・シーケンスをオーバーライドするには、SORT または MERGE ステートメントの COLLATING SEQUENCE 句を使用してください。プログラムの SORT または MERGE ステートメントごとに異なる照合シーケンスを使用することができます。

PROGRAM COLLATING SEQUENCE 節および COLLATING SEQUENCE 句は、クラス英字または英数字のキーにのみ適用されます。COLLATING SEQUENCE 句は、1 バイト ASCII コード・ページが有効化されている場合にのみ有効です。

関連タスク

8 ページの『照合シーケンスの指定』

216 ページの『ロケール付きの照合シーケンスの制御』

162 ページの『ソートまたはマージ基準の設定』

関連参照

OBJECT-COMPUTER 段落 (*COBOL for AIX 言語解説書*)

SORT ステートメント (*COBOL for AIX 言語解説書*)

データのクラスおよびカテゴリー (*COBOL for AIX 言語解説書*)

例: 入出力プロシージャを使用したソート

以下は、SORT ステートメントにおける入出力プロシージャの使用例です。この例では、基本キー (SORT-GRID-LOCATION) と 2 次キー (SORT-SHIFT) を SORT ステートメントで使用する前に、それらのキーをどのように定義できるかも示します。

```
DATA DIVISION.  
...  
SD SORT-FILE  
  RECORD CONTAINS 115 CHARACTERS  
  DATA RECORD SORT-RECORD.  
01 SORT-RECORD.  
  05 SORT-KEY.  
    10 SORT-SHIFT          PIC X(1).  
    10 SORT-GRID-LOCATION   PIC X(2).  
    10 SORT-REPORT         PIC X(3).  
  05 SORT-EXT-RECORD.  
    10 SORT-EXT-EMPLOYEE-NUM PIC X(6).  
    10 SORT-EXT-NAME       PIC X(30).  
    10 FILLER              PIC X(73).  
...  
WORKING-STORAGE SECTION.
```

```

01 TAB1.
   05 TAB-ENTRY OCCURS 10 TIMES
      INDEXED BY TAB-INDX.
      10 WS-SHIFT                PIC X(1).
      10 WS-GRID-LOCATION         PIC X(2).
      10 WS-REPORT              PIC X(3).
      10 WS-EXT-EMPLOYEE-NUM    PIC X(6).
      10 WS-EXT-NAME            PIC X(30).
      10 FILLER                  PIC X(73).
. . .
PROCEDURE DIVISION.
. . .
   SORT SORT-FILE
      ON ASCENDING KEY SORT-GRID-LOCATION SORT-SHIFT
      INPUT PROCEDURE 600-SORT3-INPUT
      OUTPUT PROCEDURE 700-SORT3-OUTPUT.
. . .
600-SORT3-INPUT.
   PERFORM VARYING TAB-INDX FROM 1 BY 1 UNTIL TAB-INDX > 10
   RELEASE SORT-RECORD FROM TAB-ENTRY(TAB-INDX)
   END-PERFORM.
. . .
700-SORT3-OUTPUT.
   PERFORM VARYING TAB-INDX FROM 1 BY 1 UNTIL TAB-INDX > 10
   RETURN SORT-FILE INTO TAB-ENTRY(TAB-INDX)
   AT END DISPLAY 'Out Of Records In SORT File'
   END-RETURN
   END-PERFORM.

```

関連タスク

161 ページの『ソートまたはマージの要求』

ソートまたはマージの成否の判断

SORT または MERGE ステートメントは、各ソートまたはマージ操作の終了後に、0 (正常終了) または 16 (異常終了) のいずれかの完了コードを返します。完了コードは、SORT-RETURN 特殊レジスターに保管されます。

それぞれの SORT または MERGE ステートメントの後で、正常終了かどうかをテストしなければなりません。以下に例を示します。

```

   SORT SORT-WORK-2
      ON ASCENDING KEY SORT-KEY
      INPUT PROCEDURE IS 600-SORT3-INPUT-PROC
      OUTPUT PROCEDURE IS 700-SORT3-OUTPUT-PROC.
   IF SORT-RETURN NOT=0
      DISPLAY "SORT ENDED ABNORMALLY. SORT-RETURN = " SORT-RETURN.
. . .
600-SORT3-INPUT-PROC SECTION.
. . .
700-SORT3-OUTPUT-PROC SECTION.
. . .

```

プログラムの中で SORT-RETURN をまったく参照しないと、COBOL ランタイムで完了コードがテストされます。16 の場合、COBOL は、ランタイム診断メッセージを出し、実行単位 (マルチスレッド化環境では、スレッド) を終了します。診断メッセージには、ソートまたはマージのエラー番号が入っています。これは、問題の原因の決定に役立たせることができます。

SORT または MERGE ステートメントの 1 つ以上 (しかし、必ずしも全部ではない) について SORT-RETURN をテストすると、COBOL ランタイムで完了コードが検査されません。ただし、SORT または MERGE ステートメントの後に、iwzGetSortErrno サービスを呼び出すことによって、例えば以下のように、ソートまたはマージのエラー番号を取得することができます。

```
77 sortErrno    PIC 9(9)    COMP-5.
...
CALL 'iwzGetSortErrno' USING sortErrno
...
```

エラー番号とその意味のリストについては、以下に示す関連参照を参照してください。

関連参照

『ソートおよびマージ・エラー番号』

ソートおよびマージ・エラー番号

プログラム内で SORT-RETURN を参照しない場合で、ソースまたはマージ操作からの完了コードが 16 である場合、COBOL for AIX はランタイム診断メッセージを発行します。これには、以下の表に示すゼロ以外のエラー番号のいずれかが含まれています。

表 18. ソートおよびマージ・エラー番号

エラー番号	説明
0	エラーなし
1	レコードの順序が間違っています
2	同等鍵付きレコードが検出されました
3	複数の main 関数が指定されました (内部エラー)
4	パラメーター・ファイルにエラーがあります
5	パラメーター・ファイルをオープンできませんでした
6	オペランドがオプションから欠落しています
7	オペランドが拡張オプションから欠落しています
8	オプション内のオペランドが無効です
9	拡張オプション内のオペランドが無効です
10	無効なオプションが指定されました
11	無効な拡張オプションが指定されました
12	無効な一時ディレクトリーが指定されました
13	無効なファイル名が指定されました
14	無効なフィールドが指定されました
15	フィールドがレコード内にありません
16	フィールドがレコード内で短すぎます
17	SELECT の指定に構文エラーがあります
18	無効な定数が SELECT で指定されました
19	SELECT 内の定数とデータ型の間の比較が無効です
20	SELECT 内の 2 つのデータ型の間の比較が無効です

表 18. ソートおよびマージ・エラー番号 (続き)

エラー番号	説明
21	形式の指定に構文エラーがあります
22	再フォーマットの指定に構文エラーがあります
23	無効な定数が再フォーマットの指定で指定されました
24	合計の指定に構文エラーがあります
25	フラグが複数回指定されました
26	指定された出力が多すぎます
27	入力ソースが指定されませんでした
28	出力宛先が指定されませんでした
29	無効な修飾子が指定されました
30	合計は許可されません
31	レコードが短すぎます
32	レコードが長すぎます
33	無効なバックまたはゾーン・フィールドが検出されました
34	ファイルに読み取りエラーがあります
35	ファイルに書き込みエラーがあります
36	入力ファイルをオープンできません
37	メッセージ・ファイルをオープンできません
38	VSAM ファイル・エラー
39	ターゲット・バッファのスペースが不十分です
40	一時ディスク・スペースが足りません
41	出力ファイルのスペースが足りません
42	予期しないシグナルがトラップされました
43	エラーが入力出口から戻されました
44	エラーが出力出口から戻されました
45	予期しないデータが出力ユーザー出口から戻されました
46	無効なバイトを使用した値が入力出口から戻されました
47	無効なバイトを使用した値が出力出口から戻されました
48	SMARTsort がアクティブではありません
49	実行を継続するためのストレージが不十分です
50	パラメーター・ファイルが大きすぎます
51	単一引用符が一致しません
52	引用符が一致しません
53	競合オプションが指定されました
54	レコード内の長さフィールドが無効です
55	レコード内の最終フィールドが無効です
56	必要なレコード形式が指定されませんでした
57	出力ファイルをオープンできません
58	一時ファイルをオープンできません
59	ファイル編成が無効です
60	指定されたファイル編成のユーザー出口がサポートされていません

表 18. ソートおよびマージ・エラー番号 (続き)

エラー番号	説明
61	ロケールがシステムに認識されていません
62	レコードに無効なマルチバイト文字があります
63	VSAM 編成がファイルに指定されましたが、ファイルが VSAM ではありません
64	SORT に指定されたキーが、索引付き出力ファイルの定義に使用できません
65	ファイルの VSAM 固定レコード長が、指定されたレコード形式と合致しません
66	SMARTsort オプションのファイル作成が失敗しました
67	完全修飾された、非相対パス名を作業ディレクトリーとして指定する必要があります
68	必須オプションを指定する必要があります
69	パス名が無効です
79	一時ファイルの最大数に達しました
501	関数が無効です
502	レコード・タイプが無効です
503	レコード長が無効です
504	タイプ長エラー
505	タイプが無効です
506	キー数が一致しません
507	タイプが長すぎます
508	キー・オフセットが無効です
509	昇順または降順キーが無効です
510	オーバーラップ・キーが無効です
511	キーが定義されませんでした。
512	入力ファイルが指定されませんでした
513	出力ファイルが指定されませんでした
514	入力ファイルのタイプが混在しています
515	出力ファイルのタイプが混在しています
516	入力作業バッファが無効です
517	出力作業バッファが無効です
518	COBOL 入力の入出力エラー
519	COBOL 出力の入出力エラー
520	関数がサポートされていません
521	無効なキー
522	無効な USING ファイル
523	無効な GIVING ファイル
524	作業ディレクトリーが提供されませんでした
525	作業ディレクトリーが存在しません
526	ソート共通が割り振られませんでした
527	ソート共通用のストレージがありません

表 18. ソートおよびマージ・エラー番号 (続き)

エラー番号	説明
528	バイナリー・バッファが割り振られませんでした
529	行順次ファイル・バッファが割り振られませんでした
530	ワークスペースの割り振りが失敗しました
531	FCB の割り振りが失敗しました

ソートまたはマージ操作の途中停止

ソートまたはマージ操作を停止するには、整数 16 を SORT-RETURN 特殊レジスターに移動させます。

次のいずれかの方法で、レジスターに 16 を移動してください。

- 入力または出力プロシージャの中で MOVE を使用する。

ソートまたはマージ処理は、次の RELEASE または RETURN ステートメントが実行された直後に停止されます。

- USING または GIVING ファイルの処理中に入る宣言セクションの中でこのレジスターをリセットする。

ソートまたはマージ処理は、宣言セクションから出たときに停止されます。

制御は、その後、SORT または MERGE ステートメントの次のステートメントに戻ります。

第 9 章 エラーの処理

起こり得るシステムまたは実行時の問題を予測するコードをプログラムに入れてください。そのようなコードを含めない場合、出力データまたはファイルが破損しても、ユーザーは問題が発生していることに気付くことさえない可能性があります。

エラー処理コードでは、状態の処理、メッセージの発行、プログラムの停止などのアクションを取ることができます。例えば、データ入力エラーまたはご使用のシステムで定義されたエラーに対して、独自のエラー検出ルーチンを作成することができます。どのようなイベントであっても、警告メッセージをコーディングするのはよいことです。

COBOL for AIX には、エラー状態を予測して訂正するのに役に立つ特殊なエレメントがいくつか含まれています。

- STRING および UNSTRING 操作の ON OVERFLOW
- 算術演算の ON SIZE ERROR
- 入出力エラーを処理するためのエレメント
- CALL ステートメントの ON EXCEPTION または ON OVERFLOW

関連タスク

『ストリングの結合および分割におけるエラーの処理』

170 ページの『算術演算でのエラーの処理』

171 ページの『入出力操作でのエラーの処理』

178 ページの『プログラム呼び出し時のエラーの処理』

ストリングの結合および分割におけるエラーの処理

ストリングの結合または分割中に、STRING または UNSTRING に使用されるポインターは、受信フィールドの範囲外になる可能性があります。オーバーフロー条件が存在する可能性はありますが、COBOL ではオーバーフローの発生を許可しません。

その代わりに、STRING 操作や UNSTRING 操作は完了せず、受信フィールドは未変更のままとなり、制御は次の順次ステートメントに移動します。STRING または UNSTRING ステートメントの ON OVERFLOW 句をコーディングしないと、操作未完了の通知が出されません。

次のステートメントを考えてください。

```
String Item-1 space Item-2 delimited by Item-3
      into Item-4
      with pointer String-ptr
      on overflow
          Display "A string overflow occurred"
End-String
```

以下に、ステートメントの実行前と実行後のデータ値を示します。

データ項目	PICTURE	実行前の値	実行後の値
Item-1	X(5)	AAAAA	AAAAA
Item-2	X(5)	EEEEAA	EEEEAA
Item-3	X(2)	EA	EA
Item-4	X(8)	bbbbbbb ¹	bbbbbbb ¹
String-ptr	9(2)	0	0

1. 記号 *b* は、ブランク・スペースを表します。

String-ptr の値は (0) で、受信フィールドには達しないため、オーバーフロー条件が発生し、STRING 操作は完了しません (String-ptr が 9 より大きい場合にも、オーバーフローが起こります)。ON OVERFLOW が指定されていなかった場合は、Item-4 の内容が未変更のままであったことについて通知されません。

算術演算でのエラーの処理

算術演算の結果が、それらを入れる固定小数点フィールドより大きかったり、0 除算が試みられたりすることがあります。いずれの場合も、ADD、SUBTRACT、MULTIPLY、DIVIDE、または COMPUTE ステートメントの後の ON SIZE ERROR 節でその状況を処理することができます。

ON SIZE ERROR が固定小数点オーバーフローおよび 10 進オーバーフローで正常に動作するようにするためには、TRAP(ON) ランタイム・オプションを指定する必要があります。

以下の場合、ON SIZE ERROR 節の命令ステートメントが実行され、結果フィールドは変更されません。

- 固定小数点オーバーフロー
- 0 による除算
- 0 の 0 乗
- 0 の負数乗
- 負数の分数乗

例: 0 による除算の検査

次の例は、ゼロ除算が発生した場合にプログラムが通知メッセージを出すように ON SIZE ERROR 命令ステートメントをコーディングする方法を示しています。

```

DIVIDE-TOTAL-COST.
  DIVIDE TOTAL-COST BY NUMBER-PURCHASED
    GIVING ANSWER
    ON SIZE ERROR
      DISPLAY "ERROR IN DIVIDE-TOTAL-COST PARAGRAPH"
      DISPLAY "SPENT " TOTAL-COST, " FOR " NUMBER-PURCHASED
      PERFORM FINISH
  END-DIVIDE
  . . .
  FINISH.
  STOP RUN.

```

ゼロ除算が発生すると、プログラムはメッセージを作成し、プログラム実行を停止します。

入出力操作でのエラーの処理

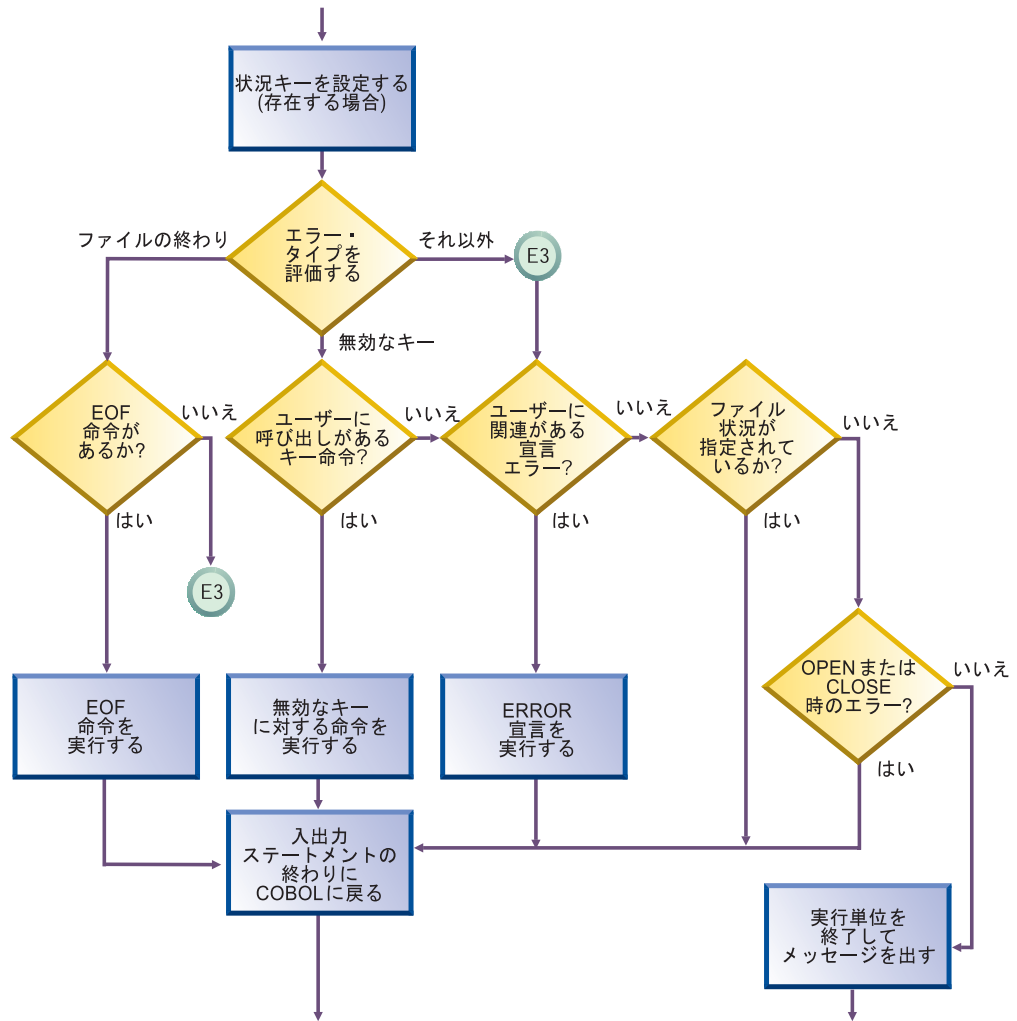
入力または出力操作が失敗しても、COBOL が自動的に訂正処置をとることはありません。重大エラーではない入出力エラー後にプログラムの実行を継続するかどうかを選択してください。

特定の入力または出力の状態またはエラーの代行受信および処理には、以下のいずれかの技法を使用することができます。

- ファイル終わり条件 (AT END)
- ERROR 宣言
- FILE STATUS 節およびファイル状況キー
- ファイル・システム状況コード
- READ または WRITE ステートメント上の命令ステートメント句
- INVALID KEY 句

プログラムを継続させる場合には、適切なエラー・リカバリー手順もコーディングする必要があります。例えば、ファイル状況キーの値を検査する手順をコーディングすることができます。入力または出力エラーをこれらのいずれかの方法で処理しなかったときは、COBOL ランタイム・メッセージが書き込まれ、実行単位が終了します。

次の図は、ファイル・システムの入力または出力エラーの後のロジック・フローを示しています。



関連タスク

- 『ファイルの終わり条件 (AT END) の使用』
- 173 ページの『ERROR 宣言のコーディング』
- 174 ページの『ファイル状況キーの使用』
- 175 ページの『ファイル・システム状況コードの使用』
- 177 ページの『INVALID KEY 句のコーディング』

関連参照

ファイル状況キー (COBOL for AIX 言語解説書)

ファイルの終わり条件 (AT END) の使用

READ ステートメントの AT END 句をコーディングすると、エラーまたは正常な条件をプログラムの設計に従って処理することができます。ファイルの終わりで、AT END 句が実行されます。AT END 句をコーディングしないと、対応する ERROR 宣言が実行されます。

多くの設計では、ファイルの終わりまでの順次読み取りが意図的に行われており、AT END 条件が予期されます。例えば、マスター・ファイルを更新するために、トランザクションが入っているファイルを処理していると想定します。

```
PERFORM UNTIL TRANSACTION-EOF = "TRUE"  
  READ UPDATE-TRANSACTION-FILE INTO WS-TRANSACTION-RECORD  
  AT END  
    DISPLAY "END OF TRANSACTION UPDATE FILE REACHED"  
    MOVE "TRUE" TO TRANSACTION-EOF  
  END READ  
  . . .  
END-PERFORM
```

NOT AT END 句が実行されるのは、READ ステートメントが正常に完了した場合だけです。ファイルの終わり以外の何らかの条件のために READ 操作が失敗すると、AT END 句も NOT AT END 句も実行されません。その代わりに、関連する宣言型プロシージャを実行した後で、READ ステートメントの終わりに制御が渡されます。

AT END 句または EXCEPTION 宣言型プロシージャのいずれもコーディングせずに、ファイルの状況キー節をコーディングすることもできます。その場合は、ファイルの終わり条件を検出した入力ステートメントまたは出力ステートメントの後の次の順次命令に、制御が渡されます。その場所に、適切な操作を実行するコードを記述する必要があります。

関連参照

AT END 句 (COBOL for AIX 言語解説書)

ERROR 宣言のコーディング

プログラムの実行時に入力または出力エラーが発生した場合に制御が与えられる ERROR 宣言型プロシージャを 1 つ以上コーディングすることができます。そのようなプロシージャをコーディングしないと、入出力エラーの発生後、ジョブが取り消されるか、異常終了します。

このようなプロシージャをそれぞれ PROCEDURE DIVISION の宣言セクションに入れます。以下のものをコーディングすることができます。

- プログラム全体用の単一の共通プロシージャ
- 各ファイル・オープン・モードのプロシージャ (INPUT、OUTPUT、I-O、または EXTEND)
- それぞれのファイルごとの個々のプロシージャ

ERROR 宣言プロシージャでは、訂正処置の試行、操作の再試行、実行の継続または終了などをコーディングすることができます。(ただし、ブロック化ファイルの処理を継続する場合、エラーが発生したレコードより後ろにあるブロック内の残りのレコードが失われることがあります。) エラーについてさらに詳しい分析を行う場合は、ERROR 宣言型プロシージャとファイル状況キーを組み合わせで使用することができます。

関連参照

EXCEPTION/ERROR 宣言 (COBOL for AIX 言語解説書)

ファイル状況キーの使用

それぞれの入力または出力ステートメントがファイルに対して実行された後、システムはファイル状況キーの 2 つの桁位置の値を更新します。一般に、最初の桁がゼロの場合、操作が正常に行われたことを表し、両方の桁がゼロの場合、異常がなかったことを意味します。

ファイル状況キーは、次のようにコーディングして設定してください。

- FILE-CONTROL 段落の FILE STATUS 節:

```
FILE STATUS IS data-name-1
```

- DATA DIVISION (WORKING-STORAGE、LOCAL-STORAGE、または LINKAGE SECTION) のデータ定義 (一例として):

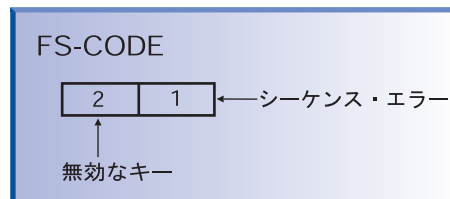
```
WORKING-STORAGE SECTION.  
01 data-name-1 PIC 9(2) USAGE NATIONAL.
```

ファイル状況キー *data-name-1* を、2 文字のカテゴリ-英数字またはカテゴリ-国別項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目として指定してください。この *data-name-1* を可変位置にすることはできません。

プログラムはファイル状況キーを検査して、エラーが発生したかどうか、また発生した場合にはどんなタイプのエラーが発生したかを発見できます。例えば、FILE STATUS 節が

```
FILE STATUS IS FS-CODE
```

FS-CODE は、次のような状況に関する情報を保持するために、COBOL によって使用されます。



各ファイルごとに、次の規則に従ってください。

- ファイルごとに異なるファイル状況キーを定義します。

すると、アプリケーション論理エラーやディスク・エラーのような、ファイル入力または出力例外の原因を判別することができます。

- それぞれの入力または出力要求の後で、ファイル状況キーを検査します。

ファイル状況キーが 0 以外の値を含んでいる場合、プログラムはエラー・メッセージを発生するか、またはその値に基づいてアクションを実行できます。

ファイル状況キー・コードをリセットする必要はありません。ファイル状況キー・コードは各入出力が試みられた後で設定されます。

ファイル状況キーに加えて、2 番目の ID を FILE STATUS 節にコーディングして、ファイル・システムの入力または出力要求に関するさらに詳細な情報を取得できます。ファイル・システム状況コードについては、下記の関連参照を参照してください。

ファイル状況キーは単独でも、INVALID KEY オプションと一緒に使用でき、EXCEPTION または ERROR 宣言を補足するためにも使用できます。このようにファイル状況キーを使用すると、それぞれの入力または出力操作の結果に関する正確な情報が得られます。

『例: ファイル状況キー』

176 ページの『例: ファイル・システム状況コードの検査』

関連タスク

136 ページの『ファイル状況フィールドの設定』

『ファイル・システム状況コードの使用』

関連参照

FILE STATUS 節 (*COBOL for AIX 言語解説書*)

ファイル状況キー (*COBOL for AIX 言語解説書*)

例: ファイル状況キー

次の例は、ファイルを開いた後にファイル状況キーの簡単な検査を行う方法を示しています。

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. SIMCHK.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT MASTERFILE ASSIGN TO AS-MASTERA  
    FILE STATUS IS MASTER-CHECK-KEY  
    . . .  
DATA DIVISION.  
    . . .  
WORKING-STORAGE SECTION.  
01 MASTER-CHECK-KEY      PIC X(2).  
    . . .  
PROCEDURE DIVISION.  
    OPEN INPUT MASTERFILE  
    IF MASTER-CHECK-KEY NOT = "00"  
        DISPLAY "Nonzero file status returned from OPEN " MASTER-CHECK-KEY  
    . . .
```

ファイル・システム状況コードの使用

要求の処理を正確に特定する上で、2 桁のファイル状況コードは一般的過ぎることがよくあります。FILE STATUS 文節に 2 番目のデータ項目をコーディングすることにより、STL および VSAM ファイル・システムの入力または出力要求に関する詳細情報を取得することができます。

```
FILE STATUS IS data-name-1 data-name-8
```

データ項目 *data-name-1* は 2 桁の COBOL ファイル状況キーを指定します。これは、2 文字のカテゴリ-英数字またはカテゴリ-国別項目、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目でなければなりません。データ項目 *data-name-8*

は、COBOL ファイル状況キーが 0 でないときにファイル・システムの状況コードが入れられるデータ項目を指定します。 *data-name-8* の長さは 6 バイト以上で、英数字項目でなければなりません。

STL ファイル・システム入出力要求に関して、*data-name-8* が 6 バイト長の場合は状況コードが含まれ、*data-name-8* が 6 バイト長を超える場合は詳細情報付きのメッセージも含まれます。

```
01 my-file-status-2.  
   02 exception-return-value PIC 9(6).  
   02 additional-info       PIC X(100).
```

上記の例で、ファイルの作成に使用した定義と異なる定義でファイルをオープンしようとする、戻りコード 39 が *exception-return-value* で戻され、オープンを実行するために必要なキーを示すメッセージが *additional-info* で戻されます。

VSAM ファイル・システム入出力要求に関して、PICTURE *X(n)* 属性と USAGE DISPLAY 属性を使用して *data-name-8* を定義します (*n* は 6 以上の数字です)。 PICTURE スtring値は、VSAM 応答メッセージ構造の最初の *n* バイトを表します。 応答メッセージ構造のサイズ、*m* が *n* より小さい場合、有用な情報は最初の *m* バイトのみです。

『例: ファイル・システム状況コードの検査』

関連参照

125 ページの『VSAM ファイル・システム』

127 ページの『STL ファイル・システム』 VSAM ファイル・システム 応答メッセージ

例: ファイル・システム状況コードの検査

次の例は、索引付きファイルを 5 番目のレコードから読み取り、それぞれの入力または出力要求の後で、ファイル状況キーを検査します。 ファイル状況キーがゼロ以外の場合に、VSAM コードが表示されます。

さらに、以下に、処理中のファイルに 6 つのレコードが入っていたことを想定した場合の、このプログラムからの出力を図示しています。

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. EXAMPLE.  
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT FILESYSFILE ASSIGN TO FILESYSFILE  
    ORGANIZATION IS INDEXED  
    ACCESS DYNAMIC  
    RECORD KEY IS FILESYSFILE-KEY  
    FILE STATUS IS FS-CODE, FILESYS-CODE.  
DATA DIVISION.  
FILE SECTION.  
FD FILESYSFILE  
   RECORD 30.  
01 FILESYSFILE-REC.  
   10 FILESYSFILE-KEY          PIC X(6).  
   10 FILLER                   PIC X(24).  
WORKING-STORAGE SECTION.  
01 RETURN-STATUS.  
   05 FS-CODE                  PIC XX.
```



```

05 FILESYS-CODE          PIC X(6).
PROCEDURE DIVISION.
OPEN INPUT FILESYSFILE.
DISPLAY "OPEN INPUT FILESYSFILE FS-CODE: " FS-CODE.

IF FS-CODE NOT = "00"
    PERFORM FILESYS-CODE-DISPLAY
    STOP RUN
END-IF.

MOVE "000005" TO FILESYSFILE-KEY.
START FILESYSFILE KEY IS EQUAL TO FILESYSFILE-KEY.
DISPLAY "START FILESYSFILE KEY=" FILESYSFILE-KEY
      " FS-CODE: " FS-CODE.

IF FS-CODE NOT = "00"
    PERFORM FILESYS-CODE-DISPLAY
END-IF.

IF FS-CODE = "00"
    PERFORM READ-NEXT UNTIL FS-CODE NOT = "00"
END-IF.

CLOSE FILESYSFILE.
STOP RUN.

READ-NEXT.
READ FILESYSFILE NEXT.
DISPLAY "READ NEXT FILESYSFILE FS-CODE: " FS-CODE.
IF FS-CODE NOT = "00"
    PERFORM FILESYS-CODE-DISPLAY
END-IF.
DISPLAY FILESYSFILE-REC.

FILESYS-CODE-DISPLAY.
DISPLAY "FILESYS-CODE ==>", FILESYS-CODE.

```

INVALID KEY 句のコーディング

INVALID KEY 句を、索引付きおよび相対ファイルに対する、READ、START、WRITE、REWRITE、および DELETE ステートメントに組み込むことができます。INVALID KEY 句に制御が与えられるのは、誤った索引キーのために入力エラーまたは出力エラーが発生した場合です。

状況キーを評価し、特定の INVALID KEY 条件を判別するには、INVALID KEY 句と一緒に FILE STATUS 節を使用してください。

INVALID KEY 句は、いくつかの点で ERROR 宣言とは異なります。INVALID KEY 句:

- 限られたタイプのエラーのみで作動します。ERROR 宣言は、すべての形式をカバーします。
- 入出力動詞に直接コーディングされます。ERROR 宣言は、別途にコーディングされます。
- 単一の入出力操作で固有です。ERROR 宣言はより一般的です。

INVALID KEY 条件を引き起こすステートメントで INVALID KEY をコーディングすると、制御は INVALID KEY 命令ステートメントに転送されます。コーディングした ERROR 宣言は実行されません。

NOT INVALID KEY 句をコーディングした場合、ステートメントが正常に完了したときにのみ実行されます。INVALID KEY 以外の条件のために操作が失敗すると、INVALID KEY 句も NOT INVALID KEY 句も実行されません。代わりに、関連した ERROR 宣言をプログラムが実行した後、制御はステートメントの最後に渡されます。

『例: FILE STATUS および INVALID KEY』

例: FILE STATUS および INVALID KEY

次の例は、ファイル状況コードおよび INVALID KEY 句を使用して、入力または出力ステートメントが失敗した理由をもっと明確に判別する方法を示しています。

マスター顧客レコードを含んでいるファイルがあり、トランザクション更新ファイルの情報が反映されるように、それらのレコードの一部を更新する必要があると想定しましょう。プログラムは、各トランザクション・レコードを読み取り、マスター・ファイルの中の対応するレコードを見つけ、必要な更新を行います。どちらのファイルのレコードにもそれぞれ顧客番号用のフィールドがあり、マスター・ファイルの中の各レコードには固有の顧客番号があります。

顧客レコードのマスター・ファイル用の FILE-CONTROL 記入項目には、索引編成を定義するステートメント、ランダム・アクセス、基本レコード・キーとして MASTER-CUSTOMER-NUMBER、およびファイル状況キーとして CUSTOMER-FILE-STATUS が含まれています。

```
.  
. (read the update transaction record)  
.  
MOVE "TRUE" TO TRANSACTION-MATCH  
MOVE UPDATE-CUSTOMER-NUMBER TO MASTER-CUSTOMER-NUMBER  
READ MASTER-CUSTOMER-FILE INTO WS-CUSTOMER-RECORD  
  INVALID KEY  
    DISPLAY "MASTER CUSTOMER RECORD NOT FOUND"  
    DISPLAY "FILE STATUS CODE IS: " CUSTOMER-FILE-STATUS  
    MOVE "FALSE" TO TRANSACTION-MATCH  
END-READ
```

プログラム呼び出し時のエラーの処理

プログラムが、別々にコンパイルされたプログラムを動的に呼び出すとき、呼び出されるプログラムが使用できないことがあります。例えば、システムがストレージ不足だったり、ロード・モジュールを見つけることができない場合です。CALL ステートメントに ON EXCEPTION 句も ON OVERFLOW 句もない場合、アプリケーションは異常終了する可能性があります。

一連のステートメントを実行してユーザー定義のエラー処理を行う場合は、ON EXCEPTION 句を使用します。例えば、以下のフラグメントでは、プログラム REPORTA が利用不可である場合、制御は ON EXCEPTION 句に渡されます。

```
MOVE "REPORTA" TO REPORT-PROG  
CALL REPORT-PROG  
  ON EXCEPTION  
    DISPLAY "Program REPORTA not available, using REPORTB."
```

```
MOVE "REPORTB" TO REPORT-PROG
CALL REPORT-PROG
END-CALL
END-CALL
```

| ON EXCEPTION 句は、初期ロードでの呼び出し先プログラムの可用性に対してのみ適
| 用されます。呼び出し先プログラムがロードされたが、何かしらの理由（初期設定
| など）で失敗した場合、ON EXCEPTION 句は実行されません。

第 2 部 各国語環境に合わせたプログラムの対応

第 10 章 国際環境でのデータの処理	183	アクティブ・ロケールおよびコード・ページ値への	
COBOL ステートメントと国別データ	184	アクセス	220
組み込み関数と国別データ	187	例: コード・ページ ID の取得および変換	221
Unicode および言語文字のエンコード	188		
COBOL での国別データ (Unicode) の使用	189		
国別データ項目の定義	189		
国別リテラルの使用	190		
国別文字形象定数の使用	191		
国別数値データ項目の定義	192		
国別グループ	192		
国別グループの使用	194		
国別グループを基本項目として使用	195		
国別グループをグループ項目として使用	195		
文字データの保管	196		
国別 (Unicode) 表現との間の変換	197		
英数字、DBCS、および整数から国別への変換			
(MOVE)	198		
英数字または DBCS から国別への変換			
(NATIONAL-OF)	198		
国別の英数字への変換 (DISPLAY-OF)	199		
デフォルト・コード・ページのオーバーライド	199		
例: 国別データと間の変換	200		
UTF-8 データの処理	201		
中国語 GB 18030 データの処理	201		
国別 (UTF-16) データの比較	202		
2 つのクラス国別オペランドの比較	203		
クラス国別オペランドとクラス数値オペランドの			
比較	204		
国別数値オペランドと他の数値オペランドの比較	204		
国別と他の文字ストリング・オペランドとの比較	204		
国別データ・オペランドと英数字グループ・オペ			
ランドの比較	205		
DBCS サポート用のコーディング	205		
DBCS データの宣言	206		
DBCS リテラルの使用	206		
DBCS リテラルの比較	207		
有効な DBCS 文字に関するテスト	207		
DBCS データを含む英数字データ項目の処理	208		
第 11 章 ロケールの設定	209		
アクティブ・ロケール	209		
文字データのコード・ページの指定	210		
環境変数を使用したロケールの指定	211		
システム設定からのロケールの決定	212		
変換が使用可能なメッセージのタイプ	213		
サポートされるロケールおよびコード・ページ	213		
ロケール付きの照合シーケンスの制御	216		
ロケール付きの英数字照合シーケンスの制御	217		
ロケール付きの DBCS 照合シーケンスの制御	218		
ロケール付きの国別照合シーケンスの制御	219		
照合シーケンスに依存する組み込み関数	219		

第 10 章 国際環境でのデータの処理

COBOL for AIX は、実行時に国別文字データとして Unicode UTF-16 をサポートします。UTF-16 は、プレーン・テキストをエンコードするための一貫性のある効率的な方法を提供します。UTF-16 を使用すると、さまざまな国の言語で動作するソフトウェアを開発できます。

以下の COBOL 機能を使用して、国別データおよびそのようなデータの文化的に依存した照合順序を処理するプログラムのコーディングおよびコンパイルを行います。

- データ型およびリテラル:
 - 文字データ型。USAGE NATIONAL 節や、カテゴリ国別、国別編集、または数字編集のデータを定義する PICTURE 節で定義します。
 - 数値データ型。USAGE NATIONAL 節や、数値データ項目 (国別 10 進数項目) または外部浮動小数点データ項目 (国別浮動小数点項目) を定義する PICTURE 節で定義します。
 - リテラル接頭部 N または NX で指定される国別リテラル
 - 形象定数 ALL *national-literal*
 - 形象定数 QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO。これらは、国別文字コンテキストで使用されるときには国別文字 (UTF-16) 値を持ちます。
- COBOL ステートメント。COBOL ステートメントおよび国別データに関する以下の関連参照に示されています。
- 組み込み関数
 - NATIONAL-OF は、英数字または 2 バイト文字セット (DBCS) 文字ストリングを USAGE NATIONAL (UTF-16) に変換します。
 - DISPLAY-OF は、国別文字ストリングを選択されたコード・ページ (EBCDIC、ASCII、EUC、または UTF-8) の USAGE DISPLAY に変換します。
 - その他の組み込み関数は、組み込み関数および国別データに関する以下の関連参照に示されています。
- GROUP-USAGE NATIONAL 節。USAGE NATIONAL データ項目のみを含み、ほとんどの操作でカテゴリ国別基本項目と同様に振る舞う、グループを定義するためのものです。
- コンパイラー・オプション:
 - NSYMBOL は、リテラル内の N 記号および PICTURE 節に対して国別処理と DBCS 処理のどちらを使用するかを制御します。
 - NCOLLSEQ。国別オペランドの比較のために、照合シーケンスを指定します。

英数字または DBCS データ項目から国別表現への暗黙変換を利用することもできます。ユーザーがこれらの項目を国別データ項目へ移動させるとき、またはこれらの項目を国別データ項目と比較するとき、コンパイラーは (ほとんどの場合に) この変換を実行します。

関連概念

- 188 ページの『Unicode および言語文字のエンコード』
- 192 ページの『国別グループ』

関連タスク

- 189 ページの『COBOL での国別データ (Unicode) の使用』
- 197 ページの『国別 (Unicode) 表現と間の変換』
- 201 ページの『UTF-8 データの処理』
- 201 ページの『中国語 GB 18030 データの処理』
- 202 ページの『国別 (UTF-16) データの比較』
- 205 ページの『DBCS サポート用のコーディング』
- 209 ページの『第 11 章 ロケールの設定』

関連参照

- 『COBOL ステートメントと国別データ』
- 187 ページの『組み込み関数と国別データ』
- 296 ページの『NCOLLSEQ』
- 296 ページの『NSYMBOL』
- データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)
- データ・カテゴリーおよび PICTURE 規則 (COBOL for AIX 言語解説書)
- MOVE ステートメント (COBOL for AIX 言語解説書)
- 一般比較条件 (COBOL for AIX 言語解説書)

COBOL ステートメントと国別データ

PROCEDURE DIVISION および以下の表に示すコンパイラー指示ステートメントで、国別データを使用できます。

表 19. COBOL ステートメントと国別データ

COBOL ステートメント	国別にできるもの	説明	詳細の参照先
ACCEPT	<i>identifier-1, identifier-2</i>	<i>identifier-1</i> がランタイム・ロケールで指示されたコード・ページから変換されるのは、入力が端末からのものである場合のみです。	35 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』
ADD	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 <i>identifier-3</i> (GIVING) は、USAGE NATIONAL を持つ数字編集にすることができます。		54 ページの『COMPUTE およびその他の算術ステートメントの使用』
CALL	<i>identifier-2, identifier-3, identifier-4, identifier-5; literal-2, literal-3</i>		517 ページの『データの受け渡し』

表 19. COBOL ステートメントと国別データ (続き)

COBOL ステートメント	国別にできるもの	説明	詳細の参照先
COMPUTE	<i>identifier-1</i> は、USAGE NATIONAL を持つ数値または数字編集にすることができます。 <i>arithmetic-expression</i> は、USAGE NATIONAL を持つ数値項目を含むことができます。		54 ページの『COMPUTE およびその他の算術ステートメントの使用』
COPY . . . REPLACING	REPLACING 句の <i>operand-1</i> 、 <i>operand-2</i>		319 ページの『第 15 章 コンパイラ指示ステートメント』
DISPLAY	<i>identifier-1</i>	<i>identifier-1</i> は、現在のロケールに関連付けられているコード・ページに変換されます。	37 ページの『画面上またはファイル内での値の表示 (DISPLAY)』
DIVIDE	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 <i>identifier-3</i> (GIVING) および <i>identifier-4</i> (REMAINDER) は、USAGE NATIONAL を持つ数字編集にすることができます。		54 ページの『COMPUTE およびその他の算術ステートメントの使用』
INITIALIZE	<i>identifier-1</i> 。REPLACING 句の <i>identifier-2</i> または <i>literal-1</i> 。	REPLACING NATIONAL または REPLACING NATIONAL-EDITED を指定する場合、 <i>identifier-2</i> または <i>literal-1</i> は、 <i>identifier-1</i> への移動における送信オペランドとして有効でなければなりません。	28 ページの『例: データ項目の初期化』
INSPECT	ID はすべてリテラルです。(TALLYING 整数データ項目である <i>identifier-2</i> は USAGE NATIONAL を持つことができます。)	これらのいずれか (TALLYING ID である <i>identifier-2</i> を除く) が USAGE NATIONAL を持っている場合、すべてが国別でなければなりません。	110 ページの『データ項目の計算および置換 (INSPECT)』
INVOKE	<i>identifier-2</i> または <i>literal-1</i> としてのメソッド名。BY VALUE 句の <i>identifier-3</i> または <i>literal-2</i> 。		450 ページの『メソッドの呼び出し (INVOKE)』
MERGE	マージ・キー、NCOLLSEQ(BIN) を指定した場合	COLLATING SEQUENCE 句は適用されません。	162 ページの『ソートまたはマージ基準の設定』
MOVE	送り出し側と受け取り側の両方、または受け取り側のみ	有効な MOVE オペランドについては、暗黙変換が実行されます。	33 ページの『基本データ項目への値の割り当て (MOVE)』 34 ページの『グループ・データ項目への値の割り当て (MOVE)』

表 19. COBOL ステートメントと国別データ (続き)

COBOL ステートメント	国別にできるもの	説明	詳細の参照先
MULTIPLY	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 <i>identifier-3</i> (GIVING) は、USAGE NATIONAL を持つ数字編集にすることができます。		54 ページの『COMPUTE およびその他の算術ステートメントの使用』
SEARCH ALL (二分探索)	キー・データ項目とその比較対象の両方	キー・データ項目とその比較対象は、比較規則に従って互換性がなければなりません。比較対象がクラス国別である場合、キーもそうでなければなりません。	82 ページの『二分探索 (SEARCH ALL)』
SORT	ソート・キー、NCOLLSEQ(BIN) を指定した場合	COLLATING SEQUENCE 句は適用されません。	162 ページの『ソートまたはマージ基準の設定』
STRING	ID はすべてリテラルです。(POINTER 整数データ項目である <i>identifier-4</i> は USAGE NATIONAL を持つことができます。)	<i>identifier-3</i> (受信データ項目) が国別である場合、すべての ID およびリテラル (POINTER ID である <i>identifier-4</i> を除く) は国別でなければなりません。	99 ページの『データ項目の結合 (STRING)』
SUBTRACT	ID はすべて USAGE NATIONAL を持つ数値項目にすることができます。 <i>identifier-3</i> (GIVING) は、USAGE NATIONAL を持つ数字編集にすることができます。		54 ページの『COMPUTE およびその他の算術ステートメントの使用』
UNSTRING	ID はすべてリテラルです。(<i>identifier-6</i> および <i>identifier-7</i> (それぞれ COUNT および TALLYING 整数データ項目) は USAGE NATIONAL を持つことができます。)	<i>identifier-4</i> (受信データ項目) が USAGE NATIONAL を持っている場合、送信データ項目およびそれぞれの区切り文字は USAGE NATIONAL を持っている必要があり、それぞれのリテラルは国別でなければなりません。	102 ページの『データ項目の分割 (UNSTRING)』
XML GENERATE	<i>identifier-1</i> (生成された XML 文書)、 <i>identifier-2</i> (ソース・フィールド)、 <i>identifier-4</i> または <i>literal-4</i> (名前空間 ID)、 <i>identifier-5</i> または <i>literal-5</i> (名前空間接頭部)		407 ページの『第 22 章 XML 出力の生成』

表 19. COBOL ステートメントと国別データ (続き)

COBOL ステートメント	国別にできるもの	説明	詳細の参照先
XML PARSE	<i>identifier-1</i> (XML 文書)	XML-NTEXT 特殊レジスターには、構文解析時に国別文字文書フラグメントが入ります。	387 ページの『第 21 章 XML 入力の処理』

関連タスク

- 41 ページの『数値データの定義』
- 43 ページの『数値データの表示』
- 189 ページの『COBOL での国別データ (Unicode) の使用』
- 202 ページの『国別 (UTF-16) データの比較』

関連参照

- 296 ページの『NCOLLSEQ』
- データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)

組み込み関数と国別データ

以下の表に示す組み込み関数で、クラス国別の引数を使用できます。

表 20. 組み込み関数と国別文字データ

組み込み関数	関数型	詳細の参照先
DISPLAY-OF	英数字	199 ページの『国別の英数字への変換 (DISPLAY-OF)』
LENGTH	整数	118 ページの『データ項目の長さの検出』
LOWER-CASE、UPPER-CASE	国別	112 ページの『大文字または小文字への変換 (UPPER-CASE、LOWER-CASE)』
NUMVAL、NUMVAL-C	数値	113 ページの『数値への変換 (NUMVAL、NUMVAL-C)』
MAX、MIN	国別	115 ページの『最大または最小データ項目の検出』
ORD-MAX、ORD-MIN	整数	115 ページの『最大または最小データ項目の検出』
REVERSE	国別	112 ページの『逆順への変換 (REVERSE)』

ゾーン 10 進数引数が許可されていれば、国別 10 進数引数を使用できます。表示浮動小数点引数が許可されていれば、国別浮動小数点引数を使用できます。(整数または数値引数を取ることのできる組み込み関数の完全なリストについては、引数に関する以下の関連参照を参照してください。)

関連タスク

- 41 ページの『数値データの定義』
- 189 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

- 引数 (COBOL for AIX 言語解説書)
- データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)

Unicode および言語文字のエンコード

COBOL for AIX では、Unicode の基本的な実行時サポートを提供しています。Unicode では、全世界で一般的に使用されている文字や記号をすべて網羅する数万文字の取り扱いが可能となります。

文字セットは、定義された文字のセットですが、コード化表現と関連してはいません。コード化文字セット (本書ではコード・ページ とも呼んでいます) は、セットの文字をそのコード化表現に関係付ける明確な規則セットです。各コード・ページには名前があり、文字セットを表現するための記号を設定した一種のテーブルとなっています。それぞれの記号は、固有のビット・パターン、すなわちコード・ポイント を持ちます。コード・ページにはそれぞれ、コード化文字セット ID (CCSID) があり、1 から 65,536 までの値をとります。

Unicode には、*Unicode Transformation Format (UTF)* と呼ばれるいくつかのエンコード・スキーム (UTF-8、UTF-16、および UTF-32 など) があります。COBOL for AIX では、国別リテラルおよび USAGE NATIONAL を持つデータ項目の表現として、ビッグ・エンディアン・フォーマットの UTF-16 (CCSID 1200) を使用します。

UTF-8 は、ASCII インバリエント文字 a から z、A から Z、0 から 9、および特殊文字 (' @ , . + - = / * () など) を、ASCII で表現される場合と同様に表します。UTF-16 は、これらの文字を NX'00nn' として表します (ここで、X'nn' は ASCII での文字表現です)。

例えば、ストリング「ABC」は、UTF-16 では NX'004100420043' として表されます。UTF-8 では、「ABC」は X'414243' として表されます。

1 つ以上のエンコード・ユニット を使用して、コード化文字セットから文字を表します。UTF-16 の場合、エンコード・ユニットは 2 バイトのストレージを使用します。任意の EBCDIC、ASCII、または EUC コード・ページで定義された文字はいずれも、国別データ表現に変換されたときに 1 つの UTF-16 エンコード・ユニットで表現されます。

クロスプラットフォームに関する考慮事項: Enterprise COBOL for z/OS および COBOL for AIX は、国別データでビッグ・エンディアン・フォーマットの UTF-16 をサポートします。COBOL for Windows は、国別データでリトル・エンディアン・フォーマットの UTF-16 (UTF-16LE) をサポートします。UTF-16LE 表現でエンコードされた Unicode データを別のプラットフォームから COBOL for AIX へ移植する場合、そのデータをビッグ・エンディアン・フォーマットの UTF-16 に変換してデータを国別データとして処理する必要があります。このような変換は、COBOL for AIX では、NATIONAL-OF 組み込み関数を使用して行うことができます。

関連タスク

197 ページの『国別 (Unicode) 表現との変換』

関連参照

196 ページの『文字データの保管』

213 ページの『サポートされるロケールおよびコード・ページ』
文字セットとコード・ページ (COBOL for AIX 言語解説書)

COBOL での国別データ (Unicode) の使用

COBOL for AIX では、いくつかの方法で国別 (UTF-16) データを指定できます。

次の国別データ型が使用可能です。

- 国別データ項目 (カテゴリー国別、国別編集、および数字編集)
- 国別リテラル
- 国別文字としての形象定数
- 数値データ項目 (国別 10 進数および国別浮動小数点)

加えて、明示的または暗黙的に USAGE NATIONAL を持つデータ項目のみを含んでおり、ほとんどの操作でカテゴリー国別基本項目と同様に振る舞う、国別グループを定義できます。

これらの宣言は、必要とされるストレージ量に影響します。

関連概念

188 ページの『Unicode および言語文字のエンコード』

192 ページの『国別グループ』

関連タスク

『国別データ項目の定義』

190 ページの『国別リテラルの使用』

191 ページの『国別文字形象定数の使用』

192 ページの『国別数値データ項目の定義』

194 ページの『国別グループの使用』

197 ページの『国別 (Unicode) 表現との間の変換』

202 ページの『国別 (UTF-16) データの比較』

関連参照

196 ページの『文字データの保管』

データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)

国別データ項目の定義

国別 (UTF-16) 文字ストリングを保持する国別データ項目を、USAGE NATIONAL 節で定義します。

以下のカテゴリーの国別データ項目を定義できます。

- 国別
- 国別編集
- 数字編集

カテゴリー国別データ項目を定義するには、1 つ以上の PICTURE 記号 N のみを含む PICTURE 節をコーディングしてください。

国別編集データ項目を定義するには、以下のそれぞれの記号の少なくとも 1 つを含む PICTURE 節をコーディングしてください。

- 記号 N

- 単純追加編集記号 B、0、または /

クラス国別の数字編集データ項目を定義するには、数字編集項目を定義する PICTURE 節 (例えば、-\$999.99) をコーディングしてください。また、USAGE NATIONAL 節をコーディングしてください。USAGE NATIONAL を持つ数字編集データ項目は、USAGE DISPLAY を持つ数字編集項目を使用するのと同様に使用できます。

また、PICTURE 節により数値として定義された基本項目に BLANK WHEN ZERO 節をコーディングすれば、データ項目を数字編集として定義することもできます。

PICTURE 節をコーディングしたが、1 つ以上の PICTURE 記号 N のみを含むデータ項目用に USAGE 節をコーディングしなかった場合、コンパイラー・オプション NSYMBOL(NATIONAL) を使用して、そうした項目が国別データ項目 (DBCS 項目ではなく) として取り扱われるようにしてください。

関連タスク

43 ページの『数値データの表示』

関連参照

296 ページの『NSYMBOL』

BLANK WHEN ZERO 節 (COBOL for AIX 言語解説書)

国別リテラルの使用

国別リテラルを指定するには、接頭部文字 N を使用し、オプション NSYMBOL(NATIONAL) を指定してコンパイルします。

次のいずれかの表記を使用できます。

- N"character-data"
- N'character-data'

オプション NSYMBOL(DBCS) を指定してコンパイルする場合、リテラル接頭部文字 N は国別リテラルではなく DBCS リテラルを指定します。

国別リテラルを 16 進値として指定するには、接頭部 NX を使用します。次のいずれかの表記を使用できます。

- NX"hexadecimal-digits"
- NX'hexadecimal-digits'

次の MOVE ステートメントのそれぞれは、国別データ項目 Y を文字「AB」の UTF-16 値に設定します。

```
01 Y pic NN usage national.  
...  
    Move NX"00410042" to Y  
    Move N"AB"        to Y  
    Move "AB"         to Y
```

国別リテラルを必要とするコンテキストで英数字 16 進数リテラルを使用しないでください。そのような使用法は誤解を招きやすくなります。例えば、次のステートメントの場合も、UTF-16 文字「AB」(16 進ビット・パターン 4142 ではない) が、Y に移動されます (Y は、USAGE NATIONAL で定義されます)。

Move X"4142" to Y

国別リテラルは、SPECIAL-NAMES 段落で使用したり、プログラム名として使用したりすることはできません。国別リテラルは、METHOD-ID 段落のオブジェクト指向メソッドを指定したり、INVOKE ステートメント内のメソッド名を指定したりするのに使用できます。

SOSI コンパイラー・オプションを使用して、国別リテラル内のシフトアウトおよびシフトインの処理方法を制御します。

関連タスク

26 ページの『リテラルの使用』

関連参照

296 ページの『NSYMBOL』

305 ページの『SOSI』

国別リテラル (*COBOL for AIX 言語解説書*)

国別文字形象定数の使用

国別文字を必要とするコンテキストでは、形象定数の ALL *national-literal* を使用できます。ALL 国別リテラル は、国別リテラルを構成する連続したエンコード・ユニットの連結によって生成される、ストリングの全部または一部を表します。

国別文字を必要とするコンテキスト (MOVE ステートメント、暗黙移動、または、国別オペランドを持つ比較条件など) では、形象定数 QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO を使用できます。こうしたコンテキストでは、形象定数は国別文字 (UTF-16) 値を表します。

国別文字を必要とするコンテキストで形象定数 HIGH-VALUE を使用すると、その値は NX'FFFF' です。国別文字を必要とするコンテキストで LOW-VALUE を使用すると、その値は NX'0000' です。NCOLLSEQ(BIN) コンパイラー・オプションが有効化されている場合にのみ、国別文字が必要なコンテキストで、HIGH-VALUE または LOW-VALUE を使用できます。

制約事項: HIGH-VALUE または HIGH-VALUE から割り当てられた値は、あるデータ表現から別のデータ表現への値の変換 (例えば、USAGE DISPLAY と USAGE NATIONAL の間の変換、または CHAR(EBCDIC) コンパイラー・オプションが有効な場合の ASCII と EBCDIC の間の変換) が起こるような仕方では使用してはなりません。X'FF' (EBCDIC 照合シーケンスが使用されているときの、英数字コンテキストでの HIGH-VALUE の値) は有効な EBCDIC または ASCII 文字を表しませんし、NX'FFFF' は有効な国別文字を表しません。このような値を別の表現に変換すると、置換文字が使用されることとなります (X'FF' でも NX'FFFF' でもなくなります)。次の例を見てください。

```
01 nat1-data PIC NN Usage National.
01 alph-data PIC XX.
. . .
MOVE HIGH-VALUE TO nat1-data, alph-data
IF nat1-data = alph-data. . .
```

上の IF ステートメントは、オペランドのそれぞれが HIGH-VALUE に設定された場合であっても、偽と評価されます。基本英数字オペランドが国別オペランドと比較される前に、英数字オペランドは、一時国別データ項目に移動させられたかのように扱われ、英数字文字は対応する国別文字に変換されます。しかし、X'FF' が UTF-16 に変換される場合、UTF-16 項目は置換文字値を取得するので、NX'FFFF' と比較して等しいとはみなされません。

関連タスク

197 ページの『国別 (Unicode) 表現と間の変換』
202 ページの『国別 (UTF-16) データの比較』

関連参照

271 ページの『CHAR』
296 ページの『NCOLLSEQ』
形象定数 (COBOL for AIX 言語解説書)
DISPLAY-OF (「COBOL for AIX 言語解説書」)

国別数値データ項目の定義

国別文字 (UTF-16) で表される数値データを保持するデータ項目を、USAGE NATIONAL 節で定義します。国別 10 進数項目および国別浮動小数点項目を定義できます。

国別 10 進数項目を定義するには、記号 9、P、S、および V のみを含む PICTURE 節をコーディングしてください。PICTURE 節が S を含んでいる場合、その項目で SIGN IS SEPARATE 節が有効でなければなりません。

国別浮動小数点項目を定義するには、浮動小数点項目を定義する PICTURE 節をコーディングしてください (例えば、+99999.9E-99)。

国別 10 進数項目は、ゾーン 10 進数項目と同じように使用できます。国別浮動小数点項目は、表示浮動小数点項目と同じように使用できます。

関連タスク

41 ページの『数値データの定義』
43 ページの『数値データの表示』

関連参照

SIGN 節 (COBOL for AIX 言語解説書)

国別グループ

GROUP-USAGE NATIONAL 節で明示的または暗黙的に指定される国別グループは、USAGE NATIONAL を持つデータ項目のみを含みます。ほとんどの場合、国別グループ項目は、PIC N(m) (ここで、m はグループ内の国別 (UTF-16) 文字の数です) として記述されたカテゴリ—国別基本項目として再定義されているかのように処理されます。

ただし、国別グループに対する操作の中には (英数字グループに対する一部の操作の場合と同様に)、グループ・セマンティクスが適用されるものがあります。そのような操作 (例えば、MOVE CORRESPONDING や INITIALIZE) は、国別グループ内の基本項目を認識または処理します。

可能な場合、USAGE NATIONAL 項目を含んでいる英数字グループではなく、国別グループを使用してください。国別グループでの国別データの処理の場合、英数字グループ内の国別データの処理と比較して、いくつかの利点があります。

- 国別グループを、USAGE NATIONAL を持つもっと長いデータ項目に移動させると、受信項目に国別文字が埋め込まれます。これに対して、国別文字を含む英数字グループを、国別文字を含むもっと長い英数字グループに移動させると、埋め込みには英数字スペースが使用されます。その結果、データ項目の取り扱いを誤ることがあります。
- 国別グループを、USAGE NATIONAL を持つもっと短いデータ項目に移動させると、国別グループは国別文字境界で切り捨てられます。これに対して、国別文字を含む英数字グループを、国別文字を含むもっと短い英数字グループに移動させると、国別文字の 2 バイト間で切り捨てが起こります。
- 国別グループを国別編集または数字編集項目に移動させると、グループの内容が編集されます。これに対し、英数字グループを編集項目に移動させた場合、編集は行われません。
- 国別グループを、STRING、UNSTRING、または INSPECT ステートメントのオペランドとして使用した場合、次のようになります。
 - グループの内容は、1 バイト文字としてではなく、国別文字として処理されます。
 - TALLYING および POINTER オペランドは、国別文字の論理レベルで作動します。
 - 国別グループ・オペランドは、他の国別オペランド・タイプの混じり合ったものと一緒にサポートされます。

これに対し、これらのコンテキストで国別文字を含む英数字グループを使用した場合、文字はバイトごとに処理されます。結果として、取り扱いが無効になったり、データの破壊が起こることがあります。

USAGE NATIONAL グループ: グループ項目では、グループ内のそれぞれの基本データ項目の USAGE の便利な省略表現として、グループ・レベルで USAGE NATIONAL 節を指定できます。ただし、このようなグループは国別グループではなく、英数字グループであり、多数の操作 (移動や比較など) において USAGE DISPLAY の基本データ項目のように振る舞います (ただし、データの編集や変換は行われません)。

関連タスク

34 ページの『グループ・データ項目への値の割り当て (MOVE)』

99 ページの『データ項目の結合 (STRING)』

102 ページの『データ項目の分割 (UNSTRING)』

110 ページの『データ項目の計算および置換 (INSPECT)』

194 ページの『国別グループの使用』

関連参照

GROUP-USAGE 節 (COBOL for AIX 言語解説書)

国別グループの使用

グループ・データ項目を国別グループとして定義するには、グループ・レベルで項目の GROUP-USAGE NATIONAL 節をコーディングしてください。グループは、明示的または暗黙的に USAGE NATIONAL を持つデータ項目のみを含むことができます。

以下のデータ記述項目は、レベル 01 グループとその従属グループが国別グループ項目であることを指定します。

```
01 Nat-Group-1  GROUP-USAGE NATIONAL.  
  02 Group-1.  
    04 Month    PIC 99.  
    04 DayOf    PIC 99.  
    04 Year     PIC 9999.  
  02 Group-2   GROUP-USAGE NATIONAL.  
    04 Amount  PIC 9(4).99  USAGE NATIONAL.
```

上の例で、Nat-Group-1 は国別グループであり、その従属グループ Group-1 および Group-2 も国別グループです。Group-1 に関して GROUP-USAGE NATIONAL 節が暗黙指定され、Group-1 の従属項目に関して USAGE NATIONAL が暗黙指定されています。Month、DayOf、および Year は国別 10 進数項目であり、Amount は USAGE NATIONAL を持つ数字編集項目です。

英数字グループ内の国別グループは、次の例のようにして従属させることができます。

```
01 Alpha-Group-1.  
  02 Group-1.  
    04 Month    PIC 99.  
    04 DayOf    PIC 99.  
    04 Year     PIC 9999.  
  02 Group-2   GROUP-USAGE NATIONAL.  
    04 Amount  PIC 9(4).99.
```

上の例で、Alpha-Group-1 および Group-1 は英数字グループであり、Group-1 内の従属項目に関して USAGE DISPLAY が暗黙指定されています。(Alpha-Group-1 が USAGE NATIONAL をグループ・レベルで指定した場合、Group-1 の従属項目のそれぞれについて USAGE NATIONAL が暗黙指定されることとなります。しかし、Alpha-Group-1 および Group-1 は (国別グループではなく) 英数字グループになり、移動や比較などの操作時に英数字グループと同様の振る舞いを示します。) Group-2 は国別グループであり、数字編集項目 Amount に関して USAGE NATIONAL が暗黙指定されています。

国別グループ内で英数字グループを従属させることはできません。国別グループ内の基本項目はすべて明示的または暗黙的に USAGE NATIONAL として記述されている必要があります。国別グループ内のグループ項目はすべて明示的または暗黙的に GROUP-USAGE NATIONAL として記述されている必要があります。

関連概念

192 ページの『国別グループ』

関連タスク

195 ページの『国別グループを基本項目として使用』

195 ページの『国別グループをグループ項目として使用』

関連参照

GROUP-USAGE 節 (COBOL for AIX 言語解説書)

国別グループを基本項目として使用

ほとんどの場合、国別グループは基本データ項目であるかのように使用できます。

次の例で、国別グループ項目 Group-1 は国別編集項目 Edited-date へ移動されます。Group-1 は移動時に基本データ項目として扱われるので、受信データ項目で編集が行われます。移動後の Edited-date の値は、国別文字で 06/23/2008 になります。

```
01 Edited-date PIC NN/NN/NNNN USAGE NATIONAL.
01 Group-1    GROUP-USAGE NATIONAL.
   02 Month   PIC 99   VALUE 06.
   02 DayOf   PIC 99   VALUE 23.
   02 Year    PIC 9999 VALUE 2008.
   . . .
   MOVE Group-1 to Edited-date.
```

Group-1 が代わりに英数字グループであり、その中で従属項目のそれぞれが USAGE NATIONAL を持っているとした場合 (それぞれの基本項目ごとに USAGE NATIONAL 節で明示的に指定されるか、あるいはグループ・レベルで USAGE NATIONAL 節で暗黙的に指定されている場合)、基本移動ではなくグループ移動が行われます。移動時には編集も変換も行われません。移動後の Edited-date の最初の 8 つの文字位置の値は、国別文字で 06232008 になり、残りの 2 つの文字位置の値は 4 バイトの英数字スペースになります。

関連タスク

34 ページの『グループ・データ項目への値の割り当て (MOVE)』

205 ページの『国別データ・オペランドと英数字グループ・オペランドの比較』

『国別グループをグループ項目として使用』

関連参照

MOVE ステートメント (COBOL for AIX 言語解説書)

国別グループをグループ項目として使用

国別グループを使用するようなことがある場合、それはグループ・セマンティクスで処理されます。つまり、グループ内の基本項目は認識または処理されます。

次の例で、国別グループ項目 Group-OneN に作用する INITIALIZE ステートメントにより、国別文字の値 15 はグループ内の数値項目にのみ移動されます。

```
01 Group-OneN Group-Usage National.
   05 Trans-codeN Pic N Value "A".
   05 Part-numberN Pic NN Value "XX".
   05 Trans-quantN Pic 99 Value 10.
   . . .
   Initialize Group-OneN Replacing Numeric Data By 15
```

上の Group-OneN の Trans-quantN のみが数値なので、Trans-quantN のみが値 15 を受け取ります。その他の従属項目は未変更です。

以下の表は、国別グループがグループ・セマンティクスで処理されるケースを要約したものです。

表 21. グループ・セマンティクスで処理される国別グループ項目

言語機能	国別グループ項目の使用法	説明
ADD、SUBTRACT、または MOVE ステートメントの CORRESPONDING 句	CORRESPONDING 句の規則に従って、グループとして処理する国別グループ項目を指定してください。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に処理されます。
INITIALIZE ステートメント	INITIALIZE ステートメントの規則に従って、グループとして処理する国別グループを指定してください。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に初期化されます。
名前の修飾	国別グループ項目の名前を使用して、国別グループ内の基本データ項目の名前および従属グループ項目の名前を修飾してください。	英数字グループの場合と同じ修飾の規則に従ってください。
RENAMES 節の THROUGH 句	THROUGH 句で国別グループ項目を指定するには、英数字グループ項目の場合と同じ規則を使用してください。	結果は英数字グループ項目です。
XML GENERATE ステートメントの FROM 句	XML GENERATE ステートメントの規則に従って、グループとして処理する国別グループ項目を FROM 句で指定してください。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に処理されます。

関連タスク

- 31 ページの『構造の初期化 (INITIALIZE)』
- 72 ページの『テーブルの初期化 (INITIALIZE)』
- 33 ページの『基本データ項目への値の割り当て (MOVE)』
- 34 ページの『グループ・データ項目への値の割り当て (MOVE)』
- 118 ページの『データ項目の長さの検出』
- 407 ページの『XML 出力の生成』

関連参照

- 修飾 (COBOL for AIX 言語解説書)
- RENAMES 節 (COBOL for AIX 言語解説書)

文字データの保管

以下のテーブルを使用して英数字 (DISPLAY)、DBCS (DISPLAY-1)、および Unicode (NATIONAL) のエンコード方式を比較し、ストレージの使用法について計画を立ててください。

表 22. エンコード方式と英数字、DBCS、および国別データのサイズ

特性	DISPLAY	DISPLAY-1	NATIONAL
文字エンコード・ユニット	1 バイト	2 バイト	2 バイト
コード・ページ	ASCII、EUC、UTF-8、または EBCDIC ³	ASCII DBCS または EBCDIC DBCS ³	UTF-16BE ¹

表 22. エンコード方式と英数字、DBCS、および国別データのサイズ (続き)

特性	DISPLAY	DISPLAY-1	NATIONAL
図形文字当たりのエンコード・ユニット数	1	1	1 または 2 ²
図形文字当たりのバイト数	1 バイト	2 バイト	2 または 4 バイト

1. ソース・プログラム内の国別リテラルは、実行時に UTF-16 に変換されます。

2. 大部分の文字は 1 つのエンコード・ユニットを使用して UTF-16 で表現されます。特に次の文字は、文字ごとに単一の UTF-16 エンコード・ユニットを使用して表現されません。

- COBOL 文字 A から Z、a から z、0 から 9、スペース、+ -*/= \$,,:>()> <:'
- EBCDIC、ASCII、または EUC コード・ページから変換されるすべての文字

3. ロケールに応じて、CHAR(NATIVE) または CHAR(EBCDIC) オプション、および EBCDIC_CODEPAGE 環境変数の設定

関連概念

188 ページの『Unicode および言語文字のエンコード』

関連タスク

210 ページの『文字データのコード・ページの指定』

関連参照

271 ページの『CHAR』

国別 (Unicode) 表現との変換

暗黙的または明示的にデータ項目を国別 (UTF-16) 表現に変換できます。

MOVE ステートメントを使用すれば、暗黙的に、英字、英数字、DBCS、または整数データを国別データに変換できます。暗黙変換は、英数字データ項目を USAGE NATIONAL 付きデータ項目と比較する IF ステートメントなど、他の COBOL ステートメントでも行われます。

組み込み関数 NATIONAL-OF および DISPLAY-OF をそれぞれ使用して、明示的に国別データ項目に変換したり、国別データ項目から変換したりすることができます。これらの組み込み関数を使用すると、データ項目で有効にされるコード・ページとは異なるコード・ページを変換用に指定することができます。

関連タスク

- 198 ページの『英数字、DBCS、および整数から国別への変換 (MOVE)』
- 198 ページの『英数字または DBCS から国別への変換 (NATIONAL-OF)』
- 199 ページの『国別の英数字への変換 (DISPLAY-OF)』
- 199 ページの『デフォルト・コード・ページのオーバーライド』
- 202 ページの『国別 (UTF-16) データの比較』
- 209 ページの『第 11 章 ロケールの設定』

英数字、DBCS、および整数から国別への変換 (MOVE)

MOVE ステートメントを使用して、データを国別表現に暗黙的に変換できます。

次の種類のデータをカテゴリ—国別または国別編集データ項目に移動させることができ、そのようにしてデータを国別表現に変換できます。

- 英字
- 英数字
- 英数字編集
- DBCS
- USAGE DISPLAY の整数
- USAGE DISPLAY の数字編集

同様に次の種類のデータを、USAGE NATIONAL を持つ数字編集データ項目に移動させることができます。

- 英数字
- 表示浮動小数点 (USAGE DISPLAY の浮動小数点)
- USAGE DISPLAY の数字編集
- USAGE DISPLAY の整数

国別データへの移動に関する完全な規則については、MOVE ステートメントに関する関連参照を参照してください。

例えば、以下の MOVE ステートメントは、英数字リテラル "AB" を国別データ項目 UTF16-Data に移動させます。

```
01 UTF16-Data Pic N(2) Usage National.  
   . . .  
   Move "AB" to UTF16-Data
```

上記の MOVE ステートメントの実行後、UTF16-Data には、英数字「AB」の国別表現である NX'00410042' が入ります。

USAGE NATIONAL を持つ受信データ項目で埋め込みが必要な場合、デフォルトの UTF-16 スペース文字 (NX'0020') が使用されます。切り捨てが必要な場合、それは国別文字位置の境界で行われます。

関連タスク

- 33 ページの『基本データ項目への値の割り当て (MOVE)』
- 34 ページの『グループ・データ項目への値の割り当て (MOVE)』
- 43 ページの『数値データの表示』
- 205 ページの『DBCS サポート用のコーディング』

関連参照

MOVE ステートメント (*COBOL for AIX 言語解説書*)

英数字または DBCS から国別への変換 (NATIONAL-OF)

英字、英数字、または DBCS データを国別データ項目に変換するには、NATIONAL-OF 組み込み関数を使用してください。データ項目に対して有効になって

いるコード・ページとは異なるコード・ページでソースがエンコードされている場合、ソース・コード・ページを 2 番目の引数として指定してください。

200 ページの『例: 国別データとの変換』

関連タスク

201 ページの『UTF-8 データの処理』

201 ページの『中国語 GB 18030 データの処理』

208 ページの『DBCS データを含む英数字データ項目の処理』

関連参照

NATIONAL-OF (*COBOL for AIX* 言語解説書)

国別の英数字への変換 (DISPLAY-OF)

2 番目の引数として指定されたコード・ページで表現される英数字 (USAGE DISPLAY) 文字ストリングへ国別データを変換するには、DISPLAY-OF 組み込み関数を使用してください。

2 番目の引数を省略した場合、出力のコード・ページは、実行時のロケールによって決定されます。

1 バイト文字セット (SBCS) 文字と DBCS 文字を結合した EBCDIC または ASCII コード・ページを指定すると、戻されるストリングは SBCS 文字と DBCS 文字の混合になることがあります。関数で有効なコード・ページが EBCDIC コード・ページである場合、DBCS サブストリングはシフトイン文字とシフトアウト文字で区切られています。

200 ページの『例: 国別データとの変換』

関連概念

209 ページの『アクティブ・ロケール』

関連タスク

201 ページの『UTF-8 データの処理』

201 ページの『中国語 GB 18030 データの処理』

関連参照

DISPLAY-OF (*COBOL for AIX* 言語解説書)

デフォルト・コード・ページのオーバーライド

状況によっては、実行時に有効なコード・ページとは異なるコード・ページにデータを変換しなければならない場合や、そうしたコード・ページからデータを変換しなければならない場合があります。そうするには、コード・ページを明示的に指定した変換関数を使用して項目を変換します。

DISPLAY-OF 組み込み関数の引数としてコード・ページを指定した場合に、そのコード・ページが、実行時に有効なコード・ページとは異なる場合、暗黙的変換を伴う

操作 (国別データ項目への割り当てまたは国別データ項目との比較など) で、関数結果を使用しないでください。このような操作は、実行時のコード・ページが使用されることを想定しています。

例: 国別データとの間の変換

次の例は、国別 (UTF-16) データ項目との間での変換に、NATIONAL-OF および DISPLAY-OF 組み込み関数ならびに MOVE ステートメントを使用する方法を示しています。また、複数のコード・ページでエンコードされたストリングに対して操作を行うときの明示的変換の必要性も示しています。

```
* . . .
01 Data-in-Unicode      pic N(100) usage national.
01 Data-in-Greek       pic X(100).
01 other-data-in-US-English pic X(12) value "PRICE in $ =".
* . . .
  Read Greek-file into Data-in-Greek
  Move function National-of(Data-in-Greek, "ISO8859-7")
    to Data-in-Unicode
* . . . process Data-in-Unicode here . . .
  Move function Display-of(Data-in-Unicode, "ISO8859-7")
    to Data-in-Greek
  Write Greek-record from Data-in-Greek
```

上記の例は、入力コード・ページが指定されているので正しく機能します。Data-in-Greek は、ISO8859-7 (Ascii ギリシャ語) で表されるデータとして変換されます。しかし、以下のステートメントの場合、項目内の文字すべてが、たまたまギリシャ語と英語の両方のコード・ページで表現が同じであるものでなければ、変換が誤ったものになります。

```
Move Data-in-Greek to Data-in-Unicode
```

有効なロケールが en_US.ISO8859-1 であるとする、上記の MOVE ステートメントは、コード・ページ ISO8859-1 から UTF-16BE への変換に基づいて、Data-in-Greek を Unicode に変換します。Data-in-Greek は ISO8859-7 でエンコードされるので、この変換では期待される結果が得られません。

ロケールを el_GR.ISO8859-7 に設定した場合は (つまり、プログラムが ASCII データをギリシャ語で処理する場合は)、同じ例を次のようにコーディングすることができます。

```
* . . .
01 Data-in-Unicode pic N(100) usage national.
01 Data-in-Greek  pic X(100).
* . . .
  Read Greek-file into Data-in-Greek
* . . . process Data-in-Greek here ...
* . . . or do the following (if need to process data in Unicode):
  Move Data-in-Greek to Data-in-Unicode
* . . . process Data-in-Unicode
  Move function Display-of(Data-in-Unicode) to Data-in-Greek
  Write Greek-record from Data-in-Greek
```

関連タスク

209 ページの『第 11 章 ロケールの設定』

UTF-8 データの処理

UTF-8 データを処理する必要がある場合は、最初に UTF-8 データを国別データ項目の UTF-16 に変換します。国別データを処理したあとで、データを出力のために再び UTF-8 に変換します。この変換には、それぞれ、組み込み関数 NATIONAL-OF および DISPLAY-OF を使用します。UTF-8 データにはコード・ページ 1208 を使用します。

ASCII または EBCDIC データを UTF-8 に変換するには、次の 2 つのステップを実行する必要があります (有効化されているロケールのコード・ページが、ネイティブ英数字データが既に UTF-8 でエンコードされている UTF-8 ではない場合)。

1. 関数 NATIONAL-OF を使用して、ASCII または EBCDIC ストリングを国別ストリング (UTF-16) に変換します。
2. 関数 DISPLAY-OF を使用して、国別ストリングを UTF-8 に変換します。

次の例は、ギリシャ語の EBCDIC データを UTF-8 に変換しています。

```
01 Greek-EBCDIC pic X(10) value "αβγδεζηθ".
01 UnicodeString pic N(10).
01 UTF-8-String pic X(20).
   Move function National-of(Greek-EBCDIC, 00875) to UnicodeString
   Move function Display-of(UnicodeString, 01208) to UTF-8-String
```

| **使用上の注意:** 参照変更を使用して UTF-8 でエンコードされたデータを参照する場
| 合には注意してください。UTF-8 文字のエンコードでは、1 文字に使用されるバイ
| ト数が異なります。マルチバイト文字を分割する可能性がある処理は避けてくださ
| い。

関連タスク

106 ページの『データ項目のサブストリングの参照』

197 ページの『国別 (Unicode) 表現との間の変換』

400 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

中国語 GB 18030 データの処理

GB 18030 は、中華人民共和国の政府機関によって指定された国別文字標準です。

COBOL for AIX は、GB 18030 をサポートしています。有効化されたロケールで指定されたコード・ページが GB18030 (GB 18030 をサポートするコード・ページ) である場合、GB18030 でエンコードされた、GB 18030 文字を含む USAGE DISPLAY データ項目は、プログラムで処理できます。GB 18030 文字は、それぞれ 1 から 4 バイトです。そのため、プログラム・ロジックでは、データのマルチバイトの性質を意識する必要があります。

以下の方法で、GB 18030 文字を処理できます。

- 国別データ項目を使用して、UTF-16、CCSID 01200 で表される GB 18030 文字を定義および処理します。

- データを UTF-16 へ変換し、その UTF-16 データを処理した後にデータを元のコード・ページ表現へ逆変換することにより、任意のコード・ページ (GB18030 (CCSID 1392) を含む) のデータを処理します。

変換を必要とする中国語 GB 18030 を処理する必要がある場合、まず入力データを国別データ項目の UTF-16 に変換してください。国別データ項目を処理した後、出力用としてそれを中国語 GB 18030 に逆変換してください。この変換には、組み込み関数 NATIONAL-OF および DISPLAY-OF を使用し、GB18030 または 1392 を各関数の 2 番目の引数として指定します。

次の例は、これらの変換を示しています。

```
01 Chinese-ASCII pic X(16) value "奥林匹克运动会".
01 Chinese-GB18030-String pic X(16).
01 UnicodeString pic N(14).
. . .
Move function National-of(Chinese-ASCII, 1392) to UnicodeString
* Process data in Unicode
Move function Display-of(UnicodeString, 1392) to Chinese-GB18030-String
```

関連タスク

- 197 ページの『国別 (Unicode) 表現との間の変換』
- 205 ページの『DBCS サポート用のコーディング』

関連参照

- 196 ページの『文字データの保管』

国別 (UTF-16) データの比較

国別 (UTF-16) データ、すなわち USAGE NATIONAL を持つデータ項目 (クラス国別かクラス数値かにかかわらず) および国別リテラルを、比較条件の他の種類のデータと明示的または暗黙的に比較することができます。

以下のステートメントで、国別データを使用する条件式をコード化できます。

- EVALUATE
- IF
- INSPECT
- PERFORM
- SEARCH
- STRING
- UNSTRING

続く各節では、国別データとその他のデータ項目との比較について概説します。詳細については、関連参照を参照してください。

関連タスク

- 203 ページの『2 つのクラス国別オペランドの比較』
- 204 ページの『クラス国別オペランドとクラス数値オペランドの比較』
- 204 ページの『国別数値オペランドと他の数値オペランドの比較』

204 ページの『国別と他の文字ストリング・オペランドとの比較』

205 ページの『国別データ・オペランドと英数字グループ・オペランドの比較』

関連参照

関連条件 (COBOL for AIX 言語解説書)

一般比較条件 (COBOL for AIX 言語解説書)

国別比較 (COBOL for AIX 言語解説書)

グループ比較 (COBOL for AIX 言語解説書)

2 つのクラス国別オペランドの比較

クラス国別の 2 つのオペランドの文字値を比較できます。

一方 (または両方) のオペランドは、次の項目タイプのいずれかにすることができます。

- 国別グループ
- カテゴリ国別または国別編集の基本データ項目
- USAGE NATIONAL を持つ数字編集データ項目

オペランドの 1 つは、代わりに国別リテラルまたは国別組み込み関数にすることができます。

以下のように、NCOLLSEQ コンパイラー・オプションを使用して、実行する比較のタイプを決定します。

NCOLLSEQ(BINARY)

同じ長さの 2 つのクラス国別オペランドを比較する場合、対応する文字のすべてのペアが等しいならば、それらは等しいと判別されます。対応する文字の対に等しくないものがある場合は、等しくない最初の文字のペアの 2 進値を比較することによって、より大きい 2 進値を持つオペランドが判別されます。

長さの異なるオペランドを比較する場合、短いほうのオペランドは、長いほうのオペランドの長さの位置までその右側にデフォルトの UTF-16 スペース文字 (NX'0020') が埋め込まれているものとして扱われます。

NCOLLSEQ(LOCALE)

ロケール・ベースの比較を使用した場合には、オペランドは、有効なロケールに関連付けられた照合順序のアルゴリズムを使用して、比較されます。すべてスペースから成るオペランドがシングル・スペースにまで切り詰められる場合を除いて、末尾スペースはオペランドから切り詰められます。

長さの異なるオペランドを比較する場合、短いほうのオペランドは、スペースを使用して延長されません。これは、そのような延長によって、ロケールの期待される結果が変化してしまうことがあるからです。

PROGRAM COLLATING SEQUENCE 節は、2 つのクラス国別オペランドの比較には影響しません。

関連概念

192 ページの『国別グループ』

関連タスク

194 ページの『国別グループの使用』

関連参照

296 ページの『NCOLLSEQ』

国別比較 (COBOL for AIX 言語解説書)

クラス国別オペランドとクラス数値オペランドの比較

国別リテラルまたはクラス国別データ項目を、整数リテラルまたは整数として定義された数値データ項目 (すなわち、国別 10 進数項目またはゾーン 10 進数項目) と比較できます。リテラルにできるのは多くても 1 つのオペランドです。

国別リテラルまたはクラス国別データ項目を、浮動小数点データ項目 (すなわち、表示浮動小数点または国別浮動小数点項目) と比較することもできます。

数値オペランドは、まだ国別表現でない場合には、国別 (UTF-16) 表現に変換されます。オペランドの国別文字値が比較されます。

関連参照

一般比較条件 (COBOL for AIX 言語解説書)

国別数値オペランドと他の数値オペランドの比較

国別数値オペランド (国別 10 進数オペランドおよび国別浮動小数点オペランド) は、USAGE NATIONAL を持つクラス数値のデータ項目です。

USAGE とは無関係に、数値オペランドの代数値を比較できます。そのため、国別 10 進数項目または国別浮動小数点項目を、バイナリー項目、内部 10 進数項目、ゾーン 10 進数項目、表示浮動小数点項目、または他の任意の数値項目と比較できます。

関連タスク

192 ページの『国別数値データ項目の定義』

関連参照

一般比較条件 (COBOL for AIX 言語解説書)

国別と他の文字ストリング・オペランドとの比較

国別リテラルまたはクラス国別データ項目の文字値を、以下の他の文字ストリング・オペランド (USAGE DISPLAY の英字、英数字、英数字編集、DBCS、または数字編集) のいずれかの文字値と比較できます。

これらのオペランドは、基本国別データ項目へ移動されたかのように扱われます。文字は国別 (UTF-16) 表現へ変換され、2 つの国別文字オペランドの比較が進行します。

関連タスク

191 ページの『国別文字形象定数の使用』

207 ページの『DBCS リテラルの比較』

関連参照

国別比較 (COBOL for AIX 言語解説書)

国別データ・オペランドと英数字グループ・オペランドの比較

国別リテラル、国別グループ項目、または USAGE NATIONAL を持つ任意の基本データ項目を、英数字グループと比較できます。

どちらのオペランドも変換されません。国別オペランドは、国別オペランドと同じサイズ (バイト単位) の英数字グループ項目に移動されたかのように扱われ、2 つのグループが比較されます。英数字比較は、英数字グループ・オペランドの従属項目の表現とは無関係に行われます。

例えば、Group-XN は、USAGE NATIONAL を持つ 2 つの従属項目からなる英数字グループです。

```
01 Group-XN.  
   02 TransCode PIC NN   Value "AB"  Usage National.  
   02 Quantity  PIC 999  Value 123   Usage National.  
   . . .  
   If N"AB123" = Group-XN Then Display "EQUAL"  
   Else Display "NOT EQUAL".
```

上記の IF ステートメントが実行されると、国別リテラル N"AB123" の 10 バイトが、バイトごとに Group-XN の内容と比較されます。項目は比較されて同じと見なされると、「EQUAL」が表示されます。

関連参照

グループ比較 (COBOL for AIX 言語解説書)

DBCS サポート用のコーディング

IBM COBOL for AIX は、2 バイト文字セット (DBCS) を使用する言語を含む多数の各国語のいずれでもアプリケーションの使用をサポートします。

以下のリストは、DBCS のサポートを要約したものです。

- ユーザー定義語での DBCS 文字 (マルチバイト名)
- コメントでの DBCS 文字
- DBCS データ項目 (PICTURE N、G、または G と B で定義します)
- DBCS リテラル
- 照合シーケンス
- SOSI コンパイラー・オプション

関連タスク

- 206 ページの『DBCS データの宣言』
- 206 ページの『DBCS リテラルの使用』
- 207 ページの『有効な DBCS 文字に関するテスト』
- 208 ページの『DBCS データを含む英数字データ項目の処理』
- 209 ページの『第 11 章 ロケールの設定』
- 216 ページの『ロケール付きの照合シーケンスの制御』

関連参照

305 ページの『SOSI』

DBCS データの宣言

DBCS データ項目を宣言するには、PICTURE および USAGE 節を使用してください。DBCS データ項目では、PICTURE 記号の G、G と B、または N を使用できます。

DBCS データ項目は、USAGE DISPLAY-1 節を使用して指定できます。PICTURE 記号の G を使用する場合、USAGE DISPLAY-1 を指定する必要があります。PICTURE 記号の N を指定したが、USAGE 節を省略した場合、NSYMBOL コンパイラー・オプションの設定に応じて USAGE DISPLAY-1 または USAGE NATIONAL が暗黙指定されます。

DBCS 項目の宣言で USAGE 節と一緒に VALUE 節を使用する場合、DBCS リテラルまたは形象定数 SPACE または SPACES を指定する必要があります。

データ項目に USAGE DISPLAY-1 が (明示的または暗黙的に) 存在する場合、選択されたロケールは、DBCS 文字を含んだコード・ページを示す必要があります。ロケールのコード・ページに DBCS 文字が含まれない場合には、そのデータ項目はエラーとしてフラグを立てられます。

参照変更の処理の目的のため、DBCS データ項目のそれぞれの文字は、コード・ページ幅に相当するバイト数 (つまり、2) を占有するとみなされます。

関連タスク

209 ページの『第 11 章 ロケールの設定』

関連参照

213 ページの『サポートされるロケールおよびコード・ページ』

296 ページの『NSYMBOL』

DBCS リテラルの使用

DBCS リテラルを表すには、接頭部 N または G を使用できます。

すなわち、次のいずれかの方法で DBCS リテラルを指定できます。

- N'DBCS 文字' (コンパイラー・オプション NSYMBOL(DBCS) が有効である場合)
- G'*dbcs characters*'

APOST または QUOTE コンパイラー・オプションの設定にかかわらず、引用符 (") または単一引用符 (') を DBCS リテラルの区切り文字として使用できます。

DBCS リテラルに対して、同じ開始区切り文字と終了区切り文字をコーディングする必要があります。

SOSI コンパイラー・オプションが有効化されている場合、シフトアウト (SO) 制御文字 X'1E' は、開始区切り文字の直後に続けなければなりません。シフトイン (SI) 制御文字 X'1F' は終了区切り文字の直前に来るようにする必要があります。

DBCS リテラルのほかにも、英数字リテラルを使用して、サポートされるコード・ページの 1 つの任意の文字を指定できます。ただし、SOSI コンパイラー・オプション

ンが有効化されている場合には、英数字リテラルの中にある DBCS 文字のストリングは、SO および SI 文字によって区切る必要があります。

マルチバイト文字を含んでいる英数字リテラルを継続させることはできません。さらに DBCS リテラルの長さは、B 領域の単一ソース行で使用可能なスペースによって限定されます。したがって、DBCS リテラルの最大長は 28 個の 2 バイト文字です。

マルチバイト文字を含んでいる英数字リテラルはバイトごとに、すなわち 1 バイト文字に適したセマンティクスによって、処理されます。ただし、(例えば、国別データ項目への割り当てや国別データ項目との比較のように) 明示的または暗黙的に国別データ表現に変換された場合は、そのような仕方では処理されません。

関連タスク

『DBCS リテラルの比較』
27 ページの『形象定数の使用』

関連参照

296 ページの『NSYMBOL』
301 ページの『QUOTE/APOST』
305 ページの『SOSI』
DBCS リテラル (COBOL for AIX 言語解説書)

DBCS リテラルの比較

DBCS リテラルの比較は、コンパイル時のロケールに基づきます。このため、対象とする実行時のロケールがコンパイル時のロケールと同じでない限り、2 つの DBCS リテラル間の暗黙的な関係条件を表すステートメント内で (VALUE G'*literal-1*' THRU G'*literal-2*' など) DBCS リテラルを使用することは避けてください。

関連タスク

202 ページの『国別 (UTF-16) データの比較』
209 ページの『第 11 章 ロケールの設定』

関連参照

274 ページの『COLLSEQ』
DBCS リテラル (COBOL for AIX 言語解説書)
DBCS 比較 (COBOL for AIX 言語解説書)

有効な DBCS 文字に関するテスト

漢字クラス・テストでは、有効な日本語図形文字に関するテストが行われます。このテストには、カタカナ、ひらがな、ローマ字、および漢字の文字セットが含まれます。

漢字および DBCS クラス・テストは、zSeries 定義と整合するように定義されます。両方のクラス・テストは、2 バイト文字を z/OS に定義された 2 バイト文字に変換して、内部で実行されます。変換された 2 バイト文字は、DBCS および日本語 GRAPHIC 文字がないかテストされます。

漢字クラス・テストは、最初のバイトの X'41' から X'7E' および 2 番目のバイトの X'41' から X'FE' の範囲、さらにスペース文字 X'4040' について、変換された文字を検査することで行われます。

DBCS クラス・テストでは、コード・ページの有効な図形文字に関するテストが行われます。

DBCS クラス・テストは、それぞれの文字の最初と 2 番目のバイト双方の X'41' から X'FE' の範囲、およびスペース文字 X'4040' について、変換された文字を検査することで行われます。

関連タスク

90 ページの『条件式のコーディング』

関連参照

クラス条件 (COBOL for AIX 言語解説書)

DBCS データを含む英数字データ項目の処理

DBCS 文字を含んでいる英数字データ項目に対してバイト指向の操作 (例えば、STRING、UNSTRING、または参照変更) を行うと、結果は予測不能です。そうではなく項目を国別データ項目に変換してから、処理する必要があります。

すなわち、以下のステップを実行してください。

1. MOVE ステートメントまたは NATIONAL-OF 組み込み関数を使用して、項目を国別データ項目の UTF-16 に変換します。
2. 必要に応じて国別データ項目を処理します。
3. DISPLAY-OF 組み込み関数を使用して、結果を英数字データ項目に逆変換します。

関連タスク

99 ページの『データ項目の結合 (STRING)』

102 ページの『データ項目の分割 (UNSTRING)』

106 ページの『データ項目のサブストリングの参照』

197 ページの『国別 (Unicode) 表現との間の変換』

第 11 章 ロケールの設定

アプリケーションの実行時に有効になるロケールの国/地域別情報を反映させるようにアプリケーションを記述することができます。国/地域別情報には、ソート順、文字種別、各国語、さらには日付と時刻、数値、通貨、住所、および電話番号の形式が含まれます。

COBOL for AIX では、適切なコード・ページや照合シーケンスを選択したり、言語エレメントやコンパイラ・オプションを使用して、Unicode、1 バイト文字セット、および 2 バイト文字セット (DBCS) を処理することができます。

関連概念

『アクティブ・ロケール』

関連タスク

210 ページの『文字データのコード・ページの指定』

211 ページの『環境変数を使用したロケールの指定』

216 ページの『ロケール付きの照合シーケンスの制御』

220 ページの『アクティブ・ロケールおよびコード・ページ値へのアクセス』

アクティブ・ロケール

ロケール とは、国/地域別環境に関する情報をエンコードするデータの集合をいいます。アクティブ・ロケールとは、プログラムをコンパイルまたは実行するときに有効なロケールです。アプリケーションの国/地域別環境を設定するには、アクティブ・ロケールを指定します。

一度にアクティブにできるロケールは 1 つだけです。

アクティブ・ロケールは、プログラム全体を通して国/地域別依存のインターフェースの動作に影響します。

- 文字データに使用されるコード・ページ
- メッセージ
- 照合シーケンス
- 日時形式
- 文字の種別および大/小文字の変換

アクティブ・ロケールは、次の項目には影響しません。これらについては、標準 COBOL 85 で特定の言語および動作が定義されています。

- 小数点およびグループ化された分離文字
- 通貨記号

アクティブ・ロケールは、プログラムをコンパイルおよび実行するためのコード・ページを決定します。

- コンパイルに使用されるコード・ページはコンパイル時のロケール設定に基づきます。

- アプリケーションの実行に使用されるコード・ページは実行時のロケール設定に基づきます。

ソース・プログラム内のリテラル値の評価は、コンパイル時にアクティブなロケールを使用して処理されます。例えば、プログラムを実行するために国別リテラルをソース表現から UTF-16 に変換する処理では、コンパイル時のロケールを使用します。

COBOL for AIX は、該当する環境変数とシステム設定の組み合わせからアクティブ・ロケールの設定を決定します。まず最初に、環境変数を使用されます。該当するロケールのカテゴリが環境変数によって定義されていない場合、COBOL ではデフォルトとシステム設定を使用します。

関連概念

212 ページの『システム設定からのロケールの決定』

関連タスク

『文字データのコード・ページの指定』

211 ページの『環境変数を使用したロケールの指定』

216 ページの『ロケール付きの照合シーケンスの制御』

関連参照

213 ページの『変換が使用可能なメッセージのタイプ』

文字データのコード・ページの指定

ソース・プログラムでは、COBOL の名前、リテラル、およびコメント内では、サポートされるコード・ページで表現される文字を使用できます。ユーザー定義語の構造の詳細については、以下の関連参照で引用されている、「*COBOL for AIX 言語解説書 言語解説書*」の情報を参照してください。)

実行時に、USAGE DISPLAY、USAGE DISPLAY-1、または USAGE NATIONAL で記述されているデータ項目内では、サポートされるコード・ページで表現される文字を使用することができます。

特定の英数字データ項目に対して有効となるコード・ページは、以下の側面によって決まります。

- 使用される USAGE 文節
- NATIVE 句を USAGE 文節と共に使用したかどうか
- CHAR(NATIVE) または CHAR(EBCDIC) コンパイラー・オプションを使用したかどうか
- EBCDIC_CODEPAGE 環境変数の値
- アクティブなロケール

USAGE NATIONAL データ項目の場合、コード・ページは必ずビッグ・エンディアン形式の UTF-16 です。

USAGE DISPLAY データ項目の場合、COBOL for AIX は以下のように ASCII、UTF-8、EUC、 および EBCDIC の各コード・ページからコード・ページを選択します。

- USAGE 文節の NATIVE 句で記述されたデータ項目、または CHAR(NATIVE) オプションを有効にしてコンパイルされたデータ項目は、ASCII コード・ページ、EUC コード・ページ、または UTF-8 コード・ページでエンコードされます。
- USAGE 文節の NATIVE 句を使用せずに記述されたデータ項目のうち、CHAR(EBCDIC) オプションを有効にしてコンパイルされたデータ項目は、EBCDIC コード・ページでエンコードされます。

USAGE DISPLAY-1 データ項目の場合、COBOL for AIX は以下のように ASCII コード・ページまたは EBCDIC コード・ページを選択します。

- USAGE 文節の NATIVE 句で記述されたデータ項目、または CHAR(NATIVE) オプションを有効にしてコンパイルされたデータ項目は、ASCII DBCS コード・ページでエンコードされます。
- USAGE 文節の NATIVE 句を使用せずに記述されたデータ項目のうち、CHAR(EBCDIC) オプションを有効にしてコンパイルされたデータ項目は、EBCDIC DBCS コード・ページでエンコードされます。

COBOL では、適切なコード・ページを次のように決定します。

ASCII, UTF-8, EUC

実行時のアクティブ・ロケールから

EBCDIC

EBCDIC_CODEPAGE 環境変数が設定されている場合は、この変数から決定される。それ以外の場合は、現行のロケール設定からデフォルトの EBCDIC コード・ページが決定される。

関連タスク

『環境変数を使用したロケールの指定』

関連参照

213 ページの『サポートされるロケールおよびコード・ページ』

228 ページの『ランタイム環境変数』

271 ページの『CHAR』

1 バイト文字の COBOL ワード (COBOL for AIX 言語解説書)

マルチバイト文字のユーザー定義語 (COBOL for AIX 言語解説書)

環境変数を使用したロケールの指定

COBOL プログラムのロケール情報を提供するには、いずれかの環境変数を使用します。

すべてのロケール・カテゴリー (メッセージ、照合シーケンス、日付と時刻形式、文字種別、および大/小文字の変換) に使用するコード・ページを指定するには、LC_ALL を使用します。

特定のロケール・カテゴリーの値を設定するには、以下の該当する環境変数を使用します。

- LC_MESSAGES は、肯定応答および否定応答の形式を指定するために使用します。また、これを使用して、メッセージ (エラー・メッセージやリスト・ヘッダーなど) が米国英語か日本語のどちらになるかを決定できます。日本語以外のロケールの場合は、米国英語が使用されます。
- LC_COLLATE は、関連条件内や SORT および MERGE ステートメント内などの、より大比較またはより小比較に有効な照合シーケンスを指定するために使用します。
- LC_TIME は、コンパイラ・リストに示される日付と時刻の形式を指定するために使用します。それ以外の日付および時刻の値はすべて COBOL 言語の構文に従います。
- LC_CTYPE は、文字の種別、大/小文字の変換、およびその他の文字属性を指定するために使用します。

上記のロケール環境変数のいずれかで指定されないロケール・カテゴリーは、LANG 環境変数の値から設定されます。

ロケールの環境変数を設定するには、次のような形式を使用します (*.codepageID* はオプションです)。

```
SET LC_xxxx=ll_CC.codepageID
```

ここで、LC_xxxx はロケール・カテゴリーの名前、ll は小文字の 2 文字の言語コード、CC は大文字の 2 文字の ISO 国別コード、および *codepageID* はネイティブ DISPLAY と DISPLAY-1 データに使用されるコード・ページです。COBOL for AIX では、POSIX で定められたロケール規則を使用しています。

例えば、ロケールを ISO8859-1 でエンコードされるカナダ・フランス語に設定するには、COBOL アプリケーションをコンパイルして実行するコマンド・ウィンドウで、次のコマンドを発行します。

```
export LC_ALL=fr_CA.IS08859-1
```

ロケール名の有効な値 (ll_CC) と、指定するコード・ページ (*codepageID*) はロケール名に有効でなければなりません。有効な値は、後述のロケールおよびコード・ページの表に示されています。

関連概念

『システム設定からのロケールの決定』

関連タスク

210 ページの『文字データのコード・ページの指定』

関連参照

213 ページの『サポートされるロケールおよびコード・ページ』

228 ページの『ランタイム環境変数』

システム設定からのロケールの決定

COBOL for AIX が該当するロケール・カテゴリーの値を環境変数から決定できない場合は、次のデフォルト設定が使用されます。

言語および国別コードは環境変数から決定されるが、コード・ページは決定されないという場合、COBOL for AIX では、言語と国別コードの組み合わせに対するデフォルト・システム・コード・ページが使用されます。

言語と国別コードの組み合わせに複数のコード・ページが適用される場合があります。AIX システムでデフォルトのコード・ページを選択したくない場合は、コード・ページを明示的に指定する必要があります。

UTF-8: UTF-8 エンコードは、あらゆる言語と国別コードの組み合わせに対応しています。

関連タスク

- 210 ページの『文字データのコード・ページの指定』
- 211 ページの『環境変数を使用したロケールの指定』
- 220 ページの『アクティブ・ロケールおよびコード・ページ値へのアクセス』
- 225 ページの『環境変数の設定』

関連参照

- 『サポートされるロケールおよびコード・ページ』
- 228 ページの『ランタイム環境変数』

変換が使用可能なメッセージのタイプ

以下のメッセージは各国語サポート、すなわちコンパイラー、ランタイム、およびデバッガー・ユーザー・インターフェース・メッセージ、およびリスト・ヘッダー(ロケール・ベースの日付と時刻形式を含む)に対応しています。

アクティブ・ロケールで指定された該当するテキストおよび形式が、これらのメッセージおよびリスト・ヘッダーに使用されます。

メッセージの言語とロケールに影響する LANG および NLSPATH 環境変数については、下の『関連参照』を参照してください。

関連概念

- 209 ページの『アクティブ・ロケール』

関連タスク

- 211 ページの『環境変数を使用したロケールの指定』

関連参照

- 228 ページの『ランタイム環境変数』

サポートされるロケールおよびコード・ページ

次の表に、ご使用のシステムで使用可能なロケールと、各ロケールでサポートされるコード・ページを示します。COBOL for AIX は、システム上でコンパイル時と実行時に使用できるロケールをサポートしています。

システム上で使用可能なロケールは、通常は /usr/lib/nls/loc に置かれています。

表 23. サポートされるロケールおよびコード・ページ

ロケール名 ¹	言語 ²	国または地域 ³	ASCII ベースのコード・ページ ⁴	EBCDIC コード・ページ ⁵	言語グループ
any			UTF-8	ロケールに適用可能な EBCDIC コード・ページは、ロケールのコード・ページ値にかかわらず、ロケール名 <i>language</i> および <i>COUNTRY</i> 部に基づきます。	
ar_AA	アラビア語	アラビア諸国	ISO8859-6	IBM-16804、IBM-420	アラビア語
be_BY	ベロルシア語	ベラルーシ	ISO8859-5	IBM-1025、IBM-1154	Latin 5
bg_BG	ブルガリア語	ブルガリア	ISO8859-5	IBM-1025、IBM-1154	Latin 5
ca_ES	カタロニア語	スペイン	ISO8859-1、IBM-850、8859-15	IBM-285、IBM-1145	Latin 1
cs_CZ	チェコ語	チェコ共和国	ISO8859-2	IBM-870、IBM-1153	Latin 2
da_DK	デンマーク語	デンマーク	ISO8859-1、IBM-850、8859-15	IBM-277、IBM-1142	Latin 1
de_CH	ドイツ語	スイス	ISO8859-1、IBM-850、8859-15	IBM-500、IBM-1148	Latin 1
de_DE	ドイツ語	ドイツ	ISO8859-1、IBM-850、8859-15	IBM-273、IBM-1141	Latin 1
el_GR	ギリシャ語	ギリシャ	ISO8859-7	IBM-4971、IBM-875	ギリシャ語
en_AU	英語	オーストラリア	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
en_BE	英語	ベルギー	ISO8859-1、IBM-850、8859-15	IBM-500、IBM-1148	Latin 1
en_GB	英語	英国	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
en_JP	英語	日本	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
jp_Jp	英語	米国	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
en_ZA	英語	南アフリカ	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
es_ES	スペイン語	スペイン	ISO8859-1、IBM-850、8859-15	IBM-284、IBM-1145	Latin 1
fi_FI	フィンランド語	フィンランド	ISO8859-1、IBM-850、8859-15	IBM-278、IBM-1143	Latin 1
fr_BE	フランス語	ベルギー	ISO8859-1、IBM-850、8859-15	IBM-297、IBM-1148	Latin 1
fr_CA	フランス語	カナダ	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
fr_CH	フランス語	スイス	ISO8859-1、IBM-850、8859-15	IBM-500、IBM-1148	Latin 1

表 23. サポートされるロケールおよびコード・ページ (続き)

ロケール名 ¹	言語 ²	国または地域 ³	ASCII ベースのコード・ページ ⁴	EBCDIC コード・ページ ⁵	言語グループ
fr_FR	フランス語	フランス	ISO8859-1、IBM-850、8859-15	IBM-297、IBM-1148	Latin 1
hr_HR	クロアチア語	クロアチア	ISO8859-2	IBM-870、IBM-1153	Latin 2
hu_HU	ハンガリー語	ハンガリー	ISO8859-2	IBM-870、IBM-1153	Latin 2
is_IS	アイスランド語	アイスランド	ISO8859-1、IBM-850、8859-15	IBM-871、IBM-1149	Latin 1
it_CH	イタリア語	スイス	ISO8859-1、IBM-850、8859-15	IBM-500、IBM-1148	Latin 1
it_IT	イタリア語	イタリア	ISO8859-1、IBM-850、8859-15	IBM-280、IBM-1144	Latin 1
iw_IL	ヘブライ語	イスラエル国	ISO8859-8	IBM-12712、IBM-424	ヘブライ語
Iw_IL	ヘブライ語	イスラエル国	IBM-856	IBM-12712、IBM-424	ヘブライ語
ja_JP	日本語	日本	IBM-eucJP	IBM-930、IBM-939、IBM-1390、IBM-1399	表意文字言語
Ja_JP	日本語	日本	IBM-943、IBM-932	IBM-930、IBM-939、IBM-1390、IBM-1399	表意文字言語
ko_KR	韓国語	大韓民国	IBM-eucKR	IBM-933、IBM-1364	表意文字言語
lt_LT	リトアニア語	リトアニア	IBM-1257	n/a	リトアニア語
lv_LV	ラトビア語	ラトビア	IBM-1257	n/a	ラトビア語
mk_MK	マケドニア語	マケドニア	ISO8859-5	IBM-1025、IBM-1154	Latin 5
nl_BE	オランダ語	ベルギー	ISO8859-1、IBM-850、8859-15	IBM-500、IBM-1148	Latin 1
nl_NL	オランダ語	オランダ	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
no_NO	ノルウェー語	ノルウェー	ISO8859-1、IBM-850、8859-15	IBM-277、IBM-1142	Latin 1
pl_PL	ポーランド語	ポーランド	ISO8859-2	IBM-870、IBM-1153	Latin 2
pt_BR	ポルトガル語	ブラジル	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
pt_PT	ポルトガル語	ポルトガル	ISO8859-1、IBM-850、8859-15	IBM-037、IBM-1140	Latin 1
ro_RO	ルーマニア語	ルーマニア	ISO8859-2	IBM-870、IBM-1153	Latin 2
ru_RU	ロシア語	ロシア連邦	ISO8859-5	IBM-1025、IBM-1154	Latin 5
sh_SP	セルビア語 (ラテン語)	セルビア	ISO8859-2	IBM-870、IBM-1153	Latin 2
sk_SK	スロバキア語	スロバキア	ISO8859-2	IBM-870、IBM-1153	Latin 2
sl_SI	スロベニア語	スロベニア	ISO8859-2	IBM-870、IBM-1153	Latin 2
sq_AL	アルバニア語	アルバニア	ISO8859-1、IBM-850、8859-15	IBM-500、IBM-1148	Latin 1
sv_SE	スウェーデン語	スウェーデン	ISO8859-1、IBM-850、8859-15	IBM-278、IBM-1143	Latin 1
th_TH	タイ語	タイ	TIS-620	IBM-9030	タイ語

表 23. サポートされるロケールおよびコード・ページ (続き)

ロケール名 ¹	言語 ²	国または地域 ³	ASCII ベースのコード・ページ ⁴	EBCDIC コード・ページ ⁵	言語グループ
tr_TR	トルコ語	トルコ	ISO8859-9	IBM-1026、IBM-1155	トルコ語
uk_UA	ウクライナ語	ウクライナ	ISO8859-5、IBM-1124	IBM-1123、IBM-1154	Latin 5
zh_CN	中国語 (簡体字)	中国	IBM-eucCN	IBM-1388	表意文字言語
Zh_CN	中国語	中国	GB18030	IBM-1388	表意文字言語
zh_TW	中国語 (繁体字)	台湾	IBM-eucTW	IBM-1371、IBM-937	表意文字言語

- サポートされる ISO 言語コードと ISO 国別コードの有効な組み合わせ (*language_COUNTRY*) を示します。表に示されているロケール名の各文字が大文字か小文字かは重要で、ロケールに対して特定のコード・ページが選択 (または暗示) されているロケール名の大/小文字を反映しない場合があります。選択したロケール名の各文字の適切な大/小文字については、ロケール・ディレクトリー (通常は `/usr/lib/nls/loc` に置かれている) にあるロケール・ファイル名を参照してください。UTF-8 コード・ページを含むロケール名は、通常はすべて大文字です (例えば、`ZH_CN.UTF-8`)。
- 関連した言語を示します。
- 関連した国または地域を示します。
- 対応する *language_COUNTRY* 値を持つロケールのコード・ページ ID として有効なコード・ページを示します。これらの表のエントリーは、確定したものではありません。有効なロケールの現在のリストについては、お使いのシステムのマニュアルで実行している AIX の特定のバージョンと構成について確認してください。選択するロケールは有効でなければなりません。つまり、プログラムを開発する場所と実行する場所の両方にインストールされている必要があります。
- 対応する *language_COUNTRY* 値を持つロケールのコード・ページ ID として有効なコード・ページを示します。これらのコード・ページは、`EBCDIC_CODEPAGE` 環境変数の内容として有効です。`EBCDIC_CODEPAGE` 環境変数が設定されていない場合は、この列内の右端に示すコード・ページ項目が、対応するロケールの EBCDIC コード・ページとして選択されます。

関連タスク

210 ページの『文字データのコード・ページの指定』

211 ページの『環境変数を使用したロケールの指定』

ロケール付きの照合シーケンスの制御

比較、ソート、マージなどのさまざまな操作は、プログラムおよびデータ項目に有効な照合シーケンスを使用します。照合シーケンスの制御方法は、データのクラスに対して有効なコード・ページに応じて、英字、英数字、DBCS、または国別のいずれかになります。

英字、英数字、または DBCS クラスを持つ項目にロケール・ベースの照合シーケンスが適用されるのは、`COLLSEQ(LOCALE)` コンパイラー・オプションが有効な場合のみで、`COLLSEQ(BIN)` または `COLLSEQ(EBCDIC)` が有効な場合には適用されません。同様に、国別クラス項目のロケール・ベースの照合シーケンスが適用されるのは、`NCOLLSEQ(LOCALE)` コンパイラー・オプションが有効な場合のみで、`NCOLLSEQ(BIN)` が有効な場合は適用されません。

COLLSEQ(LOCALE) または NCOLLSEQ(LOCALE) コンパイラー・オプションが有効な場合、ロケール・ベースの照合順序によって影響を受ける構文または意味体系規則を持つ言語エレメントには、コンパイル時のロケールが使用されます。それらは以下のようなものです。

- 条件名 VALUE 文節内の THRU 句
- EVALUATE ステートメント内の *literal-3* THRU *literal-4* 句
- ALPHABET 文節内の *literal-1* THRU *literal-2* 句
- SYMBOLIC CHARACTERS 文節で指定された文字の順序位置
- CLASS 文節内の THRU 句

COLLSEQ(LOCALE) コンパイラー・オプションが有効な場合、SORT または MERGE ステートメントの英数字キーの照合シーケンスは常に実行時のロケールに基づきます。

関連タスク

- 8 ページの『照合シーケンスの指定』
- 210 ページの『文字データのコード・ページの指定』
- 211 ページの『環境変数を使用したロケールの指定』
『ロケール付きの英数字照合シーケンスの制御』
- 218 ページの『ロケール付きの DBCS 照合シーケンスの制御』
- 219 ページの『ロケール付きの国別照合シーケンスの制御』
- 220 ページの『アクティブ・ロケールおよびコード・ページ値へのアクセス』
- 162 ページの『ソートまたはマージ基準の設定』

関連参照

- 213 ページの『サポートされるロケールおよびコード・ページ』
- 274 ページの『COLLSEQ』
- 296 ページの『NCOLLSEQ』

ロケール付きの英数字照合シーケンスの制御

プログラム照合シーケンスの場合、1 バイト英数字の照合シーケンスは、コンパイル時または実行時のロケールに基づきます。

ソース・プログラムで PROGRAM COLLATING SEQUENCE を指定すると、照合シーケンスはコンパイル時に設定され、実行時のロケールに関係なく使用されます。逆に、COLLSEQ コンパイラー・オプションを使用して照合シーケンスを設定すると、実行時のロケールが優先されます。

有効なコード・ページが 1 バイト ASCII コード・ページの場合、SPECIAL-NAMES 段落で以下の文節を指定できます。

- ALPHABET 文節
- SYMBOLIC CHARACTERS 文節
- CLASS 文節

有効なソース・コード・ページに DBCS 文字が含まれている場合にこれらの文節を指定すると、文節が診断され、コメントとして扱われます。COBOL のユーザー一定

義の英字名およびシンボリック文字の規則では、複数文字のシーケンスに依存する照合シーケンスではなく、文字単位の照合シーケンスを前提としています。

PROGRAM COLLATING SEQUENCE 文節を OBJECT-COMPUTER 段落内で指定した場合、*alphabet-name* に関連付けられている照合シーケンスを使用して、英数字比較の真の値が決定されます。また、PROGRAM COLLATING SEQUENCE 文節は、SORT または MERGE ステートメントで COLLATING SEQUENCE 句を指定していない限り、USAGE DISPLAY のソートおよびマージ・キーに適用されます。

COLLATING SEQUENCE 句または PROGRAM COLLATING SEQUENCE 文節を指定しない場合、有効な照合シーケンスはデフォルトで NATIVE になり、アクティブなロケール設定に基づきます。この設定は、SORT および MERGE ステートメントと、プログラムの照合シーケンスに適用されます。

この照合シーケンスは、次の項目の処理に影響します。

- ALPHABET 文節 (for example, *literal-1* THRU *literal-2* など)
- SYMBOLIC CHARACTERS 仕様
- レベル 88 の項目、比較条件、SORT および MERGE ステートメントに対する VALUE 範囲の指定

関連タスク

8 ページの『照合シーケンスの指定』

216 ページの『ロケール付きの照合シーケンスの制御』

『ロケール付きの DBCS 照合シーケンスの制御』

219 ページの『ロケール付きの国別照合シーケンスの制御』

162 ページの『ソートまたはマージ基準の設定』

関連参照

274 ページの『COLLSEQ』

データのクラスおよびカテゴリ (*COBOL for AIX 言語解説書*)

英数字比較 (*COBOL for AIX 言語解説書*)

ロケール付きの DBCS 照合シーケンスの制御

実行時のロケール・ベースの照合シーケンスは、リテラルの比較の場合を除き、常に DBCS データに適用されます。

クラス DBCS のデータ項目およびリテラルは、任意の関係演算子を持つ比較条件で使用することができます。その他のオペランドはクラス DBCS かクラス national、または英数字グループでなければなりません。DBCS 項目と編集済み DBCS 項目の間に区別はありません。

2 つの DBCS オペランドを比較する場合、COLLSEQ(LOCALE) コンパイラー・オプションが有効であれば、照合シーケンスはアクティブ・ロケールによって決定されます。有効でない場合、照合シーケンスは DBCS 文字のバイナリー値によって決定されます。PROGRAM COLLATING SEQUENCE 文節は、クラス DBCS のデータ項目またはリテラルを含む比較には影響しません。

DBCS 項目を国別項目と比較する場合、DBCS オペランドは、DBCS オペランドと同じ長さの基本国別項目に移動されるかのように処理されます。DBCS 文字は国別表現へ変換され、2 つの国別文字オペランドの比較が進行します。

DBCS 項目を英数字グループと比較する場合、変換または編集は行われません。2 つの英数字オペランドについては、比較が実行されます。比較は、データ表現には関係なくデータの各バイトを処理します。

関連タスク

- 8 ページの『照合シーケンスの指定』
- 206 ページの『DBCS リテラルの使用』
- 216 ページの『ロケール付きの照合シーケンスの制御』
- 217 ページの『ロケール付きの英数字照合シーケンスの制御』
- 『ロケール付きの国別照合シーケンスの制御』

関連参照

- 274 ページの『COLLSEQ』
- データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)
- 英数字比較 (COBOL for AIX 言語解説書)
- DBCS 比較 (COBOL for AIX 言語解説書)
- グループ比較 (COBOL for AIX 言語解説書)

ロケール付きの国別照合シーケンスの制御

USAGE NATIONAL の国別リテラルまたはデータ項目は、任意の関係演算子を持つ比較条件で使用することができます。PROGRAM COLLATING SEQUENCE 文節は国別オペランドを含む比較に影響を及ぼしません。

実行時にアクティブなロケールと関連付けられた照合順序のアルゴリズムに基づく比較を有効にするには、NCOLLSEQ(LOCALE) コンパイラー・オプションを使用します。NCOLLSEQ(BINARY) が有効な場合、照合シーケンスは国別文字のバイナリー値によって決定されます。

SORT または MERGE ステートメントが クラス国別になるのは、NCOLLSEQ(BIN) オプションが有効な場合のみです。

関連タスク

- 202 ページの『国別 (UTF-16) データの比較』
- 216 ページの『ロケール付きの照合シーケンスの制御』
- 218 ページの『ロケール付きの DBCS 照合シーケンスの制御』
- 162 ページの『ソートまたはマージ基準の設定』

関連参照

- 296 ページの『NCOLLSEQ』
- データのクラスおよびカテゴリー (COBOL for AIX 言語解説書)
- 国別比較 (COBOL for AIX 言語解説書)

照合シーケンスに依存する組み込み関数

次の組み込み関数は、文字の順序位置によって異なります。

ASCII コード・ページの場合、これらの組み込み関数は有効な照合シーケンスに基づいてサポートされます。EUC コード・ページまたは DBCS 文字を含むコード・ページの場合は、1 バイト文字の順序位置が、1 バイト文字の16 進数表現に対応するものとします。例えば、「A」の順序位置は 66 (X'41' + 1)、「*」の順序位置は 43 (X'2A' + 1) となります。

表 24. 照合シーケンスに依存する組み込み関数

組み込み関数	戻り	コメント
CHAR	順序位置引数に対応する文字	
MAX	最大値を含む引数の内容	引数は、英字、英数字、国別、または数字です。 ¹
MIN	最小値を含む引数の内容	引数は、英字、英数字、国別、または数字です。 ¹
ORD	文字引数の順序位置	
ORD-MAX	最大値を含む引数の、引数リスト内での整数順序位置	引数は、英字、英数字、国別、または数字です。 ¹
ORD-MIN	最小値を含む引数の、引数リスト内での整数順序位置	引数は、英字、英数字、国別、または数字です。 ¹
1. 関数に数値の引数が含まれている場合は、コード・ページと照合シーケンスが適用されません。		

これらの組み込み関数は、DBCS データ型ではサポートされません。

関連タスク

8 ページの『照合シーケンスの指定』

202 ページの『国別 (UTF-16) データの比較』

216 ページの『ロケール付きの照合シーケンスの制御』

アクティブ・ロケールおよびコード・ページ値へのアクセス

コンパイル時に有効なロケールを検証するには、コンパイラ・リストの最後の数行を検査します。

アプリケーションによっては、実行時にアクティブなロケールおよび EBCDIC コード・ページを検証し、コード・ページ ID を対応する CCSID に変換することが必要な場合もあります。このような照会や変換は、呼び出し可能なライブラリー・ルーチンを使用して実行することができます。

実行時にアクティブなロケールおよび EBCDIC コード・ページにアクセスするには、以下のようにライブラリー関数 `_iwzGetLocaleCP` を呼び出します。

```
CALL "_iwzGetLocaleCP" USING output1, output2
```

変数 `output1` は、次のフォーマットでヌル終了ロケール値を表す、20 文字の英数字項目です。

- 2 文字の言語コード
- 下線 (_)
- 2 文字の国別コード

- ピリオド (.)
- ロケールのコード・ページ値

例えば、en_US.IBM-1252 は、言語コード en、国別コード US、コード・ページ IBM-1252 のロケール値を表しています。

変数 *output2* は、有効なヌル終了 EBCDIC コード・ページ ID を表す、10 文字の英数字項目 (IBM-1140 など) です。

コード・ページ ID を対応する CCSID に変換するには、次のようにライブラリー関数 `_iwzGetCCSID` を呼び出します。

```
CALL "_iwzGetCCSID" USING input, output RETURNING returncode
```

input は、ヌル終了コード・ページ ID を表す英数字項目です。

output は、4 バイトの符号付き 2 進数データ項目 (PIC S9(5) COMP-5 として定義された項目など) です。入力コード・ページ ID ストリングまたはエラー・コード -1 に対応する CCSID が戻されます。

returncode は、次のように設定される、4 バイトの符号付き 2 進数データ項目です。

- | | |
|-----------|--|
| 0 | 成功 |
| 1 | コード・ページ ID は有効だが、関連する CCSID がない。 <i>output</i> は -1 に設定されます。 |
| -1 | コード・ページ ID が有効なコード・ページではない。 <i>output</i> は -1 に設定されます。 |

これらのサービスを呼び出すには、PGMNAME(MIXED) および NODYNAM コンパイラー・オプションを使用する必要があります。

『例: コード・ページ ID の取得および変換』

関連タスク

209 ページの『第 11 章 ロケールの設定』

関連参照

280 ページの『DYNAM』

299 ページの『PGMNAME』

319 ページの『第 15 章 コンパイラー指示ステートメント』

例: コード・ページ ID の取得および変換

次の例は、呼び出し可能サービス `_iwzGetLocaleCP` および `_iwzGetCCSID` を使用して、有効なロケールと EBCDIC コード・ページをそれぞれ取得し、コード・ページ ID を対応する CCSID に変換する方法を示しています。

```
cb1 pgmname(1m)
    Identification Division.
    Program-ID. "Samp1".
    Data Division.
    Working-Storage Section.
    01 locale-in-effect.
```

```

05 ll-cc          pic x(5).
05 filler-period pic x.
05 ASCII-CP      Pic x(14).
01 EBCDIC-CP     pic x(10).
01 CCSID         pic s9(5) comp-5.
01 RC           pic s9(5) comp-5.
01 n            pic 99.

```

Procedure Division.

Get-locale-and-codepages section.

Get-locale.

```

Display "Start Samp1."
Call "_iwzGetLocaleCP"
  using locale-in-effect, EBCDIC-CP
Move 0 to n
Inspect locale-in-effect
  tallying n for characters before initial x'00'
Display "locale in effect: " locale-in-effect (1 : n)
Move 0 to n
Inspect EBCDIC-CP
  tallying n for characters before initial x'00'
Display "EBCDIC code page in effect: "
  EBCDIC-CP (1 : n).

```

Get-CCSID-for-EBCDIC-CP.

```

Call "_iwzGetCCSID" using EBCDIC-CP, CCSID returning RC
Evaluate RC
  When 0
    Display "CCSID for " EBCDIC-CP (1 : n) " is " CCSID
  When 1
    Display EBCDIC-CP (1 : n)
      " does not have a CCSID value."
  When other
    Display EBCDIC-CP (1 : n) " is not a valid code page."
End-Evaluate.

```

Done.

Goback.

ロケールを ja_JP.IBM-943 (set LC_ALL=ja_JP.IBM-943) に設定した場合、このサンプル・プログラムからの出力は次のようになります。

```

Start Samp1.
locale in effect: ja_JP.IBM-943
EBCDIC code page in effect: IBM-1399
CCSID for IBM-1399 is 0000001399

```

関連タスク

211 ページの『環境変数を使用したロケールの指定』

第 3 部 プログラムのコンパイル、リンク、実行、デバッグ

第 12 章 プログラムのコンパイル、リンク、実行	225	矛盾するコンパイラー・オプション	265
環境変数の設定	225	ADATA	266
コンパイラー環境変数	226	ADDR	266
ランタイム環境変数	228	ARITH	268
TZ 環境パラメーター変数	234	BINARY	269
例: 環境変数の設定とアクセス	234	CALLINT	270
コンパイル済みプログラム	235	CHAR	271
コマンド行からのコンパイル	236	CICS	273
例: コンパイルでの cob2 の使用	237	COLLSEQ	274
シェル・スクリプトを使用したコンパイル	237	COMPILE	276
SFS ファイルのためのコンパイル	238	CURRENCY	277
PROCESS (CBL) ステートメントによるコンパイ		DATEPROC	278
ラー・オプションの指定	238	DIAGTRUNC	279
デフォルトのコンパイラー構成の変更	239	DYNAM	280
コンパイルの調整	240	ENTRYINT	280
構成ファイル内のスタンザ	241	EXIT	281
ソース・プログラムのエラーの訂正	242	文字ストリング形式	282
コンパイル・エラー・メッセージの重大度コード	242	ユーザー出口作業域	283
コンパイル・エラー・メッセージのリストの生成	243	出口モジュールのパラメーター・リスト	283
コンパイラー検出エラーに関するメッセージ		INEXIT の使用	284
およびリスト	243	LIBEXIT の使用	284
コンパイル・エラー・メッセージの形式	244	PRTEXIT の使用	286
cob2、cob2_j、および cob2_r オプション	245	ADEXIT の使用	286
コンパイルに適用されるオプション	245	FLAG	286
リンクに適用されるオプション	246	FLAGSTD	288
コンパイルとリンクの両方に適用されるオプシ		FLOAT	289
ョン	247	LIB	290
プログラムのリンク	247	LINECOUNT	291
リンカーへのオプションの引き渡し	248	LIST	291
例: リンクでの呼び出しコマンドの使用	250	LSTFILE	292
リンカーの入出力ファイル	251	MAP	293
リンカーの検索規則	252	MAXMEM	294
リンカー・ファイル名のデフォルト	253	MDECK	295
リンク内のエラーの訂正	253	NCOLLSEQ	296
プログラムの実行	254	NSYMBOL	296
COBOL for AIX ライブラリー・コンポーネントの		NUMBER	297
再配布	254	OPTIMIZE	298
第 13 章 オブジェクト指向アプリケーションの		PGMNAME	299
コンパイル、リンク、および実行	257	PGMNAME(UPPER)	300
オブジェクト指向アプリケーションのコンパイル	257	PGMNAME(MIXED)	301
オブジェクト指向アプリケーションの準備	258	PROFILE	301
例: COBOL クラス定義のコンパイルとリンク	259	QUOTE/APOST	301
オブジェクト指向アプリケーションの実行	260	SEPOBJ	302
main メソッドで始まるオブジェクト指向アプリ		バッチ・コンパイル	302
ケーションの実行	260	SEQUENCE	303
COBOL プログラムで始まるオブジェクト指向ア		SIZE	304
プリケーションの実行	261	SOSI	305
Java SDK サポート	261	SOURCE	306
第 14 章 コンパイラー・オプション	263	SPACE	307
		SPILL	308
		SQL	308

SSRANGE	309
TERMINAL	310
TEST	310
THREAD	311
TRUNC	312
TRUNC の例 1.	314
TRUNC の例 2.	314
VBREF	315
WSCLEAR	315
XREF	316
YEARWINDOW	317
ZWB	318
第 15 章 コンパイラー指示ステートメント	319
第 16 章 ランタイム・オプション	325
CHECK	325
DEBUG	326
ERRCOUNT	326
I FILEMODE	327
FILESYS	328
TRAP	328
UPSI	329
第 17 章 デバッグ	331
ソース言語によるデバッグ	331
プログラム・ロジックのトレース	332
入出力エラーの検出および処理	332
データの妥当性検査	333
初期化されていないデータの検出	333
プロシージャに関する情報の生成	334
例: USE FOR DEBUGGING	335
コンパイラー・オプションを使用したデバッグ	336
コーディング・エラーの検出	336
行シーケンス問題の検出	337
有効範囲の検査	337
診断するエラーのレベルの選択	338
例: 組み込みメッセージ	339
プログラム・エンティティ定義および参照の検 出	340
データ項目のリスト	341
デバッガーの使用	341
リストの入手	342
例: 短縮リスト	343
例: SOURCE および NUMBER 出力	345
例: MAP 出力	346
例: 組み込みマップ要約	346
MAP 出力で使用される用語および記号	347
例: ネストされたプログラム・マップ	348
例: XREF 出力: データ名相互参照	349
例: XREF 出力: プログラム名相互参照	350
I 例: XREF 出力: COPY/BASIS 相互参照	350
例: 組み込み相互参照	351
例: VBREF コンパイラー出力	352
オフセット情報を含むメッセージによるデバッグ	353
ユーザー出口のデバッグ	353
アセンブラー・ルーチンのデバッグ	354

第 12 章 プログラムのコンパイル、リンク、実行

続くセクションでは、環境変数の設定、コンパイル、リンク、実行、エラーの訂正、および ライブラリー・コンポーネントの再配布の方法について説明します。

関連タスク

『環境変数の設定』

235 ページの『コンパイル済みプログラム』

242 ページの『ソース・プログラムのエラーの訂正』

247 ページの『プログラムのリンク』

253 ページの『リンク内のエラーの訂正』

254 ページの『プログラムの実行』

254 ページの『COBOL for AIX ライブラリー・コンポーネントの再配布』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

環境変数の設定

プログラムに必要な値を設定するには、環境変数を使用します。環境変数の値を指定するには、`export` コマンドまたは `putenv()` POSIX 関数を使用します。環境変数を設定しなかった場合は、デフォルト値が適用されるか、またはその変数は定義されません。

環境変数 は、変化する可能性のあるユーザー環境またはプログラム環境の一部を定義します。例えば、プログラムが別のプログラムによって動的に呼び出されたときに、COBOL ランタイムでそのプログラムを検索できる場所を定義するには、`COBPATH` 環境変数を使用します。環境変数は、コンパイラーおよびランタイム・ライブラリーの両方で使用されます。

IBM COBOL for AIX をインストールすると、インストール・プロセスによって、COBOL for AIX コンパイラーおよびランタイム・ライブラリーにアクセスするための環境変数が設定されます。単純な COBOL プログラムをコンパイルおよび実行するために設定する必要がある環境変数は、`NLSPATH` および `LANG` だけです。これらの変数は、インストール時に設定されます。

`export` コマンドを使用すれば、2 つある以下の場所のどちらかで環境変数の値を変更できます。

- コマンド・シェルのプロンプト (例: XTERM ウィンドウ)。この環境変数の定義は、該当のシェルまたはその下位シェル (該当のシェルから直接または間接的に呼び出されるシェル) から実行するプログラム (プロセスまたは子プロセス) に適用されます。
- ホーム・ディレクトリーにある `.profile` ファイル。 `.profile` ファイル内の環境変数を定義すると、それらの変数の値は AIX セッションを開始するときに必ず自動的に定義され、その値がすべてのシェル・プロセスに適用されます。

また、`putenv()` POSIX 関数を使用して COBOL プログラム内から環境変数を設定したり、`getenv()` POSIX 関数を使用して環境変数にアクセスすることもできます。

一部の環境変数 (`COBPATH` や `NLSPATH` など) は、ファイル検索を行うディレクトリーを定義します。複数のディレクトリー・パスがリストされる場合は、各パスがコロンで区切られます。環境変数で定義されたパスは、`export` ステートメント内の最初のパスから最後のパスへの順番に評価されます。環境変数のパス内で同じ名前のファイルが複数定義されている場合は、最初に 見つかったファイルのコピーが使用されます。

例えば、次の `export` コマンドを発行した場合、2 つのディレクトリーを含むように `COBPATH` 環境変数 (この環境変数は、動的にアクセスされるプログラムを COBOL ランタイムが検索できる場所を定義します) が設定されます。これらのディレクトリーのうち、最初に指定されたディレクトリーが先に検索されます。

```
export COBPATH=/users/me/bin:/mytools/bin
```

234 ページの『例: 環境変数の設定とアクセス』

関連タスク

240 ページの『コンパイルの調整』

関連参照

『コンパイラー環境変数』

228 ページの『ランタイム環境変数』

コンパイラー環境変数

COBOL コンパイラーでは、次の環境変数を使用します。

COBOL ワードは大/小文字を区別しないよう定義されているため、*library-name* や *text-name* などの COBOL ワードの文字はすべて、大文字として扱われます。したがって、そのような名前に対応する環境変数名は大文字でなければなりません。例えば、`COPY MyCopy` に対応する環境変数名は `MYCOPY` です。

COBCPYEXT

COPY name ステートメントにファイル接尾部が指定されていない場合は、コピーブックの検索に使用するためのファイル接尾部を指定します。3 文字のファイル接尾部を 1 つ以上指定します。拡張子の前のピリオドはなくても構いません。複数のファイル接尾部を指定する場合は、スペースまたはコマで区切ります。

`COBCPYEXT` が定義されていない場合は、`CPY`、`CBL`、`COB` (小文字の `cpy`、`cbl`、`cob` も同等) の各接尾部が検索されます。

COBLSTDIR

コンパイラー・リスト・ファイルの書き込み先ディレクトリーを指定します。有効な任意のパスを指定します。絶対パスを指定するには、先行スラッシュを指定します。それ以外の場合は、現行ディレクトリーからの相対パスになります。末尾のスラッシュはオプションです。

`COBLSTDIR` が定義されていない場合は、コンパイラー・リストが現行ディレクトリーに書き込まれます。

COBOPT

コンパイラー・オプションを指定します。複数のコンパイラー・オプションを指定するには、各オブジェクトをスペースまたはコンマで区切ります。AIX コマンド・シェルに対して意味を持つブランクまたは文字がリストに含まれている場合は、オプション・リストを引用符で囲みます。以下に、その例を示します。

```
export COBOPT="TRUNC(OPT) TERMINAL"
```

個々のコンパイラー・オプションにデフォルト値が適用されます。

DB2DBDFT

組み込み SQL ステートメントでプログラムをコンパイルするためのデータベースを指定します。

library-name

ユーザー定義語として *library-name* を指定した場合は、名前が環境変数として使用され、この環境変数の値がコピーブックの位置を指定するパスに使用されます。

ライブラリー名を指定しない場合は、コンパイラーが次の順序でライブラリー・パスを検索します。

1. 現行ディレクトリー
2. -Ixxx オプションで指定されたパス (設定されている場合)
3. SYSLIB 環境変数で指定されたパス

ファイルが検出されると、検索は終了します。

詳細については、コンパイラー指示ステートメントに関する下記の関連参照で COPY ステートメントのドキュメンテーションを参照してください。

LANG 詳しくは、下のランタイム環境変数についての『関連参照』を参照してください。

NLSPATH

詳しくは、下のランタイム環境変数についての『関連参照』を参照してください。

SYSLIB

ライブラリー名による修飾なしのテキスト名で、COBOL の COPY ステートメントに使用するパスを指定します。また、SQL INCLUDE ステートメントに使用するパスも指定します。

text-name

ユーザー定義語として *text-name* を指定した場合は、環境変数の値がファイル名として使用され、またコピーブックのパス名として使用される場合もあります。

複数のパス名を指定するには、各パス名をコロン (:) またはセミコロン (;) で区切ります。

詳細については、コンパイラー指示ステートメントに関する下記の関連参照で COPY ステートメントのドキュメンテーションを参照してください。

関連概念

357 ページの『DB2 coprocessor』

関連タスク

359 ページの『DB2 coprocessor を用いた SQL INCLUDE の使用』

関連参照

『ランタイム環境変数』

245 ページの『cob2、cob2_j、および cob2_r オプション』

263 ページの『第 14 章 コンパイラー・オプション』

319 ページの『第 15 章 コンパイラー指示ステートメント』

ランタイム環境変数

COBOL ランタイム・ライブラリーでは、次の環境変数を使用します。

割り当て名

ASSIGN 文節で指定する COBOL ファイルです。この *assignment-name* の用法は、COBOL ワードの規則に従います。以下に、その例を示します。

```
export OUTPUTFILE=january.car_results
```

環境変数として設定されていないユーザー定義語に対して割り当てを行う場合は、ユーザー定義語のリテラル名を持つファイル (下記の例では OUTPUTFILE) に対して割り当てが行われます。割り当てが有効であれば、このファイルが現行ディレクトリーに書き込まれます。

assignment-name を設定したら、この環境変数を COBOL ユーザー定義語として ASSIGN 文節で使用することができます。

前の export ステートメントに基づいて、COBOL ソース・プログラムには次の SELECT および ASSIGN 文節が含まれます。

```
SELECT CARPOOL ASSIGN TO OUTPUTFILE
```

OUTPUTFILE は環境変数で定義できるため、このステートメントによりファイル `january.car_results` ヘデータが書き込まれます。

ASSIGN を使用して、標準言語ファイル・システム (STL)、レコード順次区切りファイル・システム (RSD)、VSAM などの代替ファイル・システムに格納されるファイルを指定することができます。

CLASSPATH

オブジェクト指向アプリケーションに必要な Java™ .class ファイルのディレクトリー・パスを指定します。

COBJVMINTOPTIONS

COBOL が JVM を初期化するときを使用される Java 仮想マシン (JVM) オプションを指定します。

COBMSGGS

ランタイム・エラー・メッセージの書き込み先ファイルの名前を指定します。

ランタイム・エラー・メッセージをファイルに取り込むには、export コマンドを使用して COBMSGGS をファイル名に設定します。アプリケーションを強制終了させるランタイム・エラーがプログラム内で発生した場合は、COBMSGGS の値として設定されたファイルに、強制終了の理由を示すエラー・メッセージが入ります。

COBMSGS が設定されていない場合は、エラー・メッセージが端末に出力されます。

COBPATH

COBOL ランタイムが共用ライブラリーなどの動的にアクセスされるプログラムを探すためのディレクトリー・パスを指定します。

動的ロードが必要なプログラムを実行するには、この変数を設定する必要があります。以下に例を示します。

```
export COBPATH=/pgmpath/pgmshlib
```

COBRTOPT

COBOL ランタイム・オプションを指定します。

ランタイム・オプションが複数ある場合は、コンマまたはコロンの区切りをします。サブオプションの区切り文字には、括弧または等号 (=) を使用します。オプションは大/小文字の区別をしません。例えば、次の 2 つのコマンドは同じです。

```
export COBRTOPT="check(ON),UPSI(00000000)"  
export COBRTOPT='CHECK=on:UPSI=00000000'
```

各ランタイム・オプションにはデフォルトが適用されます。詳細については、ランタイム・オプションに関する下記の関連参照をご覧ください。

EBCDIC_CODEPAGE

CHAR(EBCDIC) または CHAR(S390) コンパイラー・オプションを使用してコンパイルされるプログラムが処理する EBCDIC データに適用できる EBCDIC コード・ページを指定します。

EBCDIC コード・ページを設定するには、次のコマンドを発行します。この場合、*codepage* には使用するコード・ページの名前が入ります。

```
export EBCDIC_CODEPAGE=codepage
```

EBCDIC_CODEPAGE が設定されていない場合は、サポートされるロケールとコード・ページに関する下記の関連参照に示されているように、現在のロケールに基づいてデフォルトの EBCDIC コード・ページが選択されます。CHAR(EBCDIC) コンパイラー・オプションが有効で、有効なロケールに対して複数の EBCDIC コード・ページを適用できる場合は、ロケールのデフォルト EBCDIC コード・ページが受け入れ可能でなければ、EBCDIC_CODEPAGE 環境変数を設定する必要があります。

ENCINA_CDS_ROOT

Encina SFS サーバー名を指定します。以下に、その例を示します。

```
export ENCINA_CDS_ROOT=./cics/sfs/test1
```

ENCINA_SFS_DATA_VOLUME

SFS ファイルが作成される SFS データ・ボリュームの名前を指定します。以下に、その例を示します。

```
export ENCINA_SFS_DATA_VOLUME=sfs_SFS_SERV
```

このデータ・ボリュームは、アプリケーションがアクセスする SFS サーバーに対して定義されている必要があります。

この変数が設定されていない場合は、デフォルト名 `encinaSfsVol` が使用されます。

ENCINA_SFS_INDEX_VOLUME

代替索引ファイルが作成される SFS データ・ボリュームの名前を指定します。以下に、その例を示します。

```
export ENCINA_SFS_INDEX_VOLUME=sfs_sfs_serv
```

このデータ・ボリュームは、アプリケーションがアクセスする SFS サーバーに対して定義されている必要があります。

この変数が設定されていない場合は、代替索引ファイルが、対応するベース索引ファイルと同じデータ・ボリューム上に作成されます。

ENCINA_VSAM_AUTO_FLUSH

Encina SFS ファイルに対する入出力操作ごとのすべての変更が、制御がアプリケーションに戻される前にディスクにコミットされるかどうか（つまり、SFS の強制操作機能を使用可能にするかどうか）を指定します。この環境変数に `OFF` を指定することによって、SFS ファイルを使用するアプリケーションのパフォーマンスを向上させることができます。以下に、その例を示します。

```
export ENCINA_VSAM_AUTO_FLUSH=OFF
```

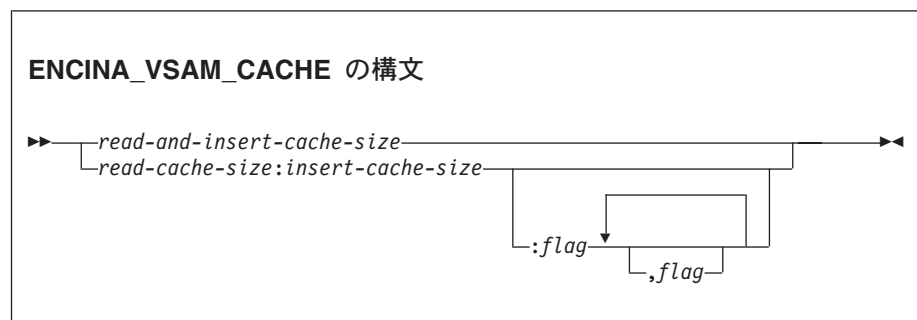
この環境変数が `OFF` に設定されている場合、SFS は遅延書き込み方針を使用します。つまり、SFS ファイルに対する変更は、ファイルがクローズされるまではディスクに書き込まれません。

SFS のクライアント側のキャッシュが有効な場合（つまり、環境変数 `ENCINA_VSAM_CACHE` がゼロ以外の有効な値に設定されている場合）、`ENCINA_VSAM_AUTO_FLUSH` の設定は無視されます。強制操作は使用不可です。

SFS のクライアント側のキャッシュが有効でない場合、`ENCINA_VSAM_AUTO_FLUSH` の値はデフォルトの `ON` に設定されます。

ENCINA_VSAM_CACHE

SFS ファイルに対するクライアント側のキャッシュが使用可能かどうかを指定します。クライアント側のキャッシュを使用可能にすることによって、SFS ファイルを使用するアプリケーションのパフォーマンスを向上させることができます。



サイズの単位はページ数です。ゼロのサイズは、キャッシュが使用不可であることを示します。指定できるフラグは次のとおりです。

ALLOW_DIRTY_READS

アクセスするファイルがロックされている必要があるという読み取りキャッシュの制限を取り除きます。

INSERTS_DESPITE_UNIQUE_INDICES

クラスター・ファイルのすべてのアクティブな索引と、入力順および相対ファイルのアクティブな代替索引が、重複を許可する場合のみ挿入をキャッシュするという、挿入キャッシュの制限を取り除きます。

例えば、次のコマンドは読み取りキャッシュのサイズを 16 ページに設定し、挿入キャッシュのサイズを 64 ページに設定します。また、ダーティー読み取りを可能にし、ユニーク索引であっても挿入を可能にします。

```
export ENCINA_VSAM_CACHE=16:64:ALLOW_DIRTY_READS, ¥  
INSERTS_DESPITE_UNIQUE_INDICES
```

デフォルトでは、クライアント側のキャッシュは使用不可です。

LANG ロケールを指定します (環境変数を使用たてロケールの指定に関する関連タスクを参照)。後述のように、LANG は NLSPATH 環境変数の値にも影響を与えます。

例えば、次のコマンドは言語ロケール名を米国英語に設定します。

```
export LANG=en_US
```

LC_ALL

ロケールを指定します。LC_ALL を使用するロケール設定は、LANG またはその他の LC_xx 環境変数を使用する設定よりも優先されます (環境変数を使用したロケールの指定に関する関連タスクを参照)。

LC_COLLATE

ロケールの照合動作を指定します。LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LC_CTYPE

ロケールのコード・ページを指定します。LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LC_MESSAGES

ロケールのメッセージの言語を指定します。LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LC_TIME

日時情報の形式に使用するロケールを決定します。LC_ALL が指定されている場合は、この設定がオーバーライドされます。

LIBPATH

COBOL ランタイム・ライブラリーの絶対パス名を指定します。デフォルトはインストール時に設定されます。

LOCPATH

ロケール情報データベース用の検索パスを指定します。パスは、コロンで区切られたディレクトリー名のリストです。

LOCPATH は、英数字データ項目のロケール・ベースの比較など、ロケールを参照するすべての操作に使用されます。

NLSPATH

メッセージ・カタログおよびヘルプ・ファイルの絶対パス名を指定します。NLSPATH の設定は必須です。インストール時には `usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat` の初期値が設定されます。%L には、LANG 環境変数で指定された値が代入されます。%N には、COBOL で使用されるメッセージ・カタログ名が代入されます。

COBOL for AIX インストールは、`usr/lib/nls/msg/xx` 内のメッセージに入れます。ここで、`xx` は、COBOL for AIX がサポートする言語です。デフォルトは `en_US` です。

NLSPATH を設定する際には、値を置き換えるのではなく NLSPATH に値を追加してください。この環境変数は、他のプログラムが使用する可能性があります。以下に例を示します。

```
DIR=xxxx
nlspath=$DIR/%L/%N:$NLSPATH
export NLSPATH
```

`xxxx` は、COBOL がインストールされているディレクトリーです。ディレクトリー `xxxx` には、COBOL メッセージ・カタログを含むディレクトリー `xxxx/en_US` (英語の言語設定の場合) が含まれている必要があります。

この製品には、次の言語によるメッセージが組み込まれています。

jp-Jp 英語

ja_JP、Ja_JP

日本語

メッセージの言語とロケール設定の言語は別に指定することができます。例えば、環境変数 LANG を `en_US` に設定し、環境変数 LC_ALL を `ja_JP.IBM-eucJP` に設定することが可能です。この例では、COBOL コンパイラまたはランタイム・メッセージはすべて英語で処理され、プログラム内のネイティブ ASCII (DISPLAY または DISPLAY-1) データはコード・ページ IBM-eucJP (ASCII 日本語コード・ページ) のエンコードとして処理されます。

コンパイラは、NLSPATH および LANG 環境変数の値の組み合わせを使用して、メッセージ・カタログにアクセスします。NLSPATH が正しく設定されているが、LANG が上記のロケール値のいずれかに設定されていない場合は、警告メッセージが生成され、コンパイラは `en_US` メッセージ・カタログをデフォルトに設定します。NLSPATH 値が無効の場合、終了エラー・メッセージが生成されます。

ランタイム・ライブラリーも NLSPATH を使用してメッセージ・カタログにアクセスします。NLSPATH を正しく設定しないと、ランタイム・メッセージが短縮形で戻されます。

PATH 実行可能プログラムのディレクトリー・パスを指定します。

SYSIN, SYSIPT, SYSOUT, SYSLIST, SYSLST, CONSOLE, SYSPUNCH, SYSPCH

これらの COBOL 環境名は、ACCEPT および DISPLAY ステートメントで使用される簡略名に対応する環境変数名として使用されます。

環境変数を設定しない場合、デフォルトでは、SYSIN および SYSIPT が論理入力装置 (キーボード) に割り当てられます。SYSOUT、SYSLIST、SYSLST、CONSOLE は、システムの論理出力装置 (画面) に割り当てられます。SYSPUNCH および SYSPCH には、デフォルトでは値が割り当てられません。これらは明示的に定義しない限り無効です。

例えば、次のコマンドは CONSOLE を定義します。

```
export CONSOLE=/users/mypath/myfile
```

CONSOLE は、次のソース・コードとともに使用される場合があります。

```
SPECIAL-NAMES.  
  CONSOLE IS terminal  
  .  
  .  
  .  
  DISPLAY 'Hello World' UPON terminal
```

TMP SORT および MERGE 関数の一時作業ファイル (必要な場合) の場所を指定します。デフォルトは不定で、ソート・ユーティリティー・インストール・プログラムによって設定されます。

以下に例を示します。

```
export TMP=/tmp
```

TZ ロケールによって使用される時間帯情報を記述します。TZ の形式は次のとおりです。

```
export TZ=SSS[+|-]nDDD[,sm,sw,sd,st,em,ew,ed,et,shift]
```

TZ がいない場合は、デフォルトのロケール値として EST5EDT が使用されます。標準時間帯しか指定しない場合は、*n* (GMT からの時間差) のデフォルト値が 5 ではなく 0 になります。

sm、*sw*、*sd*、*st*、*em*、*ew*、*ed*、*et*、*shift* のうち 1 つでも値を指定する場合は、これらすべての値を指定する必要があります。これらの値のうち 1 つでも無効な場合は、ステートメント全体が無効と見なされ、時間帯情報は変更されません。

例えば、次のステートメントは、標準時間帯を CST、夏時間調整を CDT、CST と UTC の時間差を 6 時間に設定します。夏時間調整の開始値と終了値は設定されません。

```
export TZ=CST6CDT
```

これ以外に考えられる値としては、Pacific United States を表す PST8PDT や、Mountain United States を表す MST7MDT があります。

関連タスク

211 ページの『環境変数を使用したロケールの指定』

121 ページの『ファイルの識別』

148 ページの『COBOL アプリケーションで SFS ファイルにアクセスできるように』

する』

151 ページの『SFS パフォーマンスの向上』

225 ページの『環境変数の設定』

関連参照

213 ページの『サポートされるロケールおよびコード・ページ』

『TZ 環境パラメーター変数』

271 ページの『CHAR』

325 ページの『第 16 章 ランタイム・オプション』

TZ 環境パラメーター変数

TZ 変数値の定義は次のとおりです。

表 25. TZ 環境パラメーター変数

変数	説明	デフォルト値
SSS	標準時間帯 ID。アルファベットで始まる 3 文字でなければなりません が、スペースが含まれていても構いません。	EST
<i>n</i>	標準時間帯と、協定世界時 (UTC) (以前はグリニッジ標準時 (GMT)) との時間差。正数はグリニッジ子午線より西の時間帯を示します。負数 はグリニッジ子午線より東の時間帯を示します。	5
DDD	夏時間調整 (DST) 時間帯 ID。アルファベットで始まる 3 文字でなけ ればなりません、スペースが含まれていても構いません。	EDT
<i>sm</i>	DST の開始月 (1 から 12)	4
<i>sw</i>	DST の開始週 (-4 から 4)	1
<i>sd</i>	DST の開始日 (<i>sw</i> が 0 以外の場合は 0 から 6、 <i>sw</i> が 0 の場合は 1 から 31)	0
<i>st</i>	DST の開始時刻 (秒単位)	3600
<i>em</i>	DST の終了月 (1 から 12)	10
<i>ew</i>	DST の終了週 (-4 から 4)	-1
<i>ed</i>	DST の終了日 (<i>ew</i> が 0 以外の場合は 0 から 6、 <i>ew</i> が 0 の場合は 1 から 31)	0
<i>et</i>	DST の終了時刻 (秒単位)	7200
<i>shift</i>	時間変化量 (秒単位)	3600

例: 環境変数の設定とアクセス

次の例は、標準の POSIX 関数 `getenv()` および `putenv()` を呼び出すことによって、COBOL プログラムから環境変数にアクセスする方法と環境変数を設定する方法を示しています。

`getenv()` と `putenv()` は C 関数なので、BY VALUE によって引数を渡さなければなりません。文字ストリングは、ヌル終了ストリングを指し示す BY VALUE ポインターとして渡さなければなりません。これらの関数を呼び出すプログラムは、NODYNAM オプションと PGMNAME(LONGMIXED) オプションを指定してコンパイルします。

```

CBL pgmname(longmixed),nodynam
Identification division.
Program-id. "envdemo".
Data division.
Working-storage section.
01 P pointer.
01 PATH pic x(5) value Z"PATH".
01 var-ptr pointer.
01 var-len pic 9(4) binary.
01 putenv-arg pic x(14) value Z"MYVAR=ABCDEFG".
01 rc pic 9(9) binary.
Linkage section.
01 var pic x(5000).
Procedure division.
* Retrieve and display the PATH environment variable
  Set P to address of PATH
  Call "getenv" using by value P returning var-ptr
  If var-ptr = null then
    Display "PATH not set"
  Else
    Set address of var to var-ptr
    Move 0 to var-len
    Inspect var tallying var-len
      for characters before initial X"00"
    Display "PATH = " var(1:var-len)
  End-if
* Set environment variable MYVAR to ABCDEFG
  Set P to address of putenv-arg
  Call "putenv" using by value P returning rc
  If rc not = 0 then
    Display "putenv failed"
  Stop run
  End-if
Goback.

```

コンパイル済みプログラム

プログラムのコンパイルに使用するオプションは、次の方法で指定することができます。

以下のことが可能です。

- コマンド行で **COBOPT** 環境変数を設定する。
- コンパイラー・オプションを **cob2**、**cob2_j**、または **cob2_r** コマンドへのオプションとして指定します。これらのコマンドは、コマンド行、シェル・スクリプト、または **Make** ファイルで使用できます。
- **PROCESS (CBL)** または ***CONTROL** ステートメントを使用する。**PROCESS** を使用して指定するオプションは他のオプション指定をオーバーライドします。

関連タスク

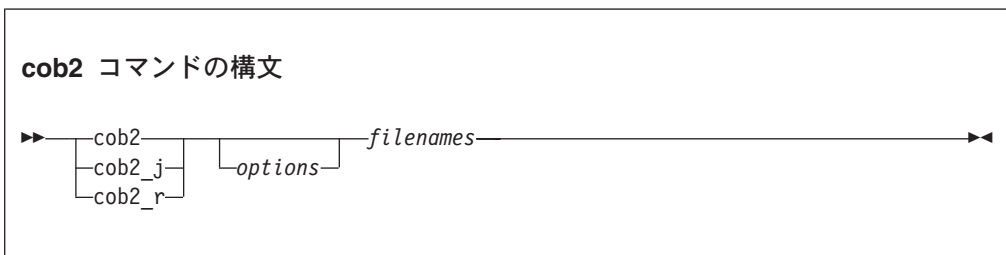
- 236 ページの『コマンド行からのコンパイル』
- 237 ページの『シェル・スクリプトを使用したコンパイル』
- 238 ページの『SFS ファイルのためのコンパイル』
- 238 ページの『**PROCESS (CBL)** ステートメントによるコンパイラー・オプションの指定』
- 239 ページの『デフォルトのコンパイラー構成の変更』
- 493 ページの『第 25 章 プラットフォーム間でのアプリケーションの移植』

関連参照

- 226 ページの『コンパイラ環境変数』
- 245 ページの『cob2、cob2_j、および cob2_r オプション』
- 263 ページの『第 14 章 コンパイラ・オプション』
- 265 ページの『矛盾するコンパイラ・オプション』
- 615 ページの『付録 A. 移行および互換性』
- 619 ページの『付録 B. ホスト COBOL との違いの要約』

コマンド行からのコンパイル

コマンド行から COBOL プログラムをコンパイルするには、そのプログラムのタイプに応じて、cob2、cob2_j、または cob2_r コマンドを発行します。(各コマンドはリンカーも呼び出します)。



コマンドを次のように選択します。

- オブジェクト指向クライアント・プログラムおよびクラス定義の場合は cob2_j
- 次のアプリケーションの場合は cob2_r
 - SFS ファイル・システム を使用するアプリケーション
 - THREAD コンパイラ・オプションを使用してコンパイルする CICS 以外のアプリケーション。
 - CICS アプリケーション。このアプリケーションは NOThread コンパイラ・オプション (デフォルト) を使用してコンパイルします。
- 上記以外の場合は cob2

cob2_j コマンドを使用してコンパイルすると、THREAD コンパイラ・オプションがデフォルトの呼び出しオプションとして自動的に組み込まれます。

複数のファイルをコンパイルするには、コマンド行の任意の位置でファイル名を指定します。オプションおよびファイル名を区切るには、スペースを使用します。例えば、次の 2 つのコマンドは同じです。

```
cob2 -g -qlib filea.cbl fileb.cbl -qflag(w)
cob2 filea.cbl -g -qflag(w) fileb.cbl -qlib
```

この 4 つのコマンドはそれぞれ、コマンド行に指定する順にコンパイラとリンカー・オプションを受け入れます。指定したオプションは、コマンド行のすべてのファイルに適用されます。(いずれかのコマンドを使用するときにリンクされるライブラリーについて詳しくは、プログラムのリンクに関する関連タスクを参照してください。)

コンパイラーに渡されるのは、特定の接尾部 (.cbl など) を持つファイルのみです。それ以外のすべてのファイルはリンカーに渡されます。

コンパイラー入出力のデフォルトの位置は、現行ディレクトリーです。

『例: コンパイルでの cob2 の使用』

関連タスク

239 ページの『デフォルトのコンパイラー構成の変更』

247 ページの『プログラムのリンク』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

263 ページの『第 14 章 コンパイラー・オプション』

例: コンパイルでの cob2 の使用

次の例に、さまざまな cob2 呼び出しで生成される出力を示します。

表 26. cob2 コマンドからの出力

コンパイル対象	入力	生成されるファイル
alpha.cbl	cob2 -c alpha.cbl	alpha.o
alpha.cbl および beta.cbl	cob2 -c alpha.cbl beta.cbl	alpha.o, beta.o
alpha.cbl (LIST および ADATA オプションを使用)	cob2 -qlist,adata alpha.cbl	alpha.wlist、alpha.o ¹ 、 alpha.lst、 alpha.adt、 a.out
1. リンクが成功すると、これらのファイルは削除されます。		

関連タスク

236 ページの『コマンド行からのコンパイル』

関連参照

266 ページの『ADATA』

291 ページの『LIST』

シェル・スクリプトを使用したコンパイル

シェル・スクリプトを使用して、cob2、cob2_j、または cob2_r コマンドを自動化することができます。

無効な構文がコマンドに渡されないようにするには、次の指針に従います。

- コンパイラー・サブオプションを区切るには、括弧ではなく等号とコロンの使用してください。
- コンパイラー・サブオプションを区切るには、単一引用符ではなく下線を使用してください。
- オプション・ストリングを引用符 ("") で囲む場合を除き、ストリング内にはリンクを使用しないでください。

表 27. シェル・スクリプト内のコンパイラー・オプション構文の例

シェル・スクリプトでの使用	コマンド行での使用
-qFLOAT=NATIVE:,CHAR=NATIVE:	-qFLOAT(NATIVE),CHAR(NATIVE)
-qEXIT=INEXIT=_String_,MYMODULE::	-qEXIT(INEXIT('String',MYMODULE))

SFS ファイルのためのコンパイル

SFS ファイル・システムを使用する場合は、cob2_r コマンドを使用してプログラムをコンパイルします。

以下に例を示します。

```
cob2_r alpha.cbl
```

-# オプションを使用して、どのライブラリーがリンクされるかを確認することができます。例えば、

```
cob2_r -# alpha.o
```

関連タスク

148 ページの『COBOL アプリケーションで SFS ファイルにアクセスできるようにする』

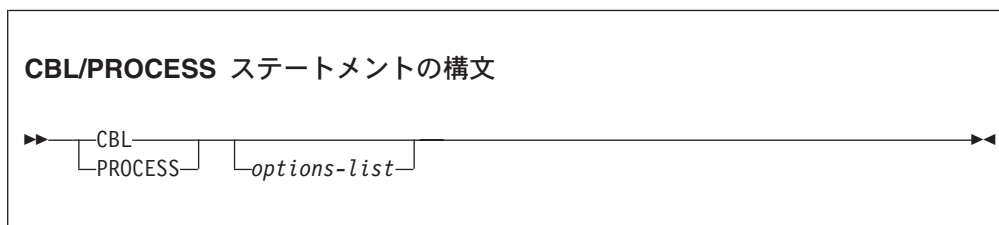
関連参照

125 ページの『SFS ファイル・システム』

245 ページの『cob2、cob2_j、および cob2_r オプション』

PROCESS (CBL) ステートメントによるコンパイラー・オプションの指定

COBOL プログラムの PROCESS ステートメントにコンパイラー・オプションをコーディングすることができます。これは、IDENTIFICATION DIVISION ヘッダーの前、かつコメント行またはコンパイラー指示ステートメントの前にコーディングする必要があります。



シーケンス・フィールドをコーディングしない場合は、PROCESS ステートメントを 1 から 66 桁目で始めることができます。シーケンス・フィールドは 1 から 6 桁目に指定することができます。シーケンス・フィールドが使用される場合には、そこに 6 文字が含まれていなければならない、最初の文字は数字でなければなりません。シーケンス・フィールドを付けて使用する場合には、PROCESS は 8 から 66 桁目の間で開始することができます。

CBL を PROCESS の同義語として使用することができます。CBL は 1 から 70 桁目の間で開始することができます。シーケンス・フィールドを付けて使用する場合には、CBL は 8 から 70 桁目の間で開始することができます。

PROCESS と *options-list* の最初のオプションとは、1 つ以上の空白で区切らなければなりません。オプションはコンマまたは空白で区切ります。個々のオプションとそのサブオプションの間にスペースを入れてはなりません。

複数の PROCESS ステートメントを使用することができます。そうする場合は、間にステートメントを入れずに、PROCESS ステートメントを続けて使用する必要があります。複数の PROCESS ステートメントにわたってオプションを継続することはできません。

プログラミングの編成で、COBOL コンパイラーのデフォルト・オプション・モジュールを使用して、PROCESS ステートメントを使用することを禁止することができます。編成で許可されていない PROCESS ステートメントが COBOL プログラムで見つかった場合、COBOL コンパイラーはエラー診断を生成します。

関連参照

CBL (PROCESS) ステートメント (*COBOL for AIX 言語解説書*)

デフォルトのコンパイラー構成の変更

cob2 コマンドで使用するデフォルト・オプションは、構成ファイル (デフォルトでは `/etc/cob2.cfg`) から取得されます。cob2 (または `cob2_j`、`cob2_r`) で使用するオプションは、そのコマンドに `-#` オプションを指定することによって表示できます。

デフォルトの構成ファイルを使用している場合、コマンド `cob2 -# abc.cb1` によって次の出力が表示されます。

```
exec: /usr/lpp/cobol/bin/IGYCCOB2 abc.cb1
exec: /usr/bin/ld -bpT:0x10000000 -bpD:0x20000000 -bhalt:5 /lib/crt0.o abc.o
      -lcob2s -L/usr/lpp/cobol/lib -lsmrtlite -lC128 -lC -lc
```

`/etc/cob2.cfg` を (`cat` コマンドで) 編集したり表示したりする場合、使用する COBOL コンパイラーが指定された行があることに気がきます。

```
DEFLT: compiler1 = /usr/lpp/cobol/bin/IGYCCOB2
```

ファイル内のその他の行は、リンカーとさまざまなオプションを指定します。

`cob2.cfg` 構成ファイルを変更して、デフォルト・オプションやコンパイラーとライブラリーの場所などを変更することができます。例えば、`cob2.cfg` の `cob2_j` スタンザは Java ライブラリー名です。使用している Java のリリース・レベルを示すために、Java ライブラリー名の更新が必要になる場合があります。

デフォルトの構成ファイルを変更する代わりに、目的に合うようにファイルのコピーを調整することができます。

関連概念

261 ページの『Java SDK サポート』

関連タスク

240 ページの『コンパイルの調整』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

241 ページの『構成ファイル内のスタンザ』

コンパイルの調整

要件に合わせてコンパイルを調整するために、デフォルト構成ファイルのコピーを変更して、それをさまざまな方法で使用することができます。

構成ファイル `/etc/cob2.cfg` には、`cob2:`、`cob2_j:`、および `cob2_r:` を開始するセクション (スタンザ) があります。これらのスタンザをリストするには、コマンド `ls /usr/bin/cob2*` を使用します。結果のリストは次のものを示しています。

```
/usr/bin/cob2
/usr/bin/cob2_j
/usr/bin/cob2_r
```

後ろ 2 つのファイルは、最初のファイルにリンクされています。コマンド `cob2`、`cob2_j`、および `cob2_r` は、同じモジュールを実行します。ただし、`cob2` は `cob2` スタンザを使用し、`cob2_j` は `cob2_j` スタンザを使用します (他も同様)。

以下のステップを実行すれば、コンパイルを調整できます。

1. デフォルト構成ファイル `/etc/cob2.cfg` のコピーを作成します。
2. 特定のコンパイル要件またはその他の COBOL コンパイル環境をサポートするよう、コピーを変更する。
3. (オプション) `-#` オプションを付けて `cob2` コマンドを発行して、変更の影響を表示します。
4. `/etc/cob2.cfg` の代わりにコピーを使用する。

特定のコンパイル用に変更した構成ファイルのコピーを、選択して使用するようになるには、`cob2` (または `cob2_j`、`cob2_r`) コマンドに `-F` オプションを付けて発行します。例えば、`myfile.cbl` をコンパイルするために、構成ファイルとして `/etc/cob2.cfg` ではなく `/u/myhome/myconfig.cfg` を使用するには、次のコマンドを発行します。

```
cob2 myfile.cbl -F/u/myhome/myconfig.cfg
```

`mycob2` など、独自のスタンザを追加する場合、`-F` オプションを使用すれば、その独自のスタンザを指定できます。

```
cob2 myfile.cbl -F/u/myhome/myconfig.cfg:mycob2
```

または、`mycob2` コマンドを定義することができます。

```
ln -s /usr/bin/cob2 /u/myhome/mycob2
mycob2 myfile.cbl -F/u/myhome/myconfig
```

あるいは、`mycob2` を `/etc/cob2.cfg` に追加することができます。

```
ln -s /usr/bin/cob2 mycob2
mycob2 myfile.cbl
```

`ln` コマンドで指定するディレクトリー (上記の `/u/myhome` など) はすべて、`PATH` 環境変数内になければなりません。

各スタンザ内の属性リストについては、スタンザに関する以下の関連参照を参照してください。任意の項目を変更することができます。例えば、リンカーで使用するライブラリーのリストにライブラリーを追加するには、`runlib2` 属性を変更します。

関連タスク

225 ページの『環境変数の設定』

関連参照

245 ページの『`cob2`、`cob2_j`、および `cob2_r` オプション』
『構成ファイル内のスタンザ』

構成ファイル内のスタンザ

次の表に示すように、構成ファイル内のスタンザには、複数ある属性のうちのいずれかを含めることができます。

接尾部に `_64` を持つ属性名は、`cob2` コマンドの `-q64` オプションを指定するときに使用されます。`-q32` が指定されたりデフォルトになっていたりする場合は、対応する接尾部なしの属性名が使用されます。

表 28. スタンザ属性

属性	説明
<code>compiler1</code>	COBOL コンパイラー・コンポーネントのパス名。 <code>cob2</code> のデフォルトは <code>/usr/lpp/cobol/bin/IGYCCOB2</code> です。
<code>crtrt_64</code>	リンカーに最初のパラメーターとして渡されるオブジェクト・ファイルのパス名。 <code>-p</code> または <code>-pg</code> オプションのいずれかを指定しない場合は、 <code>crtrt</code> の値が使用されます。デフォルトは <code>/lib/crt0.o</code> または <code>/lib/crt0_64.o</code> です。
<code>gcrtgcrt_64</code>	リンカーに最初のパラメーターとして渡されるオブジェクト・ファイルのパス名。 <code>-pg</code> オプションを指定すると、 <code>gcrt</code> の値が使用されます。デフォルトは <code>/lib/gcrt0.o</code> または <code>/lib/gcrt0_64.o</code> です。
<code>ld</code>	COBOL プログラムをリンクするために使用されるパス名。デフォルトは <code>/usr/bin/ld</code> です。
<code>ldflags</code> <code>ldflags_64</code>	オプション・フラグのストリング。コンマで区切り、リンカーに渡されます。
<code>mcrtmcrt_64</code>	<code>-p</code> オプションを指定した場合は、リンカーに最初のパラメーターとして渡されるオブジェクト・ファイルのパス名。デフォルトは <code>/lib/mcrt0.o</code> または <code>/lib/mcrt0_64.o</code> です。
<code>proflibs</code>	リンカーで使用する、コンマで区切られたプロファイル・ライブラリー。デフォルトは <code>-l/lib/profiled</code> 、 <code>-l/usr/lib/profiled</code> です。
<code>runlib1</code>	ライブラリー・オプションで、次に <code>runlib2</code> が続きます。コンパイラーはこれをリンカーにパラメーターとして渡して、プロファイルとプロファイル以外の両方のリンク用の COBOL メイン・ライブラリーを指定します。デフォルトは <code>-lcob2s</code> です。
<code>runlib2</code> <code>runlib2_64</code>	リンカーで使用する、コンマで区切られたライブラリー。プロファイルとプロファイル以外の両方に使用します。デフォルトはスタンザに依存します。
<code>use</code>	ローカル・スタンザとは別に、属性をとるスタンザ。単一値の属性では、ローカル、つまりデフォルトのスタンザが提供されていない場合は、 <code>use</code> スタンザ内の値が適用されます。コンマで区切られたリストの場合、 <code>use</code> スタンザからの値がローカル・スタンザからの値に追加されます。

関連タスク

239 ページの『デフォルトのコンパイラ構成の変更』

240 ページの『コンパイルの調整』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

ソース・プログラムのエラーの訂正

ソース・コード・エラーに関するメッセージは、そのエラーが発生した場所 (LINEID) を示します。メッセージのテキストは、どんな問題であるかを知らせます。この情報を使用して、ソース・プログラムを訂正することができます。

エラーを訂正するとしても、すべてのエラーを修正する必要はありません。警告レベルまたは通知レベルのメッセージは、プログラムの中に残っていても支障はなく、それらのエラーを除去するために、再コーディングやコンパイルを行う必要はありません。ただし、重大レベルやエラー・レベルのエラーは、プログラム障害の可能性が大きいため、訂正しなければなりません。

低い方の 4 つのレベルのエラーとは対照的に、回復不能 (U レベル) エラーは、ソース・プログラムの間違いの結果として生じたものではない場合があります。コンパイラ自体あるいはオペレーティング・システム中の欠陥から生じる可能性があります。この場合は、コンパイラが強制的に早期終了させられ、完全なオブジェクト・コードまたはリストを作成しないため、問題を解決するしかありません。多くの S レベルの構文エラーを持つプログラムに関するメッセージが発生した場合には、これらのエラーを訂正し、プログラムを再度コンパイルしてください。また、コンパイル・ジョブに変更を加えることによって、ジョブ・セットアップの問題 (データ・セット定義の欠落やコンパイラ処理用のストレージの不足などの問題) を解決することが可能です。コンパイル・ジョブ・セットアップが正しく、S レベルの構文エラーを訂正した場合は、IBM に連絡して他の U レベル・エラーの調査を要求してください。

ソース・プログラムのエラーを訂正した後で、プログラムを再コンパイルしてください。この 2 回目のコンパイルが成功した場合は、リンク・エディット・ステップに進んでください。コンパイラによってまだ問題が検出される場合は、通知メッセージだけが戻されるようになるまで、上記の手順を繰り返さなければなりません。

関連タスク

243 ページの『コンパイル・エラー・メッセージのリストの生成』

247 ページの『プログラムのリンク』

関連参照

243 ページの『コンパイラ検出エラーに関するメッセージおよびリスト』

コンパイル・エラー・メッセージの重大度コード

コンパイラが検出できるエラーは、重大度に応じて次の 5 つのカテゴリーに分けられます。

表 29. コンパイル・エラー・メッセージの重大度コード

メッセージのレベル	戻りコード	目的
通知 (I)	0	通知にすぎません。アクションを取る必要はありません。プログラムは正しく実行されます。
警告 (W)	4	エラーの可能性あることを示します。プログラムはおそらく、書かれたとおりに正しく実行されます。
エラー (E)	8	明確にエラーである条件があることを意味します。コンパイラーはエラーの訂正を試みましたが、プログラムの実行結果は予期したものではない可能性があります。エラーを訂正しなければなりません。
重大 (S)	12	重大なエラーを示す条件があることを意味します。コンパイラーはエラーを訂正できませんでした。プログラムは正しく実行されず、また、実行を試みてはなりません。オブジェクト・コードは作成されない可能性があります。
回復不能 (U)	16	コンパイルが終了するほどの重大なエラー条件があることを示します。

コンパイル・エラー・メッセージのリストの生成

ERRMSG というプログラム名を持つプログラムをコンパイルすることで、コンパイラー診断メッセージとその説明の完全なリストを生成することができます。

次に示すように、PROGRAM-ID 段落のみをコーディングできます。プログラムの残りの部分は省略します。

```
Identification Division.
Program-ID. ErrMsg.
```

関連参照

『コンパイラー検出エラーに関するメッセージおよびリスト』
244 ページの『コンパイル・エラー・メッセージの形式』

コンパイラー検出エラーに関するメッセージおよびリスト

コンパイラーはソース・プログラムを処理するときに、COBOL 言語を調べてエラーがないかどうかを検査します。エラーが見つかるたびに、コンパイラーはメッセージを出します。これらのメッセージは、コンパイラー・リスト内では (FLAG オプションに従って) 照合されます。

コンパイラー・リスト・ファイルはコンパイラー・ソース・ファイルと同じ名前ですが、サフィックス .lst が付きます。例えば、myfile.cbl のリスト・ファイルは myfile.lst となります。リスト・ファイルは、cob2 (あるいは cob2_j または cob2_r) コマンドを発行したディレクトリーに書き込まれます。

リスト内の各メッセージは、以下の情報を提供します。

- エラーの性質
- エラーを検出したコンパイラー・フェーズ
- エラーの重大度レベル

可能な限り、メッセージはエラーを訂正するための具体的な指示を与えます。

コンパイラ・オプション、CBL および PROCESS ステートメント、および BASIS、COPY、または REPLACE ステートメントの処理中に検出されたエラーに関するメッセージは、リストの上部近くに表示されます。

プログラム (行番号で順序付けされた) の中で検出されたコンパイル・エラーに関するメッセージは、プログラムごとにリストの終わり近くに表示されます。

コンパイル中に検出されたすべてのエラーの要約は、リストの下部に表示されません。

関連タスク

242 ページの『ソース・プログラムのエラーの訂正』

243 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照

『コンパイル・エラー・メッセージの形式』

242 ページの『コンパイル・エラー・メッセージの重大度コード』

286 ページの『FLAG』

コンパイル・エラー・メッセージの形式

コンパイラが発行する各メッセージには、ソース行番号、メッセージ ID、およびメッセージ・テキストが含まれます。

各メッセージの形式は次のとおりです。

nnnnnn IGYppxxxx-l message-text

nnnnnn

コンパイラが処理している最後の行のソース・ステートメントの番号。ソース・ステートメント番号は、プログラムのソース印刷出力にリストされます。コンパイル時に NUMBER オプションを指定すると、それらは元のソース・プログラム番号になります。NONUMBER を指定すると、番号はコンパイラによって生成された番号になります。

IGY このメッセージが COBOL コンパイラから出されたものであることを識別する接頭部。

pp コンパイラのどのフェーズまたはサブフェーズでエラーが発見されたかを識別する 2 文字。アプリケーション・プログラマーはこの情報を無視することができます。コンパイラ・エラーの疑いがあると診断した場合は、IBM にサポートを依頼してください。

xxxx エラー・メッセージを識別する 4 桁の数字。

l エラーの重大度レベルを示す文字 (I、W、E、S、または U)。

message-text

メッセージ・テキスト。エラー・メッセージの場合はエラーの原因となった条件の簡単な説明。

ヒント: FLAG オプションを使用してメッセージを抑止した場合は、プログラム内にさらにエラーがある可能性があることを認識しておいてください。

関連参照

242 ページの『コンパイル・エラー・メッセージの重大度コード』

286 ページの『FLAG』

cob2、cob2_j、および cob2_r オプション

以下にリストされているオプションは、呼び出しコマンド cob2、cob2_j、および cob2_r のすべてに適用されます。

コンパイルに適用されるオプション

-c プログラムをコンパイルしますが、リンクしません。

-comprc_ok=*n*

コンパイラーからの戻りコードに基づいて動作を制御します。戻りコードが *n* 以下であれば、コマンドは継続してリンク・ステップに進むか、またはコンパイルの場合のみの場合には、ゼロの戻りコードで終了します。コンパイラーによって生成された戻りコードが *n* より大きい場合は、コマンドは同じ戻りコードで終了します。

デフォルトは `-comprc_ok=4` です。

-host ホスト COBOL データ表現と言語セマンティクスに対して以下のコンパイラー・オプションを設定します。

- CHAR(EBCDIC)
- COLLSEQ(EBCDIC)
- NCOLLSEQ(BIN)
- FLOAT(S390)

`-host` オプションは、COBOL プログラムのコマンド行引数の形式を、ポインターの配列から、ストリングの長さを含むハーフワード接頭部がある EBCDIC 文字ストリングに変更します。追加情報については、コマンド行引数に関する以下の関連タスクを参照してください。

`-host` オプションを使用するときに、C または PL/I オブジェクト・ファイル内のメインルーチンをアプリケーションのメインの入り口点にしたい場合は、`-cmain` リンカー・オプションを以下で説明するように使用する必要があります。

-lxxx `library-name` も `SYSLIB` も指定されていない場合に、コピーブックの検索に使用するディレクトリーへのパス `xxx` を追加します。(このオプションは、小文字の `l` ではなく大文字の `L` です。)

各 `-I` オプションに使用できるパスは 1 つだけです。複数のパスを追加するには、`-I` オプションを必要な数だけ使用してください。I と `xxx` の間にスペースを入れしないでください。

`COPY` ステートメントを使用する場合には、`LIB` コンパイラー・オプションを必ず有効にしなければなりません。

-qxxx コンパイラーにオプションを渡します。`xxx` には任意のコンパイラー・オプションが入ります。`-q` と `xxx` の間にスペースを入れしないでください。

コンパイラー・オプションまたはサブオプションに括弧が含まれている場合や、一連のオプションが指定されている場合には、各オプションを引用符で囲みます。

複数のオプションを指定する場合は、各オプションを空白またはコンマで区切ります。例えば、次の 2 つのオプション・ストリングは同じです。

```
-qoptiona,optionb  
-q"optiona optionb"
```

シェル・スクリプトを使用して cob2 タスクを自動化する予定の場合は、`-qxxx` オプションに対して特殊な構文が用意されています。詳しくは、シェル・スクリプトでのコンパイルに関する関連タスクを参照してください。

リンクに適用されるオプション

-cmain (`-host` の指定も行う場合のみ効果があります。)メインルーチンを含む C または PL/I オブジェクト・ファイルを、実行可能ファイル内のメインの入り口点にします。C の場合、メインルーチンは関数名 `main()` で識別されます。PL/I の場合、メインルーチンは `PROC OPTIONS(MAIN)` ステートメントで識別されます。

リンクする C または PL/I オブジェクト・ファイルに、1 つ以上の COBOL オブジェクト・ファイルを持つメインルーチンが含まれている場合は、`-cmain` を使用して、C または PL/I ルーチンをメインの入り口点として指定する必要があります。C または PL/I メインルーチンを含む実行可能ファイル内のメインの入り口点として、COBOL プログラムを指定することはできません。これを行うと、予測不能な動作が発生し、診断は行われません。

-main:xxx

`xxx` を、リンカーに渡されるファイルのリスト内の最初のファイルにします。このオプションの目的は、指定したファイルを実行可能ファイル内のメインプログラムにすることです。`xxx` はオブジェクト・ファイルまたはアーカイブ・ライブラリーを一意的に識別する必要があり、接尾部はそれぞれ `.o` または `.a` でなければなりません。

`-main` が指定されていない場合は、コマンド内に指定した最初のオブジェクト、アーカイブ・ライブラリー、またはソース・ファイルが、リンカーに渡されるファイルのリスト内の最初のファイルです。

`-main:xxx` の構文が無効な場合、または `xxx` がコマンドによって処理されるオブジェクト・ファイルまたはソース・ファイルの名前ではない場合は、コマンドが終了します。

-o xxx 実行可能なモジュール `xxx` に名前を付けます。ここで、`xxx` は任意の名前です。`-o` オプションを使用しない場合、実行可能なモジュールの名前はデフォルトの `a.out` に指定されます。

-p コンパイラーが最小のプロファイル情報を含む AIX プロファイル (プログラムのパフォーマンスの評価に使用するコマンド) のために生成するオブジェクト・ファイルを設定するよう、リンカーに指示します。

-pg コンパイラーが最大のプロファイル情報を含む AIX グラフ・プロファイル

(プログラムのパフォーマンスの評価に使用するコマンド) のために生成するオブジェクト・ファイルを設定するよう、リンカーに指示します。

コンパイルとリンクの両方に適用されるオプション

- Fxxx /etc/cob2.cfg 構成ファイルに指定されているデフォルトではなく、構成ファイルまたはスタンザとして xxx を使用します。xxx の形式は以下のいずれかです。
 - *configuration_file:stanza*
 - *configuration_file*
 - *:stanza*
- g デバッガーで使用されるシンボリック情報を生成します。
- q32 32 ビット・オブジェクト・プログラムが生成されることを指定します。ADDR(32) コンパイラー・オプションを設定します。
- q64 64 ビット・オブジェクト・プログラムが生成されることを指定します。ADDR(64) コンパイラー・オプションを設定します。-b64 リンカー・オプションを設定します。このオプションは、64 ビットの実行可能モジュールを作成するようにリンカーに指示します。
- v コンパイルおよびリンクのステップを表示し、各ステップを実行します。
- # コンパイル・ステップとリンク・ステップを表示しますが、それらを実行しません。
- , ? cob2、cob2_j、または cob2_r コマンドのヘルプを表示します。

関連タスク

- 236 ページの『コマンド行からのコンパイル』
- 237 ページの『シェル・スクリプトを使用したコンパイル』
- 248 ページの『リンカーへのオプションの引き渡し』
- 532 ページの『コマンド行引数の使用』

関連参照

- 226 ページの『コンパイラー環境変数』
- 228 ページの『ランタイム環境変数』
- 266 ページの『ADDR』

プログラムのリンク

指定したオブジェクト・ファイルをリンクして、実行可能ファイルまたは共有オブジェクトを作成するには、リンカーを使用します。

リンカーは、次の方法で起動することができます。

- cob2、cob2_j、または cob2_r コマンドの使用:

これらの各コマンドは、-c オプションを指定する場合を除き、リンカーを呼び出します。アプリケーションのタイプによってどのコマンドを使用するかについては、コマンド行からのコンパイルに関する関連タスクを参照してください。

cob2_j および cob2_r は、COBOL マルチスレッド・ライブラリーとリンクします。さらに、cob2_j は Java Native Interface (JNI) ランタイム・ライブラリーとリンクします。cob2 は、COBOL 静的ランタイム・ライブラリーとリンクしません。

C とのリンク: リンカーは、.o ファイルは受け入れますが、.c ファイルは受け入れません。C と COBOL のファイルをまとめてリンクしたい場合は、xlc -c コマンドを使用して、最初に C ソース・ファイル用の .o ファイルを作成します。

- 1d コマンドの使用:

リンカーは、1d コマンドを使用して明示的に呼び出すことができます。ただし、コンパイラー呼び出しコマンドは、いくつかのリンカー・オプションとライブラリー名を、自動的にリンカーに渡します。ほとんどの場合は、コンパイラー呼び出しコマンドのいずれかを使用して、.o ファイルをリンク・エディットすることをお勧めします。

- Make ファイルの使用:

Make ファイルを作成して、プログラムのビルドに必要な一連の操作 (コンパイルやリンクなど) を編成することができます。Make ファイルでリンカー・ステートメントを使用して、希望する出力の種類を指定することができます。

リンカー・オプションは、上記のいずれかの方法を使用して指定することができます。

250 ページの『例: リンクでの呼び出しコマンドの使用』

関連タスク

『リンカーへのオプションの引き渡し』

236 ページの『コマンド行からのコンパイル』

535 ページの『第 28 章 共用ライブラリーの構築』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

251 ページの『リンカーの入出力ファイル』

252 ページの『リンカーの検索規則』

615 ページの『付録 A. 移行および互換性』

AIX プログラミングの一般概念: プログラムの作成およびデバッグ (make)

リンカーへのオプションの引き渡し

呼び出しコマンド (cob2、cob2_j、または cob2_r)、1d コマンド、または Makefile ステートメントのいずれにおいても、リンカー・オプションを指定できます。

呼び出しコマンドで指定するオプションのうち、このコマンドで認識されないものは、リンカーに渡されます。

呼び出しコマンド・オプションによっては、プログラムのリンクに影響するものがあります。特定のコマンドのオプション・セットについては、そのコマンドの資料を調べてください。使用可能なリンカー・オプションの完全なセットについて詳しくは、1d コマンドに関する関連参照を参照してください。

以下の表は、COBOL プログラムのために必要になることが多いと思われるオプションを示しています。ここに示されているように、各オプションの前にはハイフン (-) を付けます。

表 30. 共通リンカー・オプション

オプション	目的
-bdynamic、-bshared、または -bstatic	-l オプションの処理と共有オブジェクトの処理方法を制御します。
-bD:nnnn	リンカーに nnnn バイトの静的ユーザー・データ用のヒープ・スペースを割り振るように指示します。(デフォルト値は 0 です。) このスペースは、非常に大きなデータ項目 (特に WORKING-STORAGE SECTION のデータ項目) が定義されたプログラムを実行するために必要になる場合があります。
-bexpall	すべてのグローバル・シンボルをエクスポートしますが、インポートされたシンボル、アーカイブ・メンバーに定義されている参照されないシンボル、および下線 (_) で始まるシンボルを除きます。このオプションを使用するときは、エクスポート・ファイルを使用しないようにすることもできます。
-bE:foo.exp	エクスポート・ファイル foo.exp に指定されたシンボルをエクスポートします
-bhalt:error_level	リンカー・コマンドの処理が停止する前に許可されている、最大エラー・レベルを指定します
-bI:foo.imp または -bimport:foo.imp	インポート・ファイル foo.imp から記号をインポートします
-bloadmap:name	リンカー・アクションとメッセージのログがファイル name に保管されるよう要求します
-bmap:name	出力オブジェクト・ファイルのアドレス・マップを、ファイル name に書き込みます
-bmaxdata:bytes or -bmaxstack:bytes	領域のサイズが制約であるプログラムのプログラム・データ・セグメントとスタック・セグメント用に予約する最大容量を指定します
-bM:SRE	出力モジュールを共有の再入可能オブジェクトにします
-bnoentry	このオブジェクトがメインの入り口点を持たないことを示します
-bns0	共有オブジェクトを通常オブジェクト・ファイルとして静的にリンクします
-brtl または -bnortl	-l オプションを使用して指定するライブラリーを検索するために、どのアルゴリズムを使用するかを決定します
-Ldir	-l オプションで指定したライブラリーを検索するためのディレクトリーを指定します (デフォルト: /usr/lib)
-lname	指定した library-file を検索します。name は、ファイル libname.a を選択します (デフォルト: cob2.cfg にリストされているライブラリー)
-Snnnn	nnnn バイトのユーザー・スタックを割り振るようにリンカーに指示します。(デフォルト値は 0 です。) このスペースは、非常に大きなデータ項目 (特に LOCAL-STORAGE SECTION のデータ項目) が定義されたプログラムを実行するために必要になる場合があります。

『例: リンクでの呼び出しコマンドの使用』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

251 ページの『リンカーの入出力ファイル』

252 ページの『リンカーの検索規則』

AIX Commands Reference, Volume 3 (ld command)

例: リンクでの呼び出しコマンドの使用

COBOL 呼び出しコマンドを使用して、プログラムのコンパイルとリンクを両方とも実行することができます。次の例では、cob2 の使用法を示します。

- 2 つのファイルをコンパイルした後でそれらのファイルをまとめてリンクするとき、-c オプションは使用しないでください。例えば、alpha.cbl および beta.cbl をコンパイルしてリンクし、a.out を生成するには、次のように入力します。

```
cob2 alpha.cbl beta.cbl
```

このコマンドは、alpha.o と beta.o を作成し、その後に alpha.o、beta.o、および COBOL ライブラリーをリンクします。リンク・ステップが成功すると、a.out という名前の実行可能プログラムが作成されます。

- コンパイルしたファイルをライブラリーとリンクするには、次のように入力します。

```
cob2 foo.cbl -brt1 -bshared -lmylib
```

このコマンドによりリンカーは、検索パス内の各ディレクトリーで最初に libmylib.so を検索し、次にライブラリー libmylib.a を検索します。これは、いずれかが検出されるまで続きます。

- オプションを使用してライブラリーの検索を制限するには、次のように入力します。

```
cob2 foo.cbl -bdynamic -llib1 -bstatic -llib2 -brt1
```

この場合、最初のライブラリー指定を満たすために、リンカーによって各ディレクトリーでライブラリー liblib1.so first が最初に検索され、次にライブラリー liblib1.a が検索されます (前の例を参照)。ただし、同時にリンカーは、それらの同じライブラリー内の liblib2.a を検索するだけです。

- 呼び出しコマンド行で -b リンカー・オプションを使用して、静的にリンクされたオブジェクト・ファイルを作成するには、次のように入力します。

```
cob2 -bnso -bI:/usr/lib/syscalls.exp file1.cbl file2.cbl
```

- コンパイルとリンクを別々のステップで実行するには、次のようなコマンドを入力します。

```
cob2 -c file1.cbl          # Produce one object file
cob2 -c file2.cbl file3.cbl # Or multiple object files
cob2 file1.o file2.o file3.o # Link with appropriate libraries
```

- beta.cbl および gamma.cbl を sh_bg という共用ライブラリー・オブジェクト内に結合するには、次のように入力します。

```
cob2 -o sh_bg beta.cbl gamma.cbl -bexpall -bM:SRE -bnoentry
```

リンカーの入出力ファイル

リンカーはオブジェクト・ファイルどうしをリンクし、ライブラリー・ファイルが指定されている場合は実行可能出力ファイルを生成します。実行可能出力は、実行可能プログラム・ファイルか共有オブジェクトのいずれかになります。

リンカー入力

- オプション
- オブジェクト・ファイル (*.o)
- 静的ライブラリー・ファイル (*.a)
- 動的ライブラリー・ファイル (*.so)
- インポート・ファイル (*.imp)

リンカー出力

- 実行可能ファイル (デフォルトでは a.out)
- 共有オブジェクト
- 戻りコード

インポート・ファイル: インポート・ファイルは、実行時に解決する外部シンボルを識別する ASCII ファイルです。インポート・ファイルは、インポートされたシンボルを定義する共有オブジェクトを識別します。システム・ローダーは、このシンボルを検出して解決します。インポート・ファイルの最初の行が #! (ポンド記号、感嘆符) で始まる場合は、そのファイルをコマンド行で通常の入力ファイルとして指定することができます。それ以外の場合は、`-bI` または `-bimport` オプションを使用して、インポート・ファイルを指定する必要があります。

エクスポート・ファイル: エクスポート・ファイルは、インポートする別の実行可能かのオブジェクト・ファイルで使用できる外部シンボルを識別する ASCII ファイルです。エクスポート・ファイルのファイル・フォーマットは、インポート・ファイルのファイル・フォーマットと同じです。

ライブラリー・ファイル: ライブラリーは、接尾部が `.a` または `.so` のファイルです。ライブラリーを指定するには、絶対または相対パス名を指定するか、`-l` (小文字の L) オプションを `-lname` の形式で使用することができます。後者の形式は、ファイル `libname.a` を指定し、動的モードの場合はファイル `libname.so` を指定して、複数のディレクトリー内で検索されるようにします。これらの検索ディレクトリーには、`-L` オプションを使用して指定するディレクトリーと、標準ライブラリー・ディレクトリ `/usr/lib` および `/lib` が含まれます。

環境変数 `LIBPATH` は、コマンド行でいずれかを明示的に指定するライブラリー (例えば、`libc.a`) の検索や、`-l` オプション (例えば、`-lc`) を使用したライブラリーの検索には使用しません。`-l` オプションを使用して指定したライブラリーを検索するディレクトリーを示すには、`-Ldir` オプションを使用する必要があります。

共有オブジェクト、または共有オブジェクトを含むアーカイブ・ファイルを、`-lName` オプションではなく絶対または相対パス名を使用して指定すると、その

パス名が出力ファイルのローダー・セクション内のインポート・ファイルの ID ストリングに組み込まれます。この動作は、`-bnoipath` オプションを使用してオーバーライドすることができます。

AIX `ar` コマンドを使用して、単一のアーカイブ・ファイル内に 1 つ以上のファイルを結合してライブラリー・ファイルを作成することができます。

537 ページの『例: サンプルの共用ライブラリーの作成』

関連タスク

248 ページの『リンカーへのオプションの引き渡し』

関連参照

『リンカーの検索規則』

253 ページの『リンカー・ファイル名のデフォルト』

AIX Commands Reference, Volume 1 (ar command)

リンカーの検索規則

リンカーは、オブジェクト (`.o`)、ライブラリー (`.a`)、またはインポート・ファイル (`.imp`) を検索するとき、その検索内容を満たすまで複数の場所を探します。

リンカーが検索する場所は次のとおりです。

1. ファイルに対して指定するディレクトリー

ファイルとともにパスを指定した場合、リンカーはそのパスだけを検索し、そこでファイルが検出されなければリンクを停止します。

2. パスを指定しなかった場合は、現行ディレクトリー

3. 環境変数 `LIBPATH` の値 (定義されている場合)

ファイルの検索は、`ld` オプションによって異なります。 `-brtl` を指定する場合は、ランタイム・リンクが使用可能です。 `-bdynamic` または `-bshared` のいずれかとともに使用する場合は、 `-l` オプションを使用して指定するライブラリーの検索は、接尾部 `.so` または `.a` によって行われます。検索場所となる各ディレクトリーで接尾部 `.so` のファイルが検索されます。この検索が失敗すると、接尾部 `.a` のファイルが検索されます。どちらも見つからなければ、検索は次のディレクトリーで続行します。

`/usr/lib` 内のデフォルトのライブラリー以外のライブラリーを使用する場合は、その他のライブラリーの場所を示す `-L` オプションを 1 つ以上指定することができます。また、`LIBPATH` 環境変数を設定することもできます。これにより、実行時にライブラリーの検索パスを指定することができます。

リンカーがファイルを検出できない場合は、エラー・メッセージを生成してリンクを停止します。

関連タスク

248 ページの『リンカーへのオプションの引き渡し』

関連参照

『リンカー・ファイル名のデフォルト』

AIX プログラミングの一般概念: プログラムの作成およびデバッグ

リンカー・ファイル名のデフォルト

ファイル名を入力しない場合、リンカーはデフォルト名を想定します。

表 31. リンカーで想定されるデフォルトのファイル名

ファイル	デフォルトのファイル名	デフォルトの接尾部
オブジェクト・ファイル	なし。少なくとも 1 つのオブジェクト・ファイル名を入力する必要があります。	.o
出力ファイル	a.out	なし
マップ・ファイル	出力ファイルのベース名	マップ
ライブラリー・ファイル	オブジェクト・ファイルで定義されたデフォルトのライブラリー。デフォルトのライブラリーを定義するには、コンパイラー・オプションを使用します。追加のライブラリーを指定した場合は、デフォルトのライブラリーの前にそれらが検索されます。	.a or .so
インポート・ファイル	なし。インポート・ファイルの名前を明示的に入力してください。	.imp
エクスポート・ファイル	なし。エクスポート・ファイルの名前を明示的に入力してください。	.exp

リンク内のエラーの訂正

PGMNAME (UPPER) コンパイラー・オプションを使用すると、CALL ステートメントで参照されるサブプログラムの名前が大文字に変換されます。リンカーは大/小文字を区別して名前を認識するため、この変換はリンカーに影響します。

例えば Call "RexxStart" は、コンパイラーによって Call "REXXSTART" に変換されます。呼び出し先プログラムの実名が `RexxStart` の場合、リンカーはこのプログラムを検出できず、REXXSTART が未解決の外部参照であることを示すエラー・メッセージを生成します。

このタイプのエラーは一般に、他のソフトウェア・プロダクトに備わっている API ルーチン呼び出し場合に起こります。API ルーチンの名前に大文字と小文字が混在している場合は、次の両方の処置を行ってください。

- PGMNAME (MIXED) コンパイラー・オプションを使用する。
- CALL ステートメントで、API ルーチンの名前が正しく (大/小文字が正確に) 指定されていることを確認する。

プログラムの実行

COBOL プログラムを実行するには、環境変数を設定し、プログラムを実行して、実行時エラーを訂正します。

1. 必要な環境変数がすべて設定されていることを確認します。

例えば、システム・ファイル名に値を割り当てる環境変数名をプログラムが使用する場合は、その環境変数を設定する必要があります。

2. プログラムを実行します。コマンド行で、実行可能モジュールの名前を入力するか、そのモジュールを呼び出すコマンド・ファイルを実行します。

例えば、コマンド `cob2 alpha.cb1 beta.cb1-o gamma` が成功した場合は、`gamma` を入力してプログラムを実行することができます。

3. ランタイム・エラーを訂正します。

例えば、デバッガーを使用して、エラーの発生時点でプログラム状態を調べることができます。

ヒント: ランタイム・メッセージが省略されているか不完全な場合は、環境変数 `LANG` または `NLSPATH`、あるいはその両方の設定が間違っている可能性があります。

関連タスク

225 ページの『環境変数の設定』

341 ページの『デバッガーの使用』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

228 ページの『ランタイム環境変数』

615 ページの『付録 A. 移行および互換性』

781 ページの『付録 J. ランタイム・メッセージ』

COBOL for AIX ライブラリー・コンポーネントの再配布

開発したアプリケーションを COBOL for AIX ライブラリー・コンポーネントに付けて再配布することができます。COBOL アプリケーションの実動配置用に使用するコンポーネントは、任意の数の AIX システム上にインストールできます。

再配布されるアプリケーションの開発に使用する AIX システム上に、完全な COBOL for AIX 製品のライセンス交付を受けたコピーをインストールする必要があります。COBOL for AIX ファイルセットのうち、追加ライセンスを必要とせずに再配布できるのは次のファイルセットです。

- `cobol.rte` (ランタイム・ライブラリー)
- `cobol.msg.Ja_JP` (日本語のコンパイラー・メッセージおよびランタイム・メッセージ)
- `cobol.msg.ja_JP` (日本語のコンパイラー・メッセージおよびランタイム・メッセージ)

- `cobol.msg.en_US` (英語のコンパイラー・メッセージおよびランタイム・メッセージ)
- `SdU.vsam` (SMARTdata UTILITIES VSAM)
- `SdU.sfs` (SMARTdata UTILITIES SFS)

これらのファイルセットは製品 CD にあります。

上記のリストにあるファイルセットのいずれかに関連付けられているファイル名を参照するには、`ls1pp` コマンド (例: `ls1pp -f cobol.rte`) を発行します。

第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行

オブジェクト指向 (OO) アプリケーションのコンパイル、リンク、および実行は、以下に示す説明に従って、コマンド行で行うことができます。

関連概念

261 ページの『Java SDK サポート』

関連タスク

『オブジェクト指向アプリケーションのコンパイル』

258 ページの『オブジェクト指向アプリケーションの準備』

260 ページの『オブジェクト指向アプリケーションの実行』

オブジェクト指向アプリケーションのコンパイル

オブジェクト指向アプリケーションをコンパイルする場合、cob2_j コマンドを使用して、COBOL クライアント・プログラムおよびクラス定義をコンパイルし、javac コマンドを使用して、Java クラス定義をコンパイルし、バイトコード (サフィックス .class) を生成します。

cob2_j コマンドを使用してコンパイルした場合には、THREAD コンパイラー・オプションがデフォルトの起動オプションとして自動的に組み込まれます。また、Java Native Interface (JNI) ライブラリーがリンカーに渡されるライブラリー・リストに組み込まれます。(ただし、使用している Java のリリース・レベルのライブラリーに更新する必要があることがあります。Java SDK サポートについては、関連概念を参照してください。デフォルトのコンパイラー構成の変更方法については、関連タスクを参照してください。)

クラス定義を含む COBOL ソース・ファイルは、他のクラスまたはプログラム定義を含むことはできません。

COBOL クラス定義をコンパイルすると、2 つの出力ファイルが生成されます。

- クラス定義のオブジェクト・ファイル (.o)。
- COBOL クラス定義に対応するクラス定義を含む Java ソース・プログラム (.java)。この生成済みの Java クラス定義は、決して編集しないでください。COBOL クラス定義を変更する場合は、更新された COBOL クラス定義を再コンパイルして、オブジェクト・ファイルと Java クラス定義の両方を再生成しなければなりません。

COPY ステートメントを使用して COBOL クライアント・プログラムまたはクラス定義にファイル JNI.cpy を組み込む場合は、COBOL インストール・ディレクトリー (通常 /usr/lpp/cobol/include) の include サブディレクトリーをコピーブックの検索順序に指定します。cob2_j コマンドの -I オプションを使用するか、SYSLIB 環境変数を設定することによって、include サブディレクトリーを指定することができます。

関連概念

261 ページの『Java SDK サポート』

関連タスク

225 ページの『環境変数の設定』
235 ページの『コンパイル済みプログラム』
239 ページの『デフォルトのコンパイラ構成の変更』
『オブジェクト指向アプリケーションの準備』
260 ページの『オブジェクト指向アプリケーションの実行』
477 ページの『JNI サービスへのアクセス』

関連参照

226 ページの『コンパイラ環境変数』
245 ページの『cob2、cob2_j、および cob2_r オプション』
311 ページの『THREAD』

オブジェクト指向アプリケーションの準備

オブジェクト指向 COBOL アプリケーションをリンクするには、cob2_j コマンドを使用します。cob2_j を使用してリンクした場合には、スレッド対応ライブラリーがアプリケーションのリンクに使用されます。

拡張のために OO COBOL クライアント・プログラムを準備するには、オブジェクト・ファイルをリンクして、共用ライブラリーを作成します。

COBOL クラス定義を実行できるように準備する。

1. オブジェクト・ファイルをリンクし、共用ライブラリーを作成します。

結果として生成される共用ライブラリーの名前は、libClassname.so にする必要があります。ここで、Classname は外部クラス名です。クラスがパッケージの一部であり、外部クラス名にピリオド (.) が使用されている場合は、共用ライブラリー名ではピリオドを下線に変更する必要があります。例えば、Account クラスが com.acme パッケージの一部である場合、外部クラス名 (クラスの REPOSITORY 段落記入項目に定義されている) は com.acme.Account であり、このクラスの共用ライブラリー名は libcom_acme_Account.so でなければなりません。

2. 生成された Java ソースを Java コンパイラでコンパイルして、クラス・ファイル (.class) を生成します。

Classname に対応するクラス定義が格納されている COBOL ソース・ファイル Classname.cbl の場合は、次のコマンドを使用して、アプリケーションのコンポーネントのコンパイルとリンクを行います。

表 32. クラス定義のコンパイルおよびリンクのコマンド

コマンド	入力	出力
cob2_j -c Classname.cbl	Classname.cbl	Classname.o, Classname.java
cob2_j -o libClassname.so -bnoentry -bexpall -bM:SRE Classname.o	Classname.o	libClassname.so
javac Classname.java	Classname.java	Classname.class

cob2_j コマンドも、ライブラリー libjava.a をオブジェクト・ファイルにリンクさせます。

cob2_j および javac コマンドが正常に発行されると、プログラムの実行可能コンポーネントである、共用ライブラリー libClassname.so およびクラス・ファイル Classname.class が生成されます。これらのコマンドによって生成されるファイルは、すべて現行作業ディレクトリーに置かれます。

『例: COBOL クラス定義のコンパイルとリンク』

関連タスク

235 ページの『コンパイル済みプログラム』

433 ページの『クラス定義用の REPOSITORY 段落』

248 ページの『リンカーへのオプションの引き渡し』

535 ページの『第 28 章 共用ライブラリーの構築』

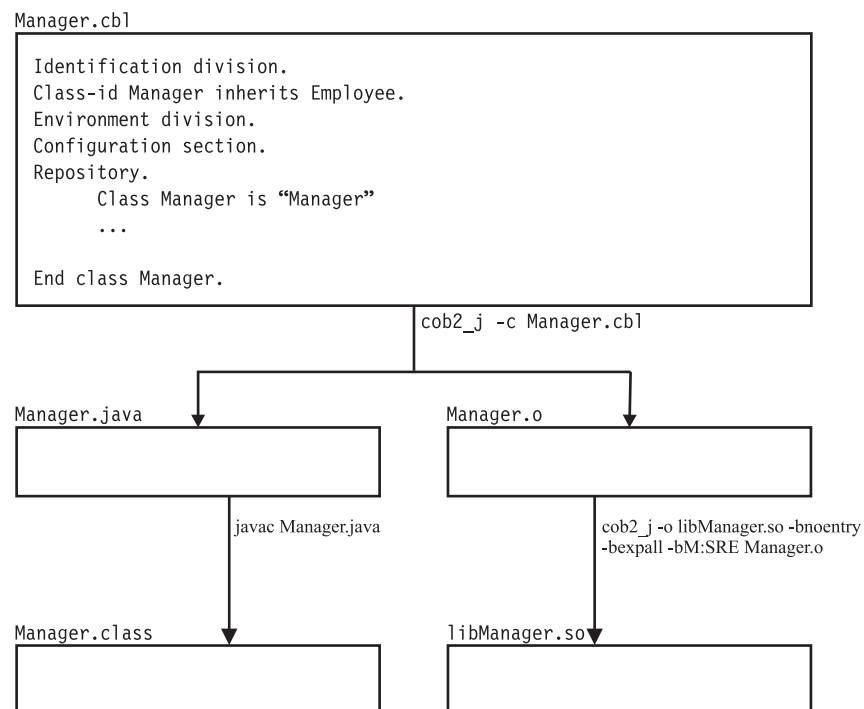
関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

615 ページの『オブジェクト指向構文、および Java 5 または Java 6 SDK』

例: COBOL クラス定義のコンパイルとリンク

次の例は、Manager.cbl という COBOL クラス定義をコンパイルおよびリンクするときに使用するコマンドと生成されるファイルを示したものです。



クラス・ファイル `Manager.class` と共用ライブラリー `libManager.so` は、アプリケーションの実行可能コンポーネントであり、現行作業ディレクトリーに生成されま
す。

関連タスク

248 ページの『リンカーへのオプションの引き渡し』

オブジェクト指向アプリケーションの実行

コマンド行からオブジェクト指向 COBOL アプリケーションを実行することができます。

COBOL クラスのための 共用ライブラリーが置かれているディレクトリーを `LIBPATH` 環境変数で指定します。また、それらの COBOL クラスと関連付けられた Java クラス・ファイルが置かれているディレクトリー・パスを次のように `CLASSPATH` 環境変数で指定します。

- パッケージの一部ではないクラスの場合は、`.class` ファイルが収められているディレクトリーでクラス・パスを終了します。
- パッケージの一部であるクラスの場合は、「root」パッケージ (完全なパッケージ名における最初のパッケージ) が収められているディレクトリーでクラス・パスを終了します。
- `.class` ファイルを含む `.jar` ファイルの場合は、`.jar` ファイルの名前でクラス・パスを終了します。

複数のパス・エントリーはコロンで区切ります。

関連タスク

『main メソッドで始まるオブジェクト指向アプリケーションの実行』

261 ページの『COBOL プログラムで始まるオブジェクト指向アプリケーションの実行』

254 ページの『プログラムの実行』

225 ページの『環境変数の設定』

427 ページの『第 23 章 オブジェクト指向プログラムの作成』

473 ページの『OO アプリケーションの構造化』

main メソッドで始まるオブジェクト指向アプリケーションの実行

COBOL と Java の混合アプリケーションの最初のルーチンが、Java クラスの `main` メソッドまたは COBOL クラスの `main` ファクトリー・メソッドである場合は、`java` コマンドを使用し、`main` メソッドを含むクラスの名前を指定して、アプリケーションを実行します。

`java` コマンドは、Java 仮想マシン (JVM) を初期化します。JVM の初期化をカスタマイズするには、以下の例で示すように、`java` コマンドのオプションを指定します。

表 33. JVM をカスタマイズするための Java コマンド・オプション

目的	オプション
システム・プロパティを指定する	<code>-Dname=value</code>
ガーベッジ・コレクションについての詳しいメッセージを JVM が生成するよう要求する	<code>-verbose:gc</code>
クラス・ロードについての詳しいメッセージを JVM が生成するよう要求する	<code>-verbose:class</code>
ネイティブ・メソッドおよび他の Java ネイティブ・インターフェース・アクティビティについての詳しいメッセージを JVM が生成するよう要求する	<code>-verbose:jni</code>
Java の初期ヒープ・サイズを <i>value</i> バイトに設定する	<code>-Xmsvalue</code>
Java の最大ヒープ・サイズを <i>value</i> バイトに設定する	<code>-Xmxvalue</code>

JVM がサポートするオプションの詳細については、`java -h` コマンドの出力または関連参照を参照してください。

関連参照

Java Technology Edition SDK User Guides

COBOL プログラムで始まるオブジェクト指向アプリケーションの実行

COBOL と Java の混合アプリケーションの先頭のルーチンが COBOL プログラムである場合は、コマンド・プロンプトでプログラム名を指定してアプリケーションを実行します。COBOL プログラムのプロセスで JVM がまだ実行されていない場合は、COBOL のランタイムが JVM を自動的に初期化します。

JVM の初期化をカスタマイズするには、`COBJVMINITOPTIONS` 環境変数を設定してオプションを指定します。オプションを区切るには、ブランクを使用します。以下に例を示します。

```
export COBJVMINITOPTIONS="-Xms10000000 -Xmx20000000 -verbose:gc"
```

関連タスク

254 ページの『プログラムの実行』

225 ページの『環境変数の設定』

関連参照

Java Technology Edition SDK User Guides

Java SDK サポート

オブジェクト指向構文を持つアプリケーションをコンパイルするための `cob2_j` コマンドは、そのアプリケーションを Java SDK 1.4 ライブラリーにデフォルトでリンクします。

それ以降の Java SDK (Java 5 や Java 6 など) を使用するには、デフォルト・コンパイラ構成ファイル `cob2.cfg` の `cob2_j` スタンザを修正して、上位の Java ライブラリーのパス名を示すようにします。

| オブジェクト指向構文を使用する 64 ビット COBOL アプリケーションは、64 ビット Java SDK を必要とします。オブジェクト指向構文を使用する 32 ビット COBOL アプリケーションは、32 ビット SDK を必要とします。64 ビット SDK ライブラリーを取得するには、cob2_j コマンドの -q64 オプションを指定します。-q32 オプションを指定した場合、32 ビット SDK ライブラリーが選択されます。

| **関連タスク**

| 239 ページの『デフォルトのコンパイラー構成の変更』

| **関連参照**

| 245 ページの『cob2、cob2_j、および cob2_r オプション』

| 266 ページの『ADDR』

| 615 ページの『オブジェクト指向構文、および Java 5 または Java 6 SDK』

第 14 章 コンパイラー・オプション

コンパイルに対する指示および制御の方法には、コンパイラー・オプションを使用する方法と、コンパイラー指示ステートメント (コンパイラー指示) を使用する方法とがあります。

コンパイラー・オプションは、次の表にリストされているプログラムの局面に影響を与えます。各オプションに結び付けられている情報は、そのオプションを指定するための構文を与えるもので、オプション、そのパラメーター、および他のパラメーターとの相互作用を説明しています。

表 34. コンパイラー・オプション

プログラムの局面	コンパイラー・オプション	デフォルト	オプションの省略形
ソース言語	268 ページの『ARITH』	ARITH(COMPAT)	AR(C E)
	273 ページの『CICS』	NOCICS	なし
	277 ページの『CURRENCY』	NOCURRENCY	CURR NOCURR
	290 ページの『LIB』	LIB	なし
	296 ページの『NSYMBOL』	NSYMBOL(NATIONAL)	NS(NAT DBCS)
	297 ページの『NUMBER』	NONUMBER	NUM NONUM
	301 ページの『QUOTE/APOST』	QUOTE	Q APOST
	303 ページの『SEQUENCE』	SEQUENCE	SEQ NOSEQ
	305 ページの『SOSI』	NOSOSI	なし
	308 ページの『SQL』	SQL("")	なし
日付処理	278 ページの『DATEPROC』	NODATEPROC、または DATEPROC(FLAG) (DATEPROC だけが指定された場合)	DP NODP
	317 ページの『YEARWINDOW』	YEARWINDOW(1900)	YW
マップおよびリスト	291 ページの『LINECOUNT』	LINECOUNT(60)	LC
	291 ページの『LIST』	NOLIST	なし
	292 ページの『LSTFILE』	LSTFILE(LOCALE)	LST
	293 ページの『MAP』	NOMAP	なし
	306 ページの『SOURCE』	SOURCE	S NOS
	307 ページの『SPACE』	SPACE(1)	なし
	310 ページの『TERMINAL』	TERMINAL	TERM NOTERM
	315 ページの『VBREF』	NOVBREF	なし
	316 ページの『XREF』	XREF(FULL)	X NOX
オブジェクト・モジュールの生成	276 ページの『COMPILE』	NOCOMPILE(S)	C NOC
	298 ページの『OPTIMIZE』	NOOPTIMIZE	OPT NOOPT
	299 ページの『PGMNAME』	PGMNAME(UPPER)	PGMN(LU LM)
	302 ページの『SEPOBJ』	SEPOBJ	なし

表 34. コンパイラー・オプション (続き)

プログラムの局面	コンパイラー・オプション	デフォルト	オプションの省略形
I オブジェクト・コード制御	266 ページの『ADDR』	ADDR(32)	なし
	269 ページの『BINARY』	BINARY(NATIVE)	なし
	271 ページの『CHAR』	CHAR(NATIVE)	なし
	274 ページの『COLLSEQ』	COLLSEQ(BIN)	なし
	279 ページの『DIAGTRUNC』	NODIAGTRUNC	DTR NODTR
	289 ページの『FLOAT』	FLOAT(NATIVE)	なし
	296 ページの『NCOLLSEQ』	NCOLLSEQ(BINARY)	NCS(L BIN B)
	312 ページの『TRUNC』	TRUNC(STD)	なし
	318 ページの『ZWB』	ZWB	なし
CALL ステートメントの動作	280 ページの『DYNAM』	NODYNAM	DYN NODYN
デバッグと診断	286 ページの『FLAG』	FLAG(I,I)	F NOF
	288 ページの『FLAGSTD』	NOFLAGSTD	なし
	309 ページの『SSRANGE』	NOSSRANGE	SSR NOSSR
	310 ページの『TEST』	NOTEST	なし
I その他	266 ページの『ADATA』	NOADATA	なし
	270 ページの『CALLINT』	CALLINT(SYSTEM,NODESC)	なし
	280 ページの『ENTRYINT』	ENTRYINT(SYSTEM)	なし
	281 ページの『EXIT』	NOEXIT	EX(INX,LIBX,PRTX,ADX)
	294 ページの『MAXMEM』	MAXMEM(2000)	なし
	295 ページの『MDECK』	NOMDECK	NOMD MD MD(C) MD(NOC)
	301 ページの『PROFILE』	PROFILE	なし
	304 ページの『SIZE』	SIZE(8388608)	SZ
	308 ページの『SPILL』	SPILL(512)	なし
	311 ページの『THREAD』	NOTHREAD	なし
	315 ページの『WSCLEAR』	NOWSCLEAR	なし

インストール先のデフォルト: 上記のオプションとともにリストされているデフォルトは、製品出荷時のデフォルトです。

パフォーマンスに関する考慮事項: ADDR、ARITH、CHAR、DYNAM、FLOAT、OPTIMIZE、SSRANGE、TEST、および TRUNC の各コンパイラー・オプションは、実行時のパフォーマンスに影響を及ぼす可能性があります。

関連タスク

- 235 ページの『コンパイル済みプログラム』
- 587 ページの『第 32 章 プログラムのチューニング』

関連参照

- 319 ページの『第 15 章 コンパイラー指示ステートメント』
- 597 ページの『パフォーマンスに関連するコンパイラー・オプション』

矛盾するコンパイラー・オプション

COBOL for AIX コンパイラーは、次の 2 つのいずれかの場合に、矛盾するコンパイラー・オプションに遭遇することがあります。つまり、同じオプションの肯定形式と否定形式の両方がオプションの優先順位の階層の中で同じレベルで指定されている場合、または相互に排他的なオプションが同じレベルで指定されている場合です。

コンパイラーは、次の優先順位 (高位から下位の順) に従ってオプションを認識します。

1. PROCESS (または CBL) ステートメントで指定されたオプション
2. cob2、cob2_j、または cob2_r コマンド呼び出しで指定されたオプション
3. COBOPT 環境変数のオプション・セット
4. デフォルト・オプション

矛盾するオプションを階層の同じレベルで指定した場合は、最後に指定したオプションの方が有効になります。

互いに排他的なコンパイラー・オプションを同じレベルで指定すると、コンパイラーはオプションの一方を対立しない値に強制し、エラー・メッセージを生成します。例えば、PROCESS ステートメントで OPTIMIZE と TEST の両方を指定すると、次の表に示すように、指定した順序に関係なく TEST が有効になり、OPTIMIZE は無視されます。

表 35. 相互に排他的なコンパイラー・オプション

指定される	無視される	強制
CICS	ADDR(64)	ADDR(32)
	DYNAM	NODYNAM
	NOLIB	LIB
	THREAD	NOTHREAD
MDECK	NOLIB	LIB
SQL	NOLIB	LIB
TEST	OPTIMIZE	NOOPTIMIZE

しかし、高レベルの優先順位で指定されたオプションは、低レベルの優先順位で指定されたオプションをオーバーライドします。例えば、COBOPT 環境変数では TEST をコーディングし、PROCESS ステートメントでは OPTIMIZE をコーディングすると、PROCESS ステートメントでコーディングされたオプションと、PROCESS ステートメントで強制的にオンにされたオプションの優先順位の方が高いため、OPTIMIZE が有効になります。

関連タスク

235 ページの『コンパイル済みプログラム』

関連参照

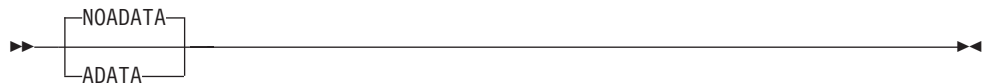
226 ページの『コンパイラー環境変数』

245 ページの『cob2、cob2_j、および cob2_r オプション』
319 ページの『第 15 章 コンパイラー指示ステートメント』

ADATA

ADATA は、コンパイラーに追加のコンパイル情報のレコードを収めた SYSADATA ファイルを作成させる場合に使用します。

ADATA オプションの構文



デフォルト: NOADATA

省略形: なし

SYSADATA 情報は、使用する目的で ADATA ON を設定する、他のツールによって使用されます。

SYSADATA ファイルのサイズは、通常、関連するプログラムのサイズに比例します。

オプションの指定: PROCESS (または CBL) ステートメントで ADATA オプションを指定することはできません。指定できるのは、以下のいずれかの方法に限られます。

- cob2 コマンドまたはそのバリエーションのいずれかのオプションとして
- COBOPT 環境変数

関連参照

711 ページの『付録 I. COBOL SYSADATA ファイルの内容』

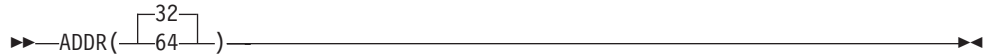
226 ページの『コンパイラー環境変数』

245 ページの『cob2、cob2_j、および cob2_r オプション』

ADDR

32 ビットのオブジェクト・プログラムを生成するのか、64 ビットのオブジェクト・プログラムを生成するのかを指示するには、ADDR コンパイラー・オプションを使用します。

ADDR オプションの構文



デフォルト: ADDR(32)

省略形: なし

オプション指定:

他のコンパイラー・オプションを指定するいずれかの方法で ADDR オプションを指定できます。プログラムのコンパイルについては、下記の関連タスクに説明があります。ただし、PROCESS (または CBL) ステートメントで ADDR を指定する場合、バッチ・コンパイルの最初のプログラムに対してのみ指定できます。バッチでは、後続のプログラムに対して ADDR オプションの値を変更することはできません。

cob2 コマンド (あるいは、そのバリエーションである cob2_j または cob2_r) の -q オプションを使用してコンパイラー・オプションを指定するとき、ADDR(32) を 32 に省略したり、ADDR(64) を 64 に省略したりできます。以下に例を示します。

```
cob2 -q64 prog64.cb1
```

ストレージ割り振り:

以下の COBOL データ型に対するストレージ割り振りは、ADDR コンパイラー・オプションの設定に依存します。

- USAGE POINTER (ADDRESS OF 特殊レジスターおよび JNIEnvPtr 特殊レジスターも同様。これらは暗黙的にこの使用法を持つ)
- USAGE PROCEDURE-POINTER
- USAGE FUNCTION-POINTER
- USAGE OBJECT REFERENCE
- USAGE INDEX

ADDR(32) が有効な場合、上記にリストされたいずれかの使用法を持つ項目がプログラムに含まれているとき、その項目ごとに 4 バイトが割り振られます。ADDR(64) が有効な場合、項目ごとに 8 バイトが割り振られます。

上記のいずれかの USAGE を持つデータ項目に対して SYNCHRONIZED 文節が指定されている場合、ADDR(32) が有効になっていると、その項目はフルワード境界に調整され、ADDR(64) が有効になっていると、その項目はダブルワード境界に調整されます。

ADDR オプションの設定は、いくつかのコンパイラー限界値に影響します。詳しくは、下記のコンパイラー限界値に関する関連参照を参照してください。

LENGTH OF 特殊レジスター:

ADDR(32) が有効な場合、LENGTH OF 特殊レジスターには次の暗黙の定義があります。

PICTURE 9(9) USAGE IS BINARY

ADDR(64) が有効な場合、LENGTH OF 特殊レジスターには次の暗黙の定義があります。

PICTURE 9(18) USAGE IS BINARY

LENGTH 組み込み関数:

ADDR(32) が有効な場合、LENGTH 組み込み関数の戻り値は 9 桁の整数です。

ADDR(64) が有効な場合、戻り値は 18 桁の整数です。

プログラミングの要件および制約事項:

- アプリケーション内のプログラム・コンポーネントはすべて、同じ ADDR オプション設定を使用してコンパイルしなければなりません。1 つのアプリケーションで 32 ビット・プログラムと 64 ビット・プログラムを混用することはできません。
- 言語間通信: 複数言語アプリケーションでは、64 ビット COBOL プログラムは 64 ビット C/C++ プログラムと結合でき、32 ビット COBOL プログラムは 32 ビット C/C++ プログラムまたは PL/I プログラムと結合できます。
- オブジェクト指向アプリケーション: オブジェクト指向構文を使用する 64 ビット COBOL アプリケーションは、64 ビット Java SDK を必要とします。オブジェクト指向構文を使用する 32 ビット COBOL アプリケーションは、32 ビット SDK を必要とします。
- CICS: CICS TXSeries 環境で実行される COBOL プログラムは 32 ビット・プログラムでなければなりません。

関連概念

261 ページの『Java SDK サポート』

関連タスク

118 ページの『データ項目の長さの検出』

235 ページの『コンパイル済みプログラム』

364 ページの『CICS のもとで実行する COBOL プログラムのコーディング』

508 ページの『COBOL および C/C++ プログラム間の呼び出し』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

265 ページの『矛盾するコンパイラー・オプション』

617 ページの『32 ビット・アプリケーションの 64 ビット・モードへの移行』

コンパイラー限界値 (COBOL for AIX 言語解説書)

ARITH

ARITH は、整数にコーディングできる最大桁数、および固定小数点の中間結果で使用される桁数に影響を与えます。

ARITH オプションの構文

▶▶ ARITH(COMPAT
EXTEND) ▶▶

デフォルト: ARITH(COMPAT)

省略形: AR(C)、AR(E)

ARITH(EXTEND) を指定すると、次のようになります。

- パック 10 進数、外部 10 進数、 および数字編集のデータ項目の PICTURE 節で指定できる桁位置の最大数が、18 から 31 へ引き上げられます。
- 固定小数点数値リテラルに指定できる桁数の最大数が、18 から 31 に上がります。以下の場所を含め、数値リテラルが現在許可されているところであればどこでも、長精度の数値リテラルを使用することができます。
 - PROCEDURE DIVISION ステートメントのオペランド
 - VALUE 節 (長精度 PICTURE を含む数値データ項目に関する)
 - 条件名の値 (長精度 PICTURE を含む数値データ項目に関する)
- NUMVAL および NUMVAL-C への引数の中で指定できる桁数の最大数が、18 から 31 に上がります。
- FACTORIAL 関数への整数引数の最大値は、29 です。
- 算術ステートメントの中間結果は、**拡張モード** を使用します。

ARITH(COMPAT) を指定すると、次のようになります。

- パック 10 進数、外部 10 進数、 および数字編集のデータ項目の PICTURE 節の桁位置の最大数は 18 です。
- 固定小数点数値リテラルに指定できる桁数の最大数は、18 です。
- NUMVAL および NUMVAL-C への引数の中で指定できる桁数の最大数は、18 です。
- FACTORIAL 関数への整数引数の最大値は、28 です。
- 算術ステートメントの中間結果は、**互換モード** を使用します。

関連概念

631 ページの『付録 D. 中間結果および算術精度』

BINARY

BINARY はコメントとして扱われます。

BINARY オプションの構文



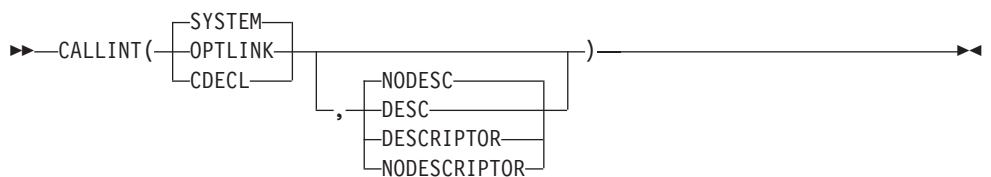
デフォルト: BINARY(NATIVE)

省略形: なし

CALLINT

CALLINT は、CALL ステートメントによって行われる呼び出しに適用できる呼び出しインターフェース規則を示し、引数記述子を生成するかどうかを示すのに使用します。

CALLINT オプションの構文



デフォルト: CALLINT(SYSTEM,NODESC)

省略形: なし

特定の CALL ステートメントでこのオプションをオーバーライドするには、コンパイラー指示 >>CALLINT を使用します。

CALLINT には 2 つのサブオプション・セットがあります。

- 呼び出しインターフェース規約の選択

SYSTEM The SYSTEM サブオプションは、プラットフォームの標準システムのリンケージ規約が呼び出し規約となることを指定します。

SYSTEM は、AIX でサポートされる唯一の呼び出しインターフェース規則です。

OPTLINK

OPTLINK サブオプションをコーディングすると、コンパイラーが I レベルの診断を生成し、ディレクティブ全体 (最初のキーワードだけでなく) が無視されます。

CDECL CDECL サブオプションをコーディングする場合、コンパイラーが I レベルの診断を生成し、ディレクティブ全体 (最初のキーワードのみではなく) が無視されます。

- 引数記述子が生成されるかどうかの指定

DESC DESC サブオプションは、CALL ステートメントの各引数に、引数記述子が渡されることを指定します。引数記述子の詳細については、下記の関連参照を参照してください。

重要: オブジェクト指向プログラムでは、DESC サブオプションを指定しないでください。

DESCRIPTOR

DESCRIPTOR サブオプションは、DESC サブオプションと同義です。

NODESC NODESC サブオプションは、CALL ステートメントのどの引数に対しても、引数記述子を渡さないことを指定します。

NODESCRIPTOR

NODESCRIPTOR サブオプションは、NODESC サブオプションと同義です。

関連参照

319 ページの『第 15 章 コンパイラー指示ステートメント』

CHAR

CHAR は、USAGE DISPLAY および USAGE DISPLAY-1 データ項目の表記および実行時の処置に影響を与えます。

CHAR オプションの構文



Diagram illustrating the syntax for CHAR options: CHAR(NATIVE | EBCDIC | S390). The options are listed in a box within the parentheses, separated by vertical bars. The diagram shows the opening and closing parentheses and a long arrow pointing to the right, indicating the continuation of the statement.

デフォルト: CHAR(NATIVE)

省略形: なし

プラットフォームのネイティブの文字表現 (ネイティブ形式) を使用するには、CHAR(NATIVE) を指定します。COBOL for AIX の場合、ネイティブ形式は、実行時に有効なロケールによって示されるコード・ページによって定義されます。コード・ページは、単一バイトの ASCII コード・ページまたは ASCII ベースのマルチバイト・コード・ページ (UTF-8、EUC、または ASCII DBCS) にすることができます。

CHAR(EBCDIC) と CHAR(S390) は同義です。これらは、DISPLAY および DISPLAY-1 データ項目が zSeries の文字表現 (つまり EBCDIC) であることを示します。

ただし、USAGE 文節の NATIVE 句で定義された DISPLAY および DISPLAY-1 データ項目は、CHAR(EBCDIC) オプションの影響を受けません。これらは常にプラットフォームのネイティブ形式で格納されます。

CHAR(EBCDIC) コンパイラ・オプションは、実行時の処理に次のような影響を及ぼします。

- **USAGE DISPLAY および USAGE DISPLAY-1 項目:** USAGE DISPLAY で記述されたデータ項目内の文字は、単一バイトの EBCDIC 形式として処理されます。USAGE DISPLAY-1 で記述されたデータ項目内の文字は、EBCDIC DBCS 形式として処理されます (後続の記述で、*EBCDIC* という用語は、USAGE DISPLAY の単一バイトの EBCDIC 形式および USAGE DISPLAY-1 の EBCDIC DBCS 形式を指しています)。
 - ネイティブ形式でエンコードされたデータは、端末からの ACCEPT で EBCDIC 形式に変換されます。
 - EBCDIC データは、端末への DISPLAY でネイティブ形式に変換されます。
 - 英数字リテラルおよび DBCS リテラルの内容は、EBCDIC でエンコードされたデータ項目への割り当てのため、EBCDIC 形式に変換されます。CHAR(EBCDIC) オプションが有効な場合の文字データの比較に関する規則については、COLLSEQ オプションに関する下記の関連参照を参照してください。
 - 編集には EBCDIC 文字を使用します。
 - 埋め込みには EBCDIC スペースを使用します。英数字操作 (割り当てや比較など) に使用されるグループ項目は、グループ内の基本項目の定義に関わらず、単一バイトの EBCDIC スペースで埋め込まれます。
 - USAGE DISPLAY 項目への割り当て、またはこの項目との関係条件において、VALUE 文節内で使用される表意定数 SPACE または SPACES は、1 バイトの EBCDIC スペース (つまり X'40') として扱われます。
 - DISPLAY-1 項目への割り当て、またはこの項目との関係条件において、VALUE 文節内で使用される表意定数 SPACE または SPACES は、1 バイトの EBCDIC DBCS スペース (つまり X'4040') として扱われます。
 - Class テストは、EBCDIC の値範囲に基づいて実行されます。
- **USAGE DISPLAY 項目**
 - CALL *identifier*, CANCEL *identifier*, または形式 6 SET ステートメント内の *program-name* はネイティブ形式に変換されます (*identifier* で示されるデータ項目が EBCDIC でエンコードされている場合)。
 - ASSIGN USING *data-name* の *data-name* で示されるデータ項目内の *file-name* は、ネイティブ形式に変換されます (データ項目が EBCDIC でエンコードされている場合)。
 - SORT-CONTROL 特殊レジスタ内の *file-name* は、ソートまたはマージ関数に渡される前にネイティブ形式に変換されます (SORT-CONTROL には、暗黙的な定義 USAGE DISPLAY があります)。
 - ゾーン 10 進数データ (USAGE DISPLAY を使用した数値 PICTURE 文節) および display 浮動小数点データは、EBCDIC 形式として処理されます。例えば、PIC S9 value "1" の値は、X'31' ではなく X'F1' です。

- **グループ項目:** 英数字グループ項目は、USAGE DISPLAY 項目と同様に扱われます。(英数字グループ項目の USAGE 文節は、グループ自体ではなくグループ内の基本項目に適用されます。)

16 進数リテラルが文字データに割り当てられるか、文字データと比較される場合は、16 進数リテラルが EBCDIC 文字を表すものと想定されます。例えば、X'C1' は、値 'A' を持つ英数字項目と等しくなります。

表意定数 HIGH-VALUE または HIGH-VALUES、LOW-VALUE または LOW-VALUES、SPACE または SPACES、ZERO または ZEROS、および QUOTE または QUOTES は論理的に、EBCDIC でエンコードされたデータ項目の割り当てまたは比較を行うための EBCDIC 文字表現として扱われます。

英数字の USAGE DISPLAY 項目間の比較では、照合シーケンスとして、2 進数 (16 進数) 値に基づく文字順序シーケンスが使用され、1 バイト文字の代替照合シーケンスが指定されている場合は、これによって変更されます。

関連タスク

210 ページの『文字データのコード・ページの指定』

関連参照

274 ページの『COLLSEQ』

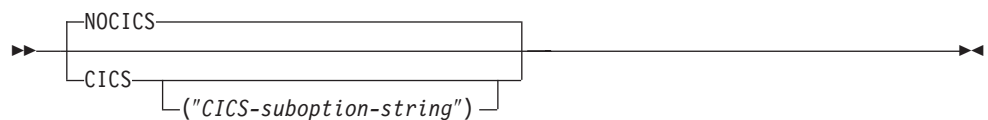
397 ページの『XML 文書のエンコード方式』

627 ページの『付録 C. zSeries ホスト・データ形式についての考慮事項』

CICS

CICS コンパイラー・オプションを指定すると、組み込みの CICS 変換プログラムが使用可能になり、CICS サブオプションを指定できるようになります。COBOL ソース・プログラムに EXEC CICS ステートメントが含まれており、プログラムが分離型の CICS 変換プログラムで処理されていないときは、CICS オプションを使用する必要があります。

CICS オプションの構文



デフォルト: NOCICS

省略形: なし

CICS オプションは、CICS プログラムをコンパイルする場合にのみ使用してください。CICS オプションを指定してコンパイルされたプログラムは、非 CICS 環境では実行することができません。

NOCICS オプションを指定した場合、ソース・プログラム内で検出された CICS ステートメントはすべて診断され、破棄されます。

引用符または単一引用符のどちらかを使用して、CICS サブオプションのストリングを区切ります。

CBL または PROCESS ステートメントで、上記の構文を使用することができます。cob2 または cob2_r コマンドで CICS オプションを使用した場合、ストリング区切り文字として使用できるのは、単一引用符 (') のみです (-q"CICS('options')")。

長い CICS サブオプション・ストリングを、複数のサブオプション・ストリングに分割して、複数の CBL または PROCESS ステートメントに置くことができます。それぞれの CICS サブオプションは、指定された順に連結されます。例えば、ソース・ファイル mypgm.cbl に以下のコードが含まれているとします。

```
cb1 . . . CICS("string2") . . .  
cb1 . . . CICS("string3") . . .
```

コマンド cob2_r mypgm.cbl -q"CICS('string1')" を発行すると、コンパイラーは、次のサブオプション・ストリングを組み込み CICS 変換プログラムに渡します。

```
"string1 string2 string3"
```

ここに示すように、連結されたストリングはシングル・スペースで区切られます。同じ CICS サブオプションのインスタンスが複数見つかった場合は、連結ストリングにおいて最後に指定されたものが有効となります。コンパイラーでは、連結されたサブオプション・ストリングの長さは 4KB に制限されます。

関連概念

369 ページの『組み込みの CICS 変換プログラム』

関連タスク

363 ページの『第 19 章 COBOL プログラムの開発 (CICS の場合)』

関連参照

265 ページの『矛盾するコンパイラー・オプション』

COLLSEQ

COLLSEQ は、英数字および DBCS オペランドを比較するための照合シーケンスを指定します。

COLLSEQ オプションの構文

```
▶▶—COLLSEQ( [BINARY  
              —LOCALE  
              —EBCDIC] ) —————▶▶
```

デフォルト: COLLSEQ(BINARY)

省略形は、CS(L)、CS(E)、CS(BIN)、CS(B) です。

COLLSEQ に次のサブオプションを指定できます。

- COLLSEQ(EBCDIC): ASCII 照合シーケンスではなく EBCDIC 照合シーケンスを使用します。
- COLLSEQ(LOCALE): (ロケールの照合に関する国/地域別情報と整合した) ロケールに依存する照合を使用します。
- COLLSEQ(BIN): 16 進値の文字を使用します。ロケール設定は影響しません。この設定を行うと、実行時のパフォーマンスが向上します。

STANDARD-1、STANDARD-2、または EBCDIC の英字名を持つソースに PROGRAM COLLATING SEQUENCE 文節を使用する場合は、英数字オペランドの比較では COLLSEQ オプションが無視されます。PROGRAM COLLATING SEQUENCE is NATIVE を指定した場合は、COLLSEQ オプションが適用されます。それ以外の場合は、PROGRAM COLLATING SEQUENCE 文節で指定された英字名がリテラルで定義されているときに、COLLSEQ オプションによって指定された照合シーケンスが使用されます。このオプションは、この英字名で指定されたユーザー定義のシーケンスによって変更されます。(詳細については、ALPHABET 文節の関連参照をご覧ください。)

PROGRAM COLLATING SEQUENCE 文節は、DBCS 比較には影響しません。

前述のサブオプション NATIVE は推奨できません。NATIVE サブオプションを指定する場合は、COLLSEQ(LOCALE) が前提となります。

次の表に、PROGRAM COLLATING SEQUENCE 文節が指定されていない場合に、比較に使用されるデータの型 (ASCII または EBCDIC) および有効な COLLSEQ オプションに基づいて、適用できる変換および照合シーケンスをまとめます。このオプションが指定されている場合は、コンパイラー・オプションの指定よりもソースの指定が優先されます。CHAR オプションによって、データが ASCII になるか EBCDIC になるかが決まります。

表 36. 被比較数のデータ型および照合シーケンスが比較に与える影響

被比較数	COLLSEQ(BIN)	COLLSEQ(LOCALE)	COLLSEQ(EBCDIC)
両方とも ASCII	変換は行われません。比較は 2 進数値 (ASCII) に基づきます。	変換は行われません。比較は現行のロケールに基づきます。	両方の被比較数が EBCDIC に変換されます。比較は 2 進数値 (EBCDIC) に基づきます。
ASCII と EBCDIC の混在	EBCDIC の被比較数が ASCII に変換されます。比較は 2 進数値 (ASCII) に基づきます。	EBCDIC の被比較数が ASCII に変換されます。比較は現行のロケールに基づきます。	ASCII の被比較数が EBCDIC に変換されます。比較は 2 進数値 (EBCDIC) に基づきます。
両方とも EBCDIC	変換は行われません。比較は 2 進数値 (EBCDIC) に基づきます。	両方の被比較数が ASCII に変換されません。比較は現行のロケールに基づきます。	変換は行われません。比較は 2 進数値 (EBCDIC) に基づきます。

関連タスク

8 ページの『照合シーケンスの指定』

216 ページの『ロケール付きの照合シーケンスの制御』

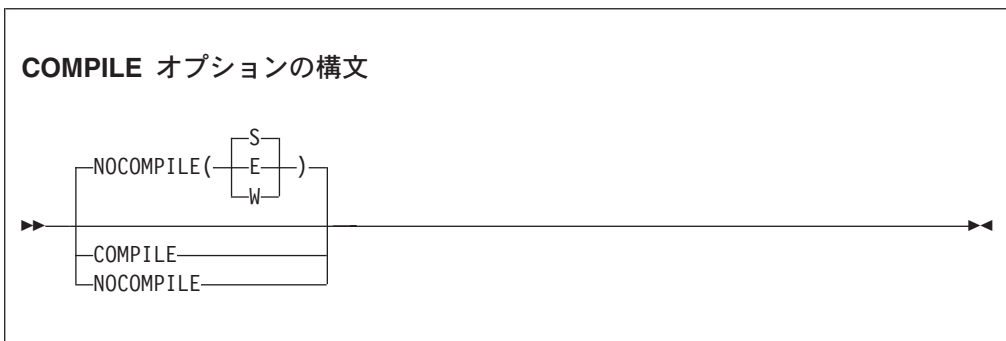
関連参照

271 ページの『CHAR』

ALPHABET 節 (COBOL for AIX 言語解説書)

COMPILE

COMPILE オプションは、重大エラーがあっても完全コンパイルを強制的に行う場合に限り、使用してください。すべての診断およびオブジェクト・コードが生成されます。コンパイルの結果として重大エラーが発生した場合は、生成されたオブジェクト・コードを実行しないでください。実行した場合の結果は保証されず、異常終了する場合があります。



デフォルト: NOCOMPILE(S)

省略形: CINOC

NOCOMPILE にサブオプションを指定しないで使用すると、構文検査を要求します (診断だけが作成され、オブジェクト・コードは生成されません)。サブオプションなしで NOCOMPILE を使用すると、オブジェクト・コードが生成されないため、いくつかのコンパイラー・オプション (LIST、OPTIMIZE、SSRANGE、および TEST) が無効になります。

NOCOMPILE にサブオプション W、E、または S を付けて使用すると、条件付き完全コンパイルを行います。コンパイラーが指定されたレベルのエラーを見つけると、完全コンパイル (診断およびオブジェクト・コード) は停止し、構文検査だけを継続します。

関連タスク

336 ページの『コーディング・エラーの検出』

関連参照

243 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

CURRENCY

CURRENCY オプションを使用すれば、COBOL プログラムで使用する代替のデフォルト通貨記号を指定することができます。(デフォルトの通貨記号はドル記号 (\$) です。)

CURRENCY オプションの構文



デフォルト: NOCURRENCY

省略形: CURRINOCURR

NOCURRENCY を指定すると、代替のデフォルト通貨記号が使用されません。

デフォルト通貨記号を変更するには、CURRENCY(*literal*) オプションを使用します。ここで、*literal* は、単一文字を表す有効な COBOL 英数字リテラル (または 16 進リテラル) です。リテラルは、次のリストのものにすることはできません。

- 数値 0 から 9
- 英大文字 A B C D E G N P R S V X Z またはその英小文字
- スペース
- 特殊文字 * + - / , . ; () " = ' `
- 形象定数
- ヌル終了リテラル
- DBCS リテラル
- 国別リテラル

プログラムが 1 つの通貨タイプしか処理しない場合には、CURRENCY SIGN 節の代わりに CURRENCY オプションを使用して、プログラムの PICTURE 節で使用する通貨記号を指定できます。プログラムで複数の通貨タイプを処理する場合は、CURRENCY SIGN 節と WITH PICTURE SYMBOL 句を併用して、異なる通貨記号タイプを指定しなければなりません。

CURRENCY オプションと CURRENCY SIGN 節の両方をプログラムで使用した場合は、CURRENCY オプションの方が無視されます。CURRENCY SIGN 節で指定した通貨記号を、PICTURE 節で使用することができます。

NOCURRENCY オプションが有効なときに、CURRENCY SIGN 節を省略すると、通貨記号の PICTURE 記号としてドル記号 (\$) が使用されます。

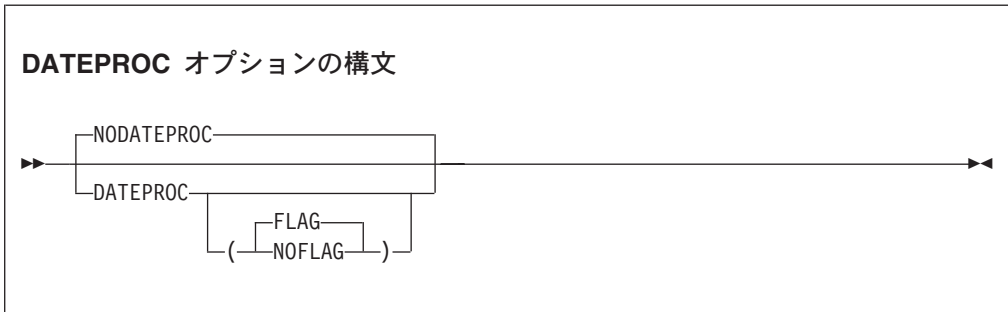
区切り文字: CURRENCY オプション・リテラルは、QUOTE|APOST コンパイラー・オプションの設定に関係なく、単一引用符または二重引用符で区切ることができます。

関連タスク

62 ページの『通貨記号の使用』

DATEPROC

DATEPROC オプションは、COBOL コンパイラーの 2000 年言語拡張を使用可能にする場合に使用します。



デフォルト: NODATEPROC、または DATEPROC(FLAG) (DATEPROC だけが指定された場合)

省略形: DPINODP

DATEPROC(FLAG)

DATEPROC(FLAG) を指定すると、2000 年言語拡張が使用可能になり、言語エレメントが拡張機能を使用するか、または言語エレメントが拡張機能の影響を受けるたびに、コンパイラーは診断メッセージを作成します。このメッセージは通常は通知レベルまたは警告レベルのメッセージであり、日付依存型処理に関連するステートメントを識別します。日付構造のエラーまたは矛盾の可能性を識別する追加のメッセージが生成されることもあります。

診断メッセージの作成とソース・リスト内またはソース・リストの後にそれらのメッセージが示されるかどうかは、FLAG コンパイラー・オプションの設定に左右されます。

DATEPROC(NOFLAG)

DATEPROC(NOFLAG) を指定すると、2000 年言語拡張は有効になりますが、COBOL ソースにエラーまたは矛盾がない限り、コンパイラーは関連メッセージを作成しません。

NODATEPROC

NODATEPROC は、このコンパイル単位に関して拡張機能を使用可能にしないことを示します。このオプションは、日付関連プログラム構造に、次のような影響を与えます。

- DATE FORMAT 節は構文検査されますが、プログラムの実行には影響しなくなります。
- DATEVAL と UNDATE 組み込み関数は無効です。すなわち、組み込み関数によって戻り値は引数の値と同じになります。
- YEARWINDOW 組み込み関数は値 0 を戻します。

関連参照

286 ページの『FLAG』

317 ページの『YEARWINDOW』

DIAGTRUNC

DIAGTRUNC を使用すると、受け取り側が数値である MOVE ステートメントの場合に、受け取りデータ項目の整数桁数が送り出しデータ項目またはリテラルよりも少ないときには、コンパイラーは、重大度 4 (警告) の診断メッセージを出します。複数の受け取り側があるステートメントでは、切り捨てられる可能性があるそれぞれの受け取り側ごとにメッセージが出されます。

DIAGTRUNC オプションの構文



デフォルト: NODIAGTRUNC

省略形: DTR、NODTR

診断メッセージは、次のようなステートメントに関連した暗黙の移動の場合にも出されます。

- INITIALIZE
- READ . . . INTO
- RELEASE . . . FROM
- RETURN . . . INTO
- REWRITE . . . FROM
- WRITE . . . FROM

送信フィールドが参照変更である場合を除いて、英数字データ名またはリテラルの送り出し側から数値の受け取り側への移動についても、診断が出されます。

TRUNC(BIN) オプションを指定した場合は、COMP-5 の受け取り側についても、2 進数の受け取り側についても診断は行われません。

関連概念

45 ページの『数値データの形式』

108 ページの『参照修飾子』

関連参照

312 ページの『TRUNC』

DYNAM

DYNAM を使用すると、CALL *literal* ステートメントにより呼び出された、ネストされていない、別々にコンパイルされたプログラムを実行時に動的にロードしたり (CALL の場合)、削除したり (CANCEL の場合) することができます。(CALL *identifier* ステートメントの場合、常にターゲット・プログラムは実行時にロードされます。このオプションの影響を受けません。)



デフォルト: NODYNAM

省略形: DYNINODYN

ON EXCEPTION 句の条件は、DYNAM オプションが有効化されている場合にのみ、CALL *literal* ステートメントで使用できます。

制約事項: 別途または組み込みの CICS 変換プログラムによって変換されるプログラムでは、DYNAM コンパイラー・オプションを使用してはなりません。

NODYNAM を使用すれば、ターゲット・プログラム名は、リンカーで解決されます。

DYNAM オプションを使用した場合、次のステートメントの動作は、

```
CALL "myprogram" . . .
```

次のステートメントと同じ動作です。

```
MOVE "myprogram" to id-1  
CALL id-1 ...
```

関連概念

507 ページの『CALL *identifier* および CALL *literal*』

関連参照

265 ページの『矛盾するコンパイラー・オプション』

ENTRYINT

このオプションは、コメントとして扱われます。

ENTRYINT オプションの構文



デフォルト: ENTRYINT(SYSTEM)

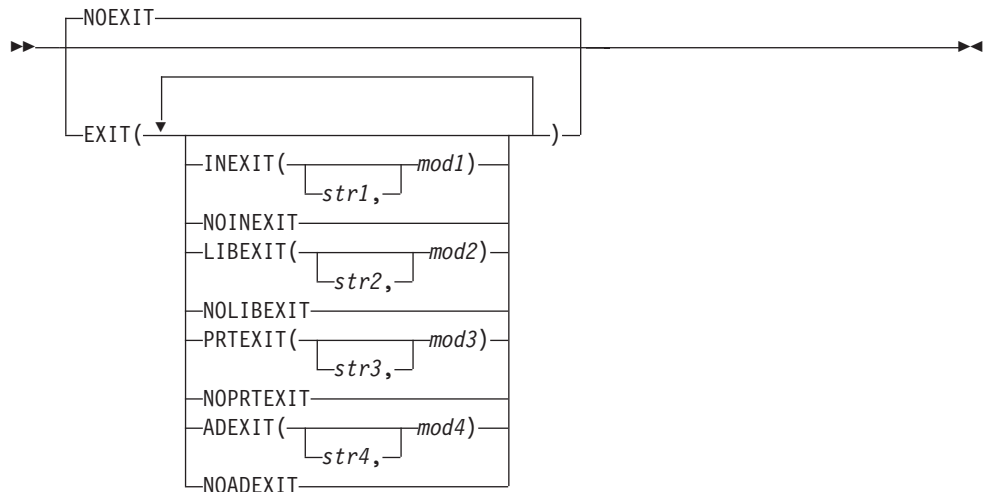
省略形: なし

EXIT

EXIT オプションは、SYSIN、SYSLIB (またはコピー・ライブラリー)、SYSPRINT、および SYSADATA の代わりにユーザー提供モジュールをコンパイラーに受け入れさせる場合に使用します。

EXIT モジュールを作成するときには、モジュールが共用ライブラリーとしてリンクされていることを確認してから、COBOL コンパイラーで実行してください。EXIT モジュールは、プラットフォームのシステム・リンケージ規約とともに呼び出されます。

EXIT オプションの構文



デフォルト: NOEXIT

省略形: EX(INXINOINX,LIBXINOLIBX,PRTXINOPRTX,ADXINOADX)

オプションの指定: PROCESS (または CBL) ステートメントで EXIT オプションを指定することはできません。指定できるのは、以下のいずれかの方法に限られます。

- cob2、cob2_j、または cob2_r コマンドでのオプションとして
- COBOPT 環境変数

サブオプションをまったく指定せずに EXIT オプションを指定する (つまり、EXIT() を指定する) と、NOEXIT が有効になります。サブオプションは、コンマまたはスペースで区切って任意の順序で指定することができます。サブオプションの肯定形式と否定形式 (INEXIT|NOINEXT、LIBEXIT|NOLIBEXIT、PRTEXTIT|NOPRTEXTIT、または ADEXIT|NOADEXIT) の両方を指定した場合は、最後に指定された形式が有効になります。同じサブオプションを複数回指定すると、最後に指定したものが有効になります。

INEXIT(['str1'],mod1)

コンパイラーは、SYSIN ではなく、ユーザー提供のロード・モジュール (*mod1* はモジュール名) からソース・コードを読み取ります。

LIBEXIT(['str2'],mod2)

コンパイラーは、*library-name* または SYSLIB ではなく、ユーザー提供のロード・モジュール (*mod2* はモジュール名) からコピーブックを入手します。COPY ステートメントまたは BASIS ステートメントと一緒に使用するためです。

PRTEXTIT(['str3'],mod3)

コンパイラーは、プリンター宛先の出力を、SYSPRINT ではなく、ユーザー提供のロード・モジュール (*mod3* はモジュール名) に渡します。

ADEXIT(['str4'],mod4)

コンパイラーは、SYSADATA 出力を、ユーザー提供のロード・モジュール (*mod4* はモジュール名) に渡します。

モジュール名 *mod1*、*mod2*、*mod3*、および *mod4* は、同じものを参照することが可能です。

サブオプション '*str1*'、'*str2*'、'*str3*'、および '*str4*' は、ロード・モジュールに渡される文字ストリングです。これらのストリングはオプションです。これらを使用する場合は、64 文字以内の長さにし、単一引用符で囲む必要があります。任意の文字を使用できますが、組み込む単一引用符は二重にしなければならず、小文字は大文字に変換されます。

文字ストリング形式

'*str1*'、'*str2*'、'*str3*'、または '*str4*' が指定された場合、そのストリングは次の形式で、適切なユーザー出口モジュールに渡されます。ここで、LL はストリングの長さを含むハーフワード (ハーフワード境界) です。次の表に、パラメーター・リスト内に使用される文字ストリングの場所を示します。

LL	ストリング
----	-------

ユーザー出口作業域

When an exit 出口を使用すると、コンパイラーはユーザー出口作業域を提供します。この作業域を使用して、出口モジュールによって割り振られたストレージのアドレスを保存することができます。これにより、モジュールは再入可能な状態になります。

ユーザー出口作業域は、フルワード境界に常駐する 4 フルワードです。この作業域が 2 進ゼロに初期設定された後、最初の出口ルーチンが呼び出されます。作業域のアドレスは、パラメーター・リストの出口モジュールに渡されます。初期化後、コンパイラーは作業域に参照を行いません。このため、コンパイル時に複数の出口がアクティブになる場合は、作業域を使用するための独自の規則を確立する必要があります。例えば、INEXIT モジュールは作業域の最初のワードを使用し、LIBEXIT モジュールは 2 番目のワードを使用し、PRTEXIT モジュールは 3 番目のワードを使用します。

出口モジュールのパラメーター・リスト

次の表に、コンパイラーが出口モジュールとやり取りするために使用するパラメーター・リストの形式を示します。

表 37. 出口モジュールのパラメーター・リスト

Offset	含まれる項目	項目の説明
00	ユーザー出口タイプ	操作を実行するユーザー出口を識別するハーフワード。 1=INEXIT; 2=LIBEXIT; 3=PRTEXIT; 4=ADEXIT
02	命令コード	操作のタイプを示すハーフワード。 0=OPEN; 1=CLOSE; 2=GET; 3=PUT; 4=FINN
04	戻りコード	出口モジュールによって設定されるフルワードで、要求された操作の状況を示します。 0=Successful; 4=End-of-data; 12=Failed
08	データ長	出口モジュールが設定するフルワードで、GET 操作によって戻されるレコードの長さを指定します。
12	データまたは 'str2'	データは、出口モジュールが設定するフルワードで、GET 操作の際にユーザー所有のバッファー内のレコードのアドレスが入ります。 'str2' は OPEN にのみ適用されます。(ハーフワード境界上の) 最初のハーフワードにストリングの長さが入り、その後にはストリングが続きます。
16	ユーザー出口作業域	ユーザー出口モジュールで使用できるように、コンパイラーが提供する 4 フルワードの作業域。
32	テキスト名	完全修飾テキスト名を含むヌル終了ストリングのアドレスが入るフルワード。FINN にのみ適用されます。

表 37. 出口モジュールのパラメーター・リスト (続き)

Offset	含まれる項目	項目の説明
36	ユーザー出口パラメーター・ストリング	4 エlement配列のアドレスが入るフルワード。各Elementは、2 バイト長のフィールドの後に、出口パラメーター・ストリングを含む 64 文字のストリングが続く構造になっています。

LIBEXIT の場合は、パラメーター・ストリング配列の 2 番目のElementだけが使用され、LIBEXIT パラメーター・ストリングの長さ、このパラメーター・ストリング自体が格納されます。

INEXIT の使用

INEXIT を指定すると、コンパイラーは初期化時に出口モジュール (*mod1*) をロードし、OPEN 命令コードを使用してモジュールを呼び出します。これにより、モジュールは、処理を行うためのソースを準備し、OPEN 要求の状況をコンパイラーに戻すことができます。その後は、コンパイラーがソース・ステートメントを要求するたびに、GET 命令コードによって出口モジュールが呼び出されます。出口モジュールは、次のステートメントのアドレスと長さ、または (ソース・ステートメントがそれ以上存在しない場合は) データ終了標識のいずれかを戻します。データ終了が存在する場合、コンパイラーは CLOSE 命令コードを使用して出口モジュールを呼び出し、モジュールがその入力に関するリソースをすべて解放できるようにします。

コンパイラーはパラメーター・リストを使用して、出口モジュールと連絡します。パラメーター・リストは、10 個のフルワードで構成されます。戻りコード、データ長、およびデータ・パラメーターは、出口モジュールによってコンパイラーに戻され、他の項目はコンパイラーから出口モジュールに渡されます。

パラメーター・リストの内容と各項目の説明については、前述の表を参照してください。

LIBEXIT の使用

LIBEXIT を指定すると、コンパイラーは初期化時に出口モジュール (*mod2*) をロードします。コンパイラーは、COPY または BASIS ステートメントが検出されるたびに、このモジュールを呼び出してコピーブックを入手します。

LIB の使用: LIBEXIT を指定する場合は、LIB コンパイラー・オプションが有効でなければなりません。

最初の呼び出しでは、OPEN 命令コードを使用してモジュールが呼び出されます。これにより、モジュールは処理用に指定された *library-name* を準備します。新規の *library-name* が初めて指定された場合にも、OPEN 命令コードが発行されます。出口モジュールは、OPEN 要求の状況を戻りコードによってコンパイラーに渡します。

OPEN 命令コードによって呼び出された出口が戻されると、FIND 命令コードによって出口モジュールが呼び出されます。出口モジュールは、指定された *library-name* 内で要求された *text-name* (または *basis-name*) に位置を設定します。この場所が

「アクティブ・コピーブック」になります。位置決めが完了すると、終了モジュールは該当する戻りコードをコンパイラーに渡します。

コンパイラーが GET 命令コードを使用して出口モジュールを呼び出すと、出口モジュールは、アクティブ・コピーブックからコピーされるレコードの長さとアドレスをコンパイラーに渡します。GET 操作は、データ終了標識がコンパイラーに渡されるまで繰り返されます。

データ終了が存在する場合、コンパイラーは CLOSE 要求を発行して、出口モジュールがその入力に関係するリソースをすべて解放できるようにします。

ネスト済み COPY ステートメント: アクティブ・コピーブックからのレコードに、COPY ステートメントを含めることができます。(ただし、ネストされた COPY ステートメントに REPLACING 句を含めたり、REPLACING 句を持つ COPY ステートメントに、ネストされた COPY ステートメントを含めたりすることはできません。) 有効な、ネストされた COPY ステートメントが検出されると、コンパイラーは要求を発行します。

- ネストされた COPY ステートメントから要求された library-name が以前にオープンされていない場合、コンパイラーは OPEN 命令コードを使用してこの出口モジュールを呼び出し、その後、新しい text-name に対しては FIND を使用します。
- 要求された library-name がすでにオープンしている場合、コンパイラーは新しく要求された text-name に対して FIND 命令コードを発行します (ここでは OPEN は発行されません)。

コンパイラーは、text-name への再帰呼び出しを許可しません。つまり、コピーブックは、そのコピーブックのデータの終わりに達するまでの間、ネストされた一連の COPY ステートメントの中で一度しか指定できません。

出口モジュールは、OPEN または FIND 要求を受け取ると、アクティブ・コピーブックに関する制御情報をスタックにプッシュしてから、要求された操作 (OPEN または FIND) を完了します。今度は、新しく要求された text-name (または basis-name) がアクティブ・コピーブックになります。

データ終了標識がコンパイラーに渡されるまで、一連の GET 要求を使用した通常の方法で処理が続けられます。

ネストされたアクティブ・コピーブックのデータ終了が検出されると、出口モジュールはその制御情報をスタックからポップします。コンパイラーからの次の要求は FIND であるため、出口モジュールは前のアクティブ・コピーブックに位置決めを再設定することができます。

コンパイラーが GET 要求を使用して出口モジュールを呼び出したら、出口モジュールはこのコピーブックから前に渡された同一レコードを渡す必要があります。同一レコードが渡されたことをコンパイラーが検証すると、データ終了標識が渡されるまで、GET 要求を使用して処理が続けられます。

LIBEXIT に使用されるパラメーター・リストの内容と各項目の説明については、前述の表を参照してください。

PRTEXTIT の使用

PRTEXTIT を指定すると、コンパイラーは初期化時に出口モジュール (*mod3*) をロードします。SYSPRINT データ・セットの代わりに出口モジュールを使用します。

コンパイラーは、OPEN 命令コードを使用してこのモジュールを呼び出します。これにより、モジュールは、処理を行うための出力先を準備し、OPEN 要求の状況をコンパイラーに戻すことができます。その後は、コンパイラーが行を印刷しようとするたびに、PUT 命令コードによって出口モジュールが呼び出されます。コンパイラーが印刷対象レコードのアドレスと長さを渡すと、出口モジュールは戻りコードによって PUT 要求の状況をコンパイラーに渡します。印刷されるレコードの最初のバイトには、ANSI プリンター制御文字が入ります。

コンパイルが終了する前に、コンパイラーは CLOSE 命令コードを使用して出口モジュールを呼び出し、モジュールがその出力先に関するリソースをすべて解放できるようにします。

PRTEXTIT に使用されるパラメーター・リストの内容と各項目の説明については、前述の表を参照してください。

ADEXIT の使用

ADEXIT を指定すると、コンパイラーは初期化時に出口モジュール (*mod4*) をロードします。この出口モジュールは、SYSADATA データ・セットに書き込まれるレコードごとに、そのレコードがファイルに書き込まれた直後に呼び出されます。

コンパイラーは、OPEN 命令コードを使用してこのモジュールを呼び出します。これにより、モジュールは、処理を行うための準備をし、OPEN 要求の状況をコンパイラーに戻すことができます。その後は、コンパイラーが SYSADATA レコードを書き込むたびに、PUT 命令コードによって出口モジュールが呼び出されます。コンパイラーが SYSADATA レコードのアドレスと長さを渡すと、出口モジュールは戻りコードによって PUT 要求の状況をコンパイラーに渡します。

コンパイルが終了する前に、コンパイラーは CLOSE 命令コードを使用して出口モジュールを呼び出し、モジュールがリソースをすべて解放できるようにします。

ADEXIT に使用されるパラメーター・リストの内容と各項目の説明については、前述の表を参照してください。

FLAG

重大度レベル *x* 以上のエラーのソース・リストの終わりに診断メッセージを作成するには、FLAG(*x*) を使用します。

FLAG オプションの構文



デフォルト: FLAG(I,I)

省略形: FINOF

x および y は、I、W、E、S、U のいずれかになります。

FLAG(x,y) を使用すると、重大度レベル x 以上のエラーに関して診断メッセージをソース・リストの終わりに作成し、重大度レベル y 以上のエラーに関してはエラー・メッセージをソース・リストに直接組み込むことができます。 y にコーディングされる重大度は、 x にコーディングされる重大度より低くってははいけません。FLAG(x,y) を使用するには、SOURCE コンパイラー・オプションも指定する必要があります。

ソース・リスト内のエラー・メッセージは、メッセージ・コードを指す矢印の中にステートメント番号を埋め込むことによって、強調されます。メッセージ・コードの後にメッセージ・テキストが続きます。以下に例を示します。

```
000413      MOVE CORR WS-DATE TO HEADER-DATE
==000413==>  IGYP2121-S      " WS-DATE " was not defined as a data-name. . . .
```

FLAG(x,y) が有効である場合は、重大度 y 以上のメッセージが、リスト内でそのメッセージの原因となった行の後に組み込まれます。(例外のメッセージについては、以下に示す関連参照資料を参照してください。)

エラーのフラグ付けを抑止する場合は、NOFLAG を使用してください。NOFLAG を使用しても、コンパイラー・オプションのエラー・メッセージは抑止されません。

組み込みメッセージ

- レベル U メッセージを組み込みに指定するのはお勧めできません。レベル U のメッセージの組み込みの指定は受け入れられますが、ソース内には何のメッセージも作成されません。
- FLAG オプションは、コンパイラー・オプションの処理前に作成された診断メッセージには影響しません。
- コンパイラー・オプション、CBL ステートメントまたは PROCESS ステートメント、または BASIS、COPY、および REPLACE の各ステートメントの処理中に生成された診断メッセージがソース・リストに組み込まれることはありません。このようなメッセージはすべて、コンパイラー出力の先頭に表示されます。
- *CONTROL または *CBL ステートメントの処理中に作成されたメッセージは、ソース・リストに組み込まれません。

関連参照

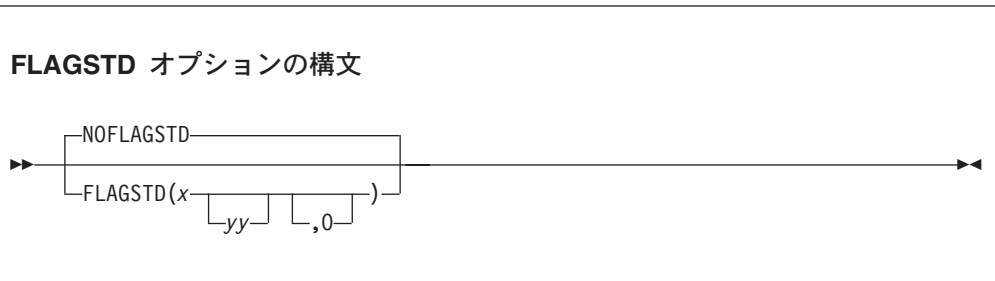
243 ページの『コンパイラ検出エラーに関するメッセージおよびリスト』

FLAGSTD

FLAGSTD を使用して、準拠していると見なされ、プログラムに組み込まれた 標準 COBOL 85 エLEMENTに関する通知メッセージを取得するように、標準 COBOL 85 のレベルまたはサブセットを指定します。

フラグ付け処理には、次の項目のどれかを指定することができます。

- 連邦情報処理標準 (FIPS) COBOL の選択されたサブセット
- オプション・モジュールのいずれか
- 廃止された言語エレメント
- サブセットとオプション・モジュールの任意の組み合わせ
- サブセットと古くなったエレメントの任意の組み合わせ
- IBM 拡張 (IBM 拡張にフラグが付けられるのは、FLAGSTD が指定され、かつ、「非規格準拠外」として識別された場合です。)



デフォルト: NOFLAGSTD

省略形: なし

x は、準拠していると見なされるよう、標準 COBOL 85 のサブセットを指定します。

- M** 最小サブセットからのものではない言語エレメントに、「規格準拠外」というフラグを付けます。
- I** 最小サブセットまたは中間サブセットからのものではない言語エレメントに、「規格準拠外」というフラグを付けます。
- H** 高位サブセットが使用されており、言語エレメントにはサブセットによってフラグが付けられません。IBM 拡張であるエレメントは、「規格準拠外、IBM 拡張」とフラグが付けられます。

yy は、単一文字または 2 文字の組み合わせによって、サブセットに組み込むオプション・モジュールを指定します。

- D** デバッグ・モジュール・レベル 1 のエレメントには、「規格準拠外」というフラグを付けません。

- N 分割モジュール・レベル 1 のエレメントには、「規格準拠外」というフラグを付けません。
- S 分割モジュール・レベル 2 のエレメントには、「規格準拠外」というフラグを付けません。

S を指定すると、N が含まれます (N は S のサブセットです)。

0 (英字) は、廃止された言語エレメントに「廃止」のフラグを付けるように指定します。

通知メッセージはソース・プログラム・リストに表示され、以下の情報を示しています。

- エレメントの「廃止」、「規格準拠外」、または「非規格準拠外」(廃止になり、しかも規格準拠外の言語エレメントには廃止のフラグだけを立てます)。
- そのエレメントが含まれている節、ステートメント、またはヘッダー。
- そのエレメントが含まれる節、ステートメント、またはヘッダーのソース・プログラム行および開始位置。
- そのエレメントが属するサブセットまたはオプション・モジュール。

FLAGSTD には、予約語の標準セットが必要です。

次の例では、関連メッセージ・コードおよびテキストとともに、フラグ付き文節、ステートメント、またはヘッダーが使用された行番号と桁が示されています。最下部に表示されている要約には、フラグ付き項目の合計と、その項目のタイプが示されています。

LINE	COL	CODE	FIPS MESSAGE TEXT				
		IGYDS8211	Comment lines before "IDENTIFICATION DIVISION": nonconforming nonstandard, IBM extension to ANS/ISO 1985.				
11.14		IGYDS8111	"GLOBAL clause": nonconforming standard, ANS/ISO 1985 high subset.				
59.12		IGYPS8169	"USE FOR DEBUGGING statement": obsolete element in ANS/ISO 1985.				
FIPS MESSAGES TOTAL				STANDARD	NONSTANDARD	OBSOLETE	
				1	1	1	

FLOAT

プラットフォームのネイティブの浮動小数点表現形式を使用するには、FLOAT(NATIVE) を指定します。COBOL for AIX の場合、ネイティブ形式は IEEE 形式になります。

FLOAT オプションの構文



デフォルト: FLOAT(NATIVE)

省略形: なし

FLOAT(HEX) と FLOAT(S390) は同義です。これらは、COMP-1 および COMP-2 データ項目が一貫して zSeries (つまり 16 進数の浮動小数点形式) で表現されることを示します。

- 16 進数の浮動小数点値は、算術演算 (計算または比較) が行われる前に IEEE 形式に変換されます。
- IEEE 浮動小数点値は、浮動小数点データ・フィールドに格納される前に、16 進形式に変換されます。
- 浮動小数点項目への割り当ては、ソースの浮動小数点データ (外部の浮動小数点など) を必要に応じて 16 進数の浮動小数点値に変換することによって行われます。

ただし、USAGE 文節の NATIVE キーワードで定義された COMP-1 および COMP-2 データ項目は、FLOAT(S390) オプションの影響を受けません。これらは常にプラットフォームのネイティブ形式で格納されます。

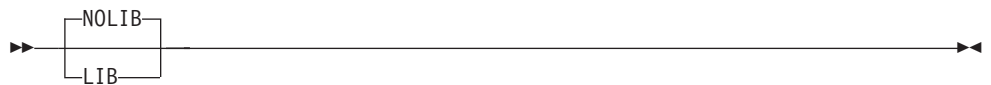
関連参照

627 ページの『付録 C. zSeries ホスト・データ形式についての考慮事項』

LIB

プログラムで COPY、BASIS、または REPLACE ステートメントを使用する場合は、LIB コンパイラー・オプションを有効にする必要があります。

LIB オプションの構文



デフォルト: NOLIB

省略形: なし

COPY ステートメントと BASIS ステートメントの場合は、さらに、コピーされたコードをコンパイラーが獲得するライブラリー（複数も可）を定義する必要があります。

- ライブラリー名を（リテラルではなく）ユーザー定義のワードで指定する場合、対応する環境変数を、コピーブックの必要なディレクトリーおよびパスにポイントするように設定する必要があります。
- COPY ステートメントで library-name を省略した場合は、cob2、cob2_j、または cob2_r コマンドの -Ixxx オプションを使用して、検索対象のパスを指定することができます。
- ライブラリー名がリテラルで指定された場合には、そのリテラル値は、実際のパス名として扱われます。

関連参照

319 ページの『第 15 章 コンパイラー指示ステートメント』

265 ページの『矛盾するコンパイラー・オプション』

LINECOUNT

LINECOUNT(*nnn*) は、コンパイル・リストの各ページに印刷する行数を指定する場合に使用します。ページ編集を抑止する場合には、LINECOUNT(0) を使用してください。

LINECOUNT オプションの構文

▶—LINECOUNT(*nnn*)—◀

デフォルト: LINECOUNT(60)

省略形: LC

nnn は、10 から 255 の整数か、0 でなければなりません。

LINECOUNT(0) を指定すると、コンパイル・リストではページ替えが行われません。

コンパイラーは、タイトル用に *nnn* のうちの 3 行を使用します。例えば、LINECOUNT(60) を指定すると、57 行のソース・コードが出力リストの各ページに印刷されます。

LIST

LIST コンパイラー・オプションは、ソース・コードのアセンブラー言語拡張のリストを作成する場合に使用します。

LIST オプションの構文



デフォルト: NOLIST

省略形: なし

PROCEDURE DIVISION でコーディングした *CONTROL (または *CBL) LIST、または NOLIST ステートメントはすべて無効です。コメントとして扱われます。

アセンブラー・リストは、名前がソース・プログラムと同じで、接尾部が .wlist のファイルに書き込まれます。

関連タスク

342 ページの『リストの入手』

関連参照

*CONTROL (*CBL) ステートメント (*COBOL for AIX 言語解説書*)

LSTFILE

生成されたコンパイラー・リストを、有効なロケールで指定されたコード・ページでエンコードするには、LSTFILE(LOCALE) を指定します。生成されたコンパイラー・リストを UTF-8 でエンコードするには、LSTFILE(UTF-8) を指定します。

LSTFILE オプションの構文



デフォルト: LSTFILE(LOCALE)

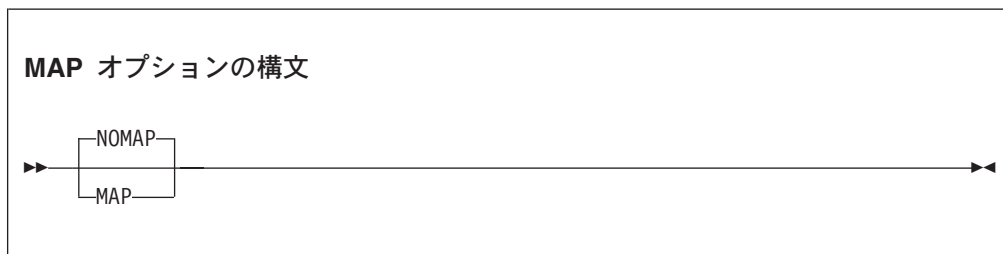
省略形: LST

関連参照

209 ページの『第 11 章 ロケールの設定』

MAP

MAP を使用すると、DATA DIVISION に定義した項目のリストを作成することができます。



デフォルト: NOMAP

省略形: なし

出力には、以下の項目が含まれます。

- DATA DIVISION のマップ
- グローバル・テーブル
- リテラル・プール
- ネストされたプログラム構造マップ、およびプログラム属性
- プログラムの WORKING-STORAGE および LOCAL-STORAGE のサイズ

MAP 出力を制限したい場合は、DATA DIVISION で *CONTROL MAP または NOMAP ステートメントを使用してください。*CONTROL NOMAP の後のソース・ステートメントは、*CONTROL MAP ステートメントによって出力が通常の MAP 形式に戻されない限り、リストには含まれません。以下に例を示します。

```
*CONTROL NOMAP          *CBL NOMAP
  01 A                    01 A
  02 B                    02 B
*CONTROL MAP             *CBL MAP
```

MAP オプションを選択すると、組み込み MAP 報告書もソース・コード・リストに印刷することができます。圧縮 MAP 情報は、DATA DIVISION の FILE SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION のデータ名定義の右側に印刷されます。XREF データと組み込み MAP 要約の両方が同じ行にあるときは、組み込み要約の方が先に印刷されます。

346 ページの『例: MAP 出力』

関連概念

331 ページの『第 17 章 デバッグ』

関連タスク

342 ページの『リストの入手』

関連参照

*CONTROL (*CBL) ステートメント (COBOL for AIX 言語解説書)

MAXMEM

MAXMEM を OPTIMIZE と一緒に使用して、コンパイラーが特定のメモリー集中の最適化のローカル・テーブルのために使用するメモリーの量を、*size* KB に制限します。特定の最適化に対して該当のメモリーが不十分な場合は、最適化の対象範囲が減らされます。

MAXMEM オプションの構文

▶—MAXMEM(*size*)————▶

デフォルト: MAXMEM(2048)

省略形: なし

値 -1 は、最適化ごとに、限界値のチェックなしで必要なだけのメモリーをとることを許可します。コンパイルされるソース・ファイル、ソース内のサブプログラムのサイズ、マシン構成、およびシステムでのワークロードによっては、この量が、使用可能なシステム・リソースを超える場合があります。

使用上の注意

- MAXMEM によって設定される限界値は、コンパイラー全体に対してではなく、特定の最適化に対するメモリー量です。コンパイル処理の全体を通して必要なテーブルは、この限界値には影響しないか、組み込まれません。
- 限界値を大きく設定しても、コンパイラーが必要とするメモリーが少ないソース・ファイルのコンパイルにはマイナスの影響はありません。
- 最適化の対象範囲を制限しても、必ずしも結果のプログラムが低速になるわけではありません。単に、パフォーマンスを向上させるためのすべての方法を見つける前に、コンパイラーが終了することがあるだけです。
- 限界値を大きくしても、必ずしも結果のプログラムが高速になるわけではありません。単に、パフォーマンスを向上させるための方法が存在する場合に、コンパイラーがそれを見つけやすくなるだけです。

これは、コンパイルされるソース・ファイル、ソース内のサブプログラムのサイズ、マシン構成、およびシステムでのワークロードによっては、設定する限界値が高すぎるとページ・スペースが使い尽くされることがあります。特に、MAXMEM(-1)を指定すると、コンパイラーが無制限の記憶量を使用しようとすることができ、最悪の場合マシンのリソースが使い尽くされることがあります。

関連参照

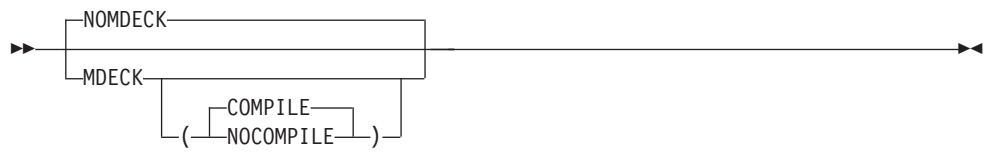
298 ページの『OPTIMIZE』

MDECK

MDECK コンパイラー・オプションは、ライブラリー処理 (すなわち、COPY、BASIS、REPLACE、EXEC CICS、および EXEC SQL INCLUDE ステートメントの拡張) の出力がファイルに書き込まれることを指定します。

MDECK 出力は、COBOL ソース・ファイルと同じ名前および接尾部 .dek を持つ、現行ディレクトリー内のファイルに書き込まれます。

MDECK オプションの構文



デフォルト: NOMDECK

省略形: NOMD、MD、MD(C)、MD(NOC)

オプション指定:

PROCESS (または CBL) ステートメントで MDECK オプションを指定することはできません。指定できるのは、以下のいずれかの方法に限られます。

- cob2、cob2_j、または cob2_r コマンドでのオプションとして
- COBOPT 環境変数

サブオプション:

- MDECK(COMPILE) が有効である場合、ライブラリー処理および MDECK 出力ファイルの生成が完了した後、正常にコンパイルが継続されますが、その際、コンパイルは COMPILEINOCOMPILER オプションの設定値に従って行われます。
- MDECK(NOCOMPILER) が有効である場合、ライブラリー処理が完了し、拡張ソース・プログラム・ファイルが書き込まれた後、コンパイルは終了します。コンパイラーは、COMPILE オプションの設定値に関係なく、構文検査やコード生成をこれ以上行いません。

サブオプションなしの MDECKMDECK を指定した場合、MDECK(COMPILE)MDECK (COMPILE) が暗黙指定されます。

MDECK 出力ファイルの内容:

MDECK オプションを CICS コンパイラー・オプション (組み込みの CICS 変換プログラム) または SQL コンパイラー・オプション (DB2 coprocessor) と一緒に使用する場合、一般に、COBOL ソース・プログラム内の EXEC CICS および EXEC SQL ステートメントが、MDECK 出力にそのまま組み込まれます。EXEC SQL INCLUDE ステートメントは、COPY ステートメントと同様に MDECK 出力内で拡張されます。

CBL、PROCESS、*CONTROL、および *CBL カード・イメージは、MDECK 出力ファイルの適切な位置に渡されます。

バッチ・コンパイル (単一入力ファイル内に複数の COBOL ソース・プログラムが含まれている) の場合、完全な拡張ソースを含んでいる単一 MDECK 出力ファイルが作成されます。

SEQUENCE コンパイラー・オプション処理はすべて MDECK ファイル内に反映されます。

COPY ステートメントは、コメントとして MDECK ファイルに組み込まれます。

関連参照

265 ページの『矛盾するコンパイラー・オプション』

319 ページの『第 15 章 コンパイラー指示ステートメント』

NCOLLSEQ

NCOLLSEQ オプションの構文

```
NCOLLSEQ( [ BINARY | LOCALE ] )
```

NCOLLSEQ は、クラス国別オペランドを比較するための照合シーケンスを指定します。

デフォルト: NCOLLSEQ(BINARY)

省略形: NCS(L)、NCS(BIN)、NCS(B)

16 進値の文字ペアを使用するには、NCOLLSEQ(BIN) を指定します。

有効なロケール値と関連付けられた照合順序のアルゴリズムを使用するには、NCOLLSEQ(LOCALE) を使用します。

関連タスク

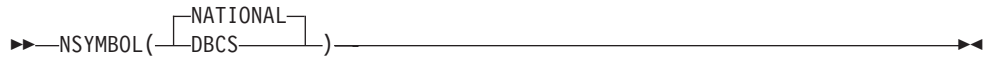
203 ページの『2 つのクラス国別オペランドの比較』

216 ページの『ロケール付きの照合シーケンスの制御』

NSYMBOL

NSYMBOL オプションは、リテラルおよび PICTURE 節で使用される N 記号の解釈を制御し、国別処理や DBCS 処理が必要かどうかを指示します。

NSYMBOL オプションの構文



デフォルト: NSYMBOL(NATIONAL)

省略形: NS(NAT|DBCS)

NSYMBOL(NATIONAL) を指定した場合:

- USAGE 節のない、記号 N のみからなる PICTURE 節で定義されたデータ項目は、USAGE NATIONAL 節が指定されている場合のように扱われます。
- N". . ." または N' . . .' の形式のリテラルは、国別リテラルとして扱われます。

NSYMBOL(DBCS) を指定した場合:

- USAGE 節のない、記号 N のみからなる PICTURE 節で定義されたデータ項目は、USAGE DISPLAY-1 節が指定されている場合のように扱われます。
- N". . ." または N' . . .' の形式のリテラルは、DBCS リテラルとして扱われます。

NSYMBOL(DBCS) オプションは、前のリリースの IBM COBOL との互換性を提供します。NSYMBOL(NATIONAL) オプションにより、前述の言語エレメントの処理がこの点に関して 標準 COBOL 2002 に準拠するようになります。

NSYMBOL(NATIONAL) は、Unicode データや Java とのインターオペラビリティのためのオブジェクト指向構文を使用するアプリケーションの場合の推奨オプションです。

NUMBER

NUMBER コンパイラー・オプションは、ソース・コードの中に行番号があり、それらの番号がエラー・メッセージと SOURCE、MAP、LIST、および XREF のリストで必要な場合に使用してください。

NUMBER オプションの構文



デフォルト: NONUMBER

省略形: NUMINONUM

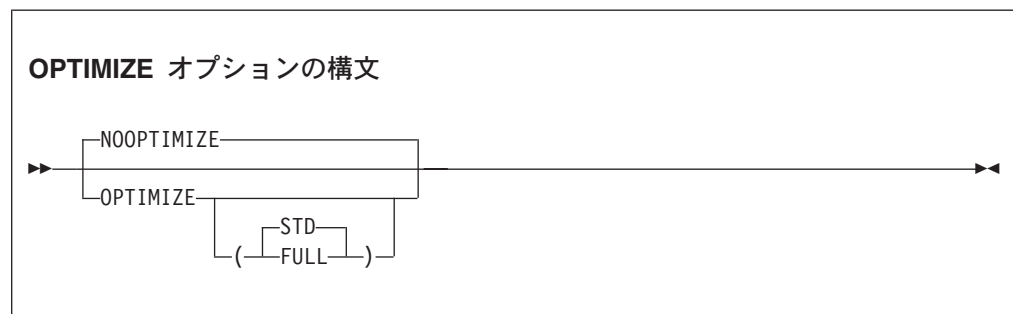
NUMBER を要求すると、コンパイラーは、桁 1 から 6 に数字だけが含まれているかどうか、および番号が数字の照合シーケンスになっているかどうかを検査します。(これに反して、SEQUENCE を使用すると、これらの桁の文字が EBCDIC 照合シーケンスになっているかどうかを検査されます。)行番号が順序どおりになっていないことがわかると、コンパイラーは先行のステートメントの行番号より 1 だけ大きい値の行番号を割り当てます。コンパイラーは、新規の値に 2 つのアスタリスクでフラグを立て、シーケンス・エラーを示すメッセージをリストに組み込みます。シーケンス検査は、先行の行の新しく割り当てられた値に基づいて、次のステートメントから継続されます。

COPY ステートメントを使用する場合、NUMBER が有効なときは、ソース・プログラムの行番号とコピーブックの行番号が対応している必要があります。

ソース・コードの中に行番号がない場合や、コンパイラーにソース・コードの行番号を無視させる場合には、NONNUMBER を使用してください。NONNUMBER が有効であると、コンパイラーは、ソース・ステートメントの行番号を生成し、それらの番号をリストで参照として使用します。

OPTIMIZE

OPTIMIZE は、オブジェクト・プログラムの実行時間を短縮するために使用します。最適化によって、オブジェクト・プログラムが使用するストレージの量を減らすこともできます。実行される最適化には、定数の伝搬、命令スケジューリング、および結果が使用されない計算の除去があります。



デフォルト: NOOPTIMIZE

省略形: OPTINOOPT

サブオプションを付けずに OPTIMIZE を指定すると、OPTIMIZE(STD) が有効になります。

FULL サブオプションは、OPT(STD) で実行される最適化に加えて、コンパイラーが DATA DIVISION から未参照のデータ項目を廃棄し、さらにこれらのデータ項目をそれぞれの VALUE 節の値に初期化するコードの生成を抑制するように要求します。OPT(FULL) が有効であると、未参照のレベル 77 項目および基本レベル 01 項目がすべて破棄されます。さらに、どの従属項目も参照されなければ、レベル 01 グル

ープ項目も破棄されます。削除された項目はリストの中で示されます。MAP オプションが有効であれば、データ・マップ情報内の XXXXX の BL 番号は、そのデータ項目が破棄されたことを示します。

推奨: データベース・アプリケーションには、OPTIMIZE(FULL) を使用してください。これによって、関連する COPY ステートメントに含まれる未使用の定数が除去されるため、大幅にパフォーマンスが向上する可能性があります。ただし、データベース・アプリケーションが未使用のデータ項目に依存している場合には、以下の推奨を参照してください。

未使用データ項目: プログラムで未使用データ項目を意図的に利用している場合は、OPT(FULL) を使用しないでください。従来は、次のような 2 つの方法が一般に使用されていました。

- 参照されたテーブルの後に未参照のテーブルを置き、2 番目のテーブルにアクセスするために最初のテーブルの範囲外添え字を使用する (以前の OS/VS COBOL プログラムで時折使用されていた手法)。ご使用のプログラムがこの手法を使用するのかどうかを判別するには、SSRANGE コンパイラー・オプションと CHECK(ON) ランタイム・オプションを一緒に使用します。この問題に対処するには、新しい COBOL の大きなテーブルのコーディング機能を使用して、テーブルを 1 つだけ使用します。
- 目印となるデータ項目を WORKING-STORAGE SECTION に置いて、プログラム・データの始めと終わりを識別できるようにする、あるいはそのデータを使用するライブラリー・ツール用のプログラムのコピーに印を付けてプログラムのバージョンを識別できるようにする方法。この問題を解決するには、これらの項目を VALUE 節ではなく、PROCEDURE DIVISION ステートメントで初期化します。この方法を使用すると、コンパイラーはこれらの項目が使用されているものと見なし、削除しません。

重大レベル以上のエラーが起こった場合、OPTIMIZE オプションはオフにされます。

関連概念

596 ページの『最適化』

関連参照

265 ページの『矛盾するコンパイラー・オプション』

294 ページの『MAXMEM』

PGMNAME

PGMNAME オプションは、プログラム名および入り口点名の処理を制御します。

PGMNAME オプションの構文

▶▶ PGMNAME (UPPER MIXED) ▶▶

デフォルト: PGMNAME(UPPER)

省略形: PGMN(LUILM)

OS/390® & VM との互換性を保つため、LONGMIXED および LONGUPPER もサポートされています。

LONGUPPER は、UPPER、LU、または U と省略することができ、LONGMIXED は、MIXED、LM、または M と省略することができます。

COMPAT: PGMNAME(COMPAT) を指定すると、PGMNAME(UPPER) が設定され、警告メッセージが戻されます。

PGMNAME オプションは、以下のコンテキストで使用される名前の処理を制御します。

- PROGRAM-ID 段落で定義されたプログラム名
- ENTRY ステートメントのプログラム入り口点名
- 以下におけるプログラム名参照:
 - CALL ステートメント
 - CANCEL ステートメント
 - SET *procedure-pointer* TO ENTRY ステートメント
 - SET *function-pointer* TO ENTRY ステートメント

PGMNAME(UPPER)

PGMNAME(UPPER) を使用する場合、PROGRAM-ID 段落で COBOL ユーザー定義語として指定されるプログラム名は、次のようなユーザー定義語に関する通常の COBOL 規則に従っていなければなりません。

- プログラム名の長さは最大 30 文字です。
- 名前で使用される文字はすべて英字、数字、またはハイフンでなければならない。
- 少なくとも 1 文字は英字にする必要があります。
- ハイフンを先頭文字や末尾文字として使用することはできません。

定義または参照のいずれかで、プログラムをリテラルとして指定する場合は、次のようになります。

- プログラム名の長さは最高 160 文字まで。
- 名前で使用される文字はすべて英字、数字、またはハイフンでなければならない。
- 少なくとも 1 文字は英字にする必要があります。
- ハイフンを先頭文字や末尾文字として使用することはできません。

外部プログラム名は、大文字に変換された英字で処理されます。

PGMNAME(MIXED)

PGMNAME(MIXED) を使用する場合、プログラム名は、切り捨てられたり、変換されたり、大文字への変換をされることなく、現状のまま処理されます。

PGMNAME(MIXED) を使用する場合、すべてのプログラム名定義は、プログラム名のリテラル形式を使用して、PROGRAM-ID 段落または ENTRY ステートメントで指定する必要があります。

PROFILE

PROFILE は、AIX プロファイル (prof ユーティリティを使用したパフォーマンス分析) 用のオブジェクト・ファイルを準備するようコンパイラーに指示します。このコンパイラー・オプションは、cob2 コマンドまたはそのバリエーションのいずれかの -p オプションとともに使用します。

PROFILE オプションの構文



デフォルト: NOPROFILE

省略形: なし

関連タスク

236 ページの『コマンド行からのコンパイル』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

QUOTE/APOST

QUOTE は、形象定数 [ALL] QUOTE または [ALL] QUOTES が 1 つ以上の引用符 (") 文字を表すようにする場合に使用します。APOST は、形象定数 [ALL] QUOTE または [ALL] QUOTES が 1 つ以上の引用符 (') 文字を表すようにする場合に使用します。

QUOTE/APOST オプションの構文



デフォルト: QUOTE

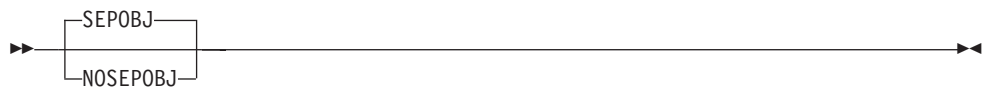
省略形: QIAPOST

区切り文字: APOST または QUOTE オプションが有効であるかどうかに関係なく、単一引用符または二重引用符のいずれかをリテラル区切り文字として使用できます。リテラルの開始の区切り文字として使用する区切り文字は、そのリテラルの終了の区切り文字としても使用しなければなりません。

SEPOBJ

SEPOBJ は、バッチ・コンパイル内で最外部にある各 COBOL プログラムを、単一のオブジェクト・ファイルとしてではなく個別のオブジェクト・ファイルとして生成するかどうかを指定します。

SEPOBJ オプションの構文



デフォルト: SEPOBJ

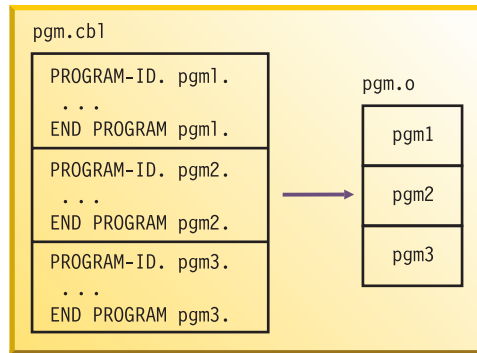
省略形: なし

バッチ・コンパイル

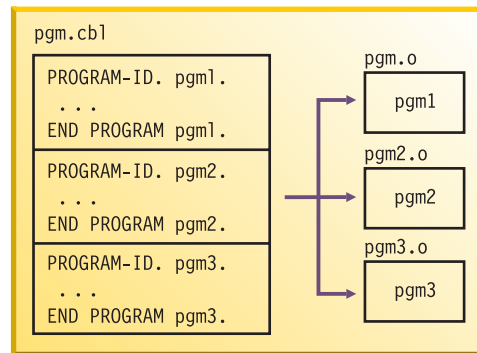
コンパイラがバッチを 1 回呼び出すことで、複数の最外部プログラム (ネストされていないプログラム) がコンパイルされる場合は、コンパイラ・オプション SEPOBJ によって、そのバッチ・コンパイルのオブジェクト・プログラム出力に対して生成されるファイルの数が決まります。

COBOL ソース・ファイル pgm.cbl に、pgm1、pgm2、pgm3 という名前の 3 つの COBOL 最外部プログラムが含まれているとします。次の図に、オブジェクト・プログラム出力が 1 ファイル (NOSEPOBJ の場合) または 3 ファイルの場合 (SEPOBJ) のどちらで生成されるかを示します。

NOSEPOBJ を使用したバッチ・コンパイル



SEPOBJ を使用したバッチ・コンパイル



使用上の注意

- 上記の例の pgm2 または pgm3 が、別のプログラムからの CALL *identifier* によって呼び出される場合、標準 COBOL 85に準拠するには SEPOBJ オプションを使用する必要があります。
- NOSEPOBJ が有効な場合は、ソース・ファイルの名前に接尾部 .o を付けたものが、オブジェクト・ファイルの名前になります。SEPOBJ が有効な場合は、PROGRAM-ID の名前に接尾部 .o を付けたものが、オブジェクト・ファイルの名前になります。
- PROGRAM-ID とオブジェクト・ファイルの名前が一致しない場合、CALL *identifier* で呼び出されるプログラムは、(PROGRAM-ID の名前ではなく) オブジェクト・ファイルの名前で参照されなければなりません。

オブジェクト・ファイルには、当該プラットフォームおよびファイル・システムに対応した、有効なファイル名を付ける必要があります。

SEQUENCE

SEQUENCE を使用すると、コンパイラーは桁 1 から 6 を調べ、ソース・ステートメントが ASCII 照合シーケンスに従って昇順に並んでいるかどうかを検査します。昇順になっていないステートメントがあると、コンパイラーは診断メッセージを出します。

桁 1 から 6 がブランクのソース・ステートメントはこのシーケンス検査には関与しないため、メッセージは出されません。

SOSI

SOSI オプションは、英数字リテラル、コメント、および DBCS ユーザー定義語での、値 X'1E' および X'1F' の扱いに影響を与えます。

SOSI オプションの構文



デフォルト: NOSOSI

省略形: なし

NOSOSI NOSOSI を使用すると、値 X'1E' および X'1F' を持つ文字位置がデータ文字として扱われます。

NOSOSI は 標準 COBOL 85 に準拠しています。

SOSI SOSI を使用すると、COBOL for AIX のシフトアウト (SO) およびシフトイン (SI) 制御文字によって、COBOL ソース・プログラム内の ASCII DBCS 文字ストリングが区切られます。SO 文字と SI 文字は、それぞれ X'1E' と X'1F' のエンコード値を持ちます。

SO 文字と SI 文字は、COBOL for AIX の COBOL ソース・コードには影響を与えません。ただし、リモート・ファイルが EBCDIC から ASCII に変換されるときにデータ処理を正しく行うために、これらの SO 文字と SI 文字がホスト DBCS の SO 文字と SI 文字のプレースホルダーとして機能する場合は例外です。

SOSI オプションが有効な場合は、COBOL for AIX の既存の COBOL 規則に加えて、次の規則が適用されます。

- すべての DBCS 文字ストリング (ユーザー定義語、DBCS リテラル、英数字リテラル、国別リテラル、およびコメント内) は、SO および SI 文字で区切る必要があります。
- ユーザー定義語には、DBCS 文字と SBCS 文字の両方を含めることはできません。
- DBCS ユーザー定義語の最大長は 14 DBCS 文字です。
- 2 バイトの小文字英字がユーザー定義語内で使用される場合は、それに対応する 2 バイトの大文字英字と同等になります。
- DBCS ユーザー定義語には少なくとも 1 文字を含める必要がありますが、1 バイト表現に、その文字に対応する文字があってはなりません。
- A から Z、a から z、0 から 9、およびハイフン (-) の 1 バイト文字の 2 バイト表現を、DBCS ユーザー定義語に含めることはできません。1 バイト表現でこれらの文字に適用される規則は、2 バイト表現でも適用されます。例えば、ハイフン (-) をユーザー定義語の先頭または末尾文字にすることはできません。

- SOSI コンパイラー・オプションが有効な場合は、X'1E' または X'1F' 値を含む英数字リテラルに対して次の規則が適用されます。
 - X'1E' および X'1F' を使用した文字位置は、SO および SI 文字として扱われます。
 - X'1E' および X'1F' を使用した文字位置は、リテラルの文字ストリング値に含まれます。ただしこれが、16 進数表記で表現されない DBCS や英数字リテラルの一部になっている場合は例外です。
 - SO 文字と SI 文字は、0 個または偶数個の仲介バイトを持つペアでなければなりません (各ペアは SO 文字で始まります)。
 - SO 文字と SI 文字のペアは、ネストすることができません。
 - 英数字リテラルは継続できません。英数字リテラルの最大長は、B 領域の単一ソース行で使用可能な桁によって限定されます。
 - 以下の言語エレメントでは、英数字リテラルを使用できません。
 - ALPHABET 文節
 - ASSIGN 文節
 - CLASS 文節
 - CURRENCY SIGN 文節
 - END METHOD マーカー
 - END PROGRAM マーカー
 - METHOD-ID 段落
 - PADDING CHARACTER 文節
 - PROGRAM-ID 段落
 - RERUN 文節
 - STOP ステートメント
- 引用符で区切られた N-literal 内に DBCS の引用符を埋め込むには、DBCS の引用符 1 つを表現するために DBCS の引用符を連続して 2 個使用します。N-literal が引用符で区切られている場合は、このリテラルに単一の DBCS 引用符を含めないでください。この規則は、一重引用符にも適用されます。
- SHIFT-OUT および SHIFT-IN 特殊レジスターは、SOSI オプションが有効かどうかにかかわらず、X'0E' および X'0F' を使用して定義されます。

一般に、ホストの COBOL プログラムが SO および SI 文字のエンコード値に依存する場合は、AIX ワークステーションで同じ動作をしません。

関連タスク

496 ページの『ASCII マルチバイト と EBCDIC DBCS ストリングの違いの処理』

SOURCE

SOURCE は、ソース・プログラムのリストを入手する場合に使用します。このリストには、PROCESS または COPY ステートメントによって組み込まれたすべてのステートメントが入ります。

SOURCE オプションの構文



デフォルト: SOURCE

省略形: SINOS

ソース・リストに組み込みメッセージが必要な場合は、SOURCE を必ず指定します。

コンパイラ出力リストにソース・コードを出したくない場合は、NOSOURCE を使用してください。

SOURCE 出力を制限したい場合は、PROCEDURE DIVISION で *CONTROL SOURCE または NOSOURCE ステートメントを使用してください。Source *CONTROL NOSOURCE ステートメントの後のソース・ステートメントは、後続の *CONTROL SOURCE ステートメントによって出力が通常の SOURCE 形式に戻されない限り、リストには含められません。

346 ページの『例: MAP 出力』

関連参照

*CONTROL (*CBL) ステートメント (COBOL for AIX 言語解説書)

SPACE

SPACE は、ソース・コード・リストで 1 行送り、2 行送り、または 3 行送りを選択するために使用します。

SPACE オプションの構文



デフォルト: SPACE(1)

省略形: なし

SPACE が意味を持つのは、SOURCE コンパイラ・オプションが有効な場合だけです。

関連参照

306 ページの『SOURCE』

SPILL

このオプションは、レジスターの予備区域とは別に設定されるサイズ (KB) を指定します。コンパイルするプログラムが非常に複雑であったり大きかったりする場合は、このオプションが必要になることがあります。

SPILL オプションの構文

▶—SPILL(*n*)—▶

デフォルト: SPILL(512)

省略形: なし

予備サイズ *n* は 96 から 32704 までの任意の整数です。

SQL

SQL コンパイラー・オプションを使用すると、DB2 coprocessor を使用可能にし、DB2 サブオプションを指定できるようになります。COBOL ソース・プログラムに SQL ステートメントが含まれており、プログラムが DB2 プリコンパイラーで処理されていない場合には、SQL オプションを必ず指定しなければなりません。

SQL オプションの構文

▶—
┌ NOSQL
└ SQL
└ ("DB2-suboption-string")

デフォルト: NOSQL

省略形: なし

NOSQL が有効な場合は、ソース・プログラム内で検出された SQL ステートメントは診断され、破棄されます。

DB2 サブオプションのストリングは、引用符または単一引用符を使用して区切ってください。

CBL または PROCESS ステートメントで、上記の構文を使用することができます。cob2 コマンド またはそのバリエーションである cob2_j または cob2_r で SQL オプシ

ョンを使用する場合、サブオプション・ストリング区切り文字として使用できるのは、単一引用符 (') のみです: `-q"SQL('suboptions')`。

関連タスク

- 357 ページの『第 18 章 DB2 環境用のプログラミング』
- 360 ページの『SQL オプションを使用したコンパイル』
- 361 ページの『DB2 サブオプションの分離』

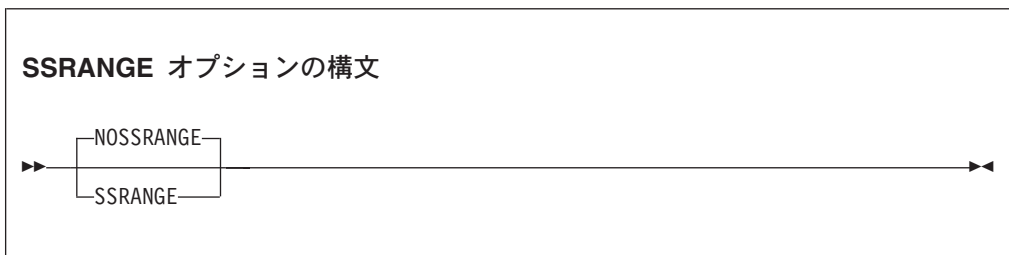
関連参照

- 265 ページの『矛盾するコンパイラー・オプション』

SSRANGE

SSRANGE を使用すると、添え字 (ALL 添え字を含む) または指標がテーブルの領域外の領域を参照しようとしているかどうかを検査するコードを生成することができます。それぞれの添え字または指標は、個別に妥当性を検査されるわけではありません。むしろ、テーブルの領域外の領域を参照しないようにするために、有効アドレスが検査されます。

定義された最大長の範囲内で参照を行うようにするために、可変長項目も検査されます。



デフォルト: NOSSRANGE

省略形: SSRINOSSR

次の点を確認するために、参照変更式が検査されます。

- 開始位置が 1 以上である。
- 開始位置が、サブジェクト・データ項目の現在の長さより大きくない。
- 長さの値 (指定されている場合) が 1 以上である。
- 開始位置と長さの値 (指定されている場合) がサブジェクト・データ項目の終わりを越えた領域を参照していない。

SSRANGE がコンパイル時に有効であると、範囲検査コードが生成されます。

CHECK(OFF) ランタイム・オプションを指定すれば、範囲検査を抑制することができます。そうすれば、範囲検査コードはオブジェクト・コードで休止状態になります。オプションとして、範囲検査コードを使用し、予期しないエラーを再コンパイルせずに解決するときに、役立てることもできます。

範囲外条件が検出されると、エラー・メッセージが生成され、プログラムは終了します。

重要: 範囲検査が行われるのは、プログラムを `SSRANGE` オプションを指定してコンパイルし、かつ `CHECK(ON)` オプションを指定して実行した場合のみです。

関連概念

108 ページの『参照修飾子』

関連タスク

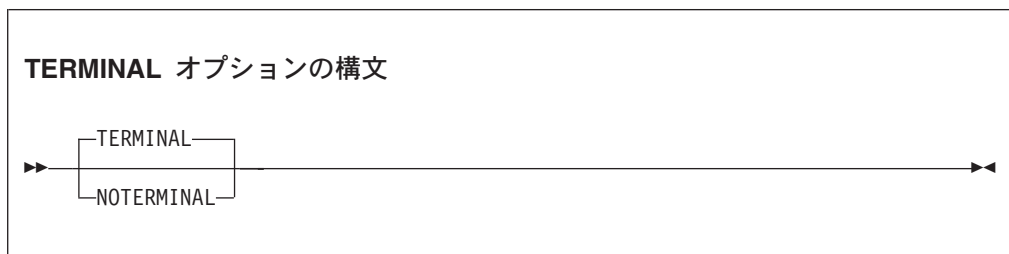
337 ページの『有効範囲の検査』

関連参照

325 ページの『CHECK』

TERMINAL

`TERMINAL` を使用すると、進行メッセージと診断メッセージをディスプレイ装置に送ることができます。



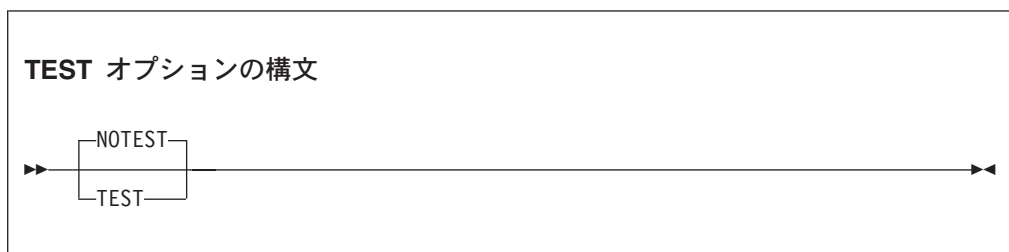
デフォルト: `TERMINAL`

省略形: `TERMINOTERM`

この追加の出力が不要な場合は、`NOTERMINAL` を使用してください。

TEST

`TEST` を使用すると、デバッガーがシンボリック・ソース・レベルのデバッグを実行できるようにするシンボルおよびステートメントの情報が入ったオブジェクト・コードが生成されます。



デフォルト: NOTEST

省略形: なし

デバッグ情報が入ったオブジェクト・コードを生成しない場合は、NOTEST を使用します。NOTEST を使用してコンパイルされたプログラムは、デバッガーで実行されますが、デバッグ・サポートは限られています。

WITH DEBUGGING MODE 節を使用した場合には、TEST オプションはオフになります。TEST は、オプション・リストに表示されますが、診断メッセージが出されて、競合のため TEST が無効である旨が通知されます。

関連参照

265 ページの『矛盾するコンパイラー・オプション』

THREAD

THREAD は、COBOL アプリケーションで、複数のスレッドを持つ実行単位での実行が有効化されることを示します。



デフォルト: NOTHREAD

省略形: なし

実行単位内のすべてのプログラムは、同じオプション (THREAD または NOTHREAD) を使用してコンパイルされる必要があります。

THREAD オプションが有効化されている場合には、次のエレメントはサポートされません。これらの言語エレメントが検出された場合は、エラーとして診断されます。

- ALTER ステートメント
- DEBUG-ITEM 特殊レジスター
- プロシージャ名が指定されていない GO TO ステートメント
- PROGRAM-ID 節の INITIAL 句
- RERUN
- 分割モジュール
- STOP リテラル・ステートメント
- STOP RUN
- USE FOR DEBUGGING ステートメント

使用上の注意

- RERUN は、THREAD ではエラーとしてフラグを立てられますが、NOTHREAD ではコメントとして許可されます。
- THREAD オプションを使用する場合には、以下のコマンドを使用してコンパイルします。

- cob2_j。オブジェクト指向 (OO) クライアントおよびクラス用です
- cob2_r。非 CICS プログラム用です

COBOL プログラムをコンパイルするときに、cob2_j を使用すると、THREAD オプションが自動的にデフォルトの呼び出しオプションとして組み込まれます。

- **CICS:** cob2_r コマンドおよび NOTHREAD オプションを使用して、CICS プログラムをコンパイルします。

パフォーマンスの考慮: THREAD オプションを使用する場合は、自動的に生成される逐次化ロジックのオーバーヘッドが原因で、実行時のパフォーマンスが多少低下することがあります。

関連タスク

236 ページの『コマンド行からのコンパイル』

545 ページの『第 29 章 マルチスレッド化のための COBOL プログラムの準備』

TRUNC

TRUNC は、バイナリー・データが移動および算術演算時に切り捨てられる方法に影響を与えます。



デフォルト: TRUNC(STD)

省略形: なし

TRUNC は、COMP-5 データ項目には効力を持ちません。COMP-5 項目は、TRUNC サブオプションの指定に関係なく、TRUNC(BIN) が有効である場合と同様に処理されます。

TRUNC(STD)

TRUNC(STD) は、MOVE ステートメントおよび算術式の中の USAGE BINARY 受信フィールドにのみ適用されます。TRUNC(STD) が有効であると、算術式の最終結果または MOVE ステートメント中の送信フィールドは、BINARY 受信フィールドの PICTURE 節の桁数に切り捨てられます。

TRUNC(OPT)

TRUNC(OPT) はパフォーマンス・オプションです。TRUNC(OPT) が有効であると、コンパイラーは、データが MOVE ステートメントおよび算術式にある USAGE BINARY 受信フィールドの PICTURE の指定に従うものと想定します。結果は最適な方法で処理され、PICTURE 節の中の桁数か、またはストレージ内の 2 進数フィールドのサイズ (ハーフワード、フルワード、またはダブルワード) に切り捨てられます。

ヒント: TRUNC(OPT) オプションを使用するのは、2 進数区域に移動されるデータが、2 進数項目に対する PICTURE 節で定義された値よりも高い精度の値にならないことが確実である場合に限定してください。そうしないと、結果は予測できません。この切り捨ては、想定される最も効果的な方法で実行されます。そのため、結果は、生成される特定のコード・シーケンスに左右されます。特定のステートメントに対して生成されたコード・シーケンスを見なければ、切り捨ての予想は不可能です。

TRUNC(BIN)

TRUNC(BIN) オプションは、USAGE BINARY データを処理するすべての COBOL 言語に適用されます。TRUNC(BIN) が有効な場合、すべての 2 進数項目 (USAGE COMP、COMP-4、または BINARY) は、固有ハードウェア 2 進数項目として、すなわち、それぞれが個々に USAGE COMP-5 と宣言されたものとして処理されます。

- BINARY 受信フィールドは、ハーフワード、フルワード、またはダブルワード境界でのみ切り捨てられます。
- BINARY 送信フィールドは、受け取り側が数値であれば、ハーフワード、フルワード、またはダブルワードとして処理されます。受け取り側が数値でない場合、TRUNC(BIN) は効力を持ちません。
- フィールドの全 2 進内容が重要です。
- DISPLAY は切り捨てを行わずに、2 進フィールドの内容全体を変換します。

推奨事項: 他のプロダクトによって設定される 2 進値を使用するプログラムの場合、推奨オプションは TRUNC(BIN) です。他のプロダクト (DB2、C/C++、および PL/I など) は、COBOL の 2 進数データ項目に、データ項目の PICTURE 節に従わない値を入れることがあります。データが BINARY データ項目用の PICTURE 節に矛盾しない場合は、CICS プログラムで TRUNC(OPT) を使用することができます。

USAGE COMP-5 には、個々のデータ項目に TRUNC(BIN) の性質を適用する効果があります。したがって、すべての 2 進数データ項目に対して TRUNC(BIN) を使用することによるパフォーマンス上のオーバーヘッドは、非 COBOL プログラムまたは他のプロダクトやサブシステムに渡されるデータ項目など、一部の 2 進数データ項目にのみ COMP-5 を指定することによって回避できます。COMP-5 の使用は、どの TRUNC サブオプションが有効であっても影響を受けません。

VALUE 節における大きなリテラル: コンパイラー・オプション TRUNC(BIN) を使用する場合、2 進数データ項目 (COMP、COMP-4、または BINARY) 用の VALUE 節に指定された数字リテラルは、PICTURE 節の 9 の数により暗黙指

定される値に制限されることはなく、通常、固有 2 進表現 (2、4、または 8 バイト) の容量までの大きさの値を持つことができます。

TRUNC の例 1

```
01 BIN-VAR      PIC S99 USAGE BINARY.
   . . .
   MOVE 123451 to BIN-VAR
```

次の表に、MOVE 後のデータ項目の値を示します。

データ項目	10 進数	16 進数	表示
送り出し側	123451	00101E2I3B	123451
受け取り側 TRUNC(STD)	51	00I33	51
受け取り側 TRUNC(OPT)	-7621	E2I3B	2J
受け取り側 TRUNC(BIN)	-7621	E2I3B	762J

ハーフワードのストレージが BIN-VAR に割り振られます。プログラムが TRUNC(STD) オプションでコンパイルされた場合は、この MOVE ステートメントの結果は 51 で、フィールドは、PICTURE 節に適合するように切り捨てられます。

プログラムが TRUNC(BIN) オプションでコンパイルされた場合、MOVE ステートメントの結果は -7621 です。このような異常に見える結果になるのは、非ゼロの高位桁が切り捨てられたためです。ここでは、生成されたコード・シーケンスは、下位のハーフワード量 X'E23B' を受け取り側に移動させるだけです。切り捨てられた新しい値はオーバーフローして 2 進数ハーフワードの符号ビットになるため、値が負の数になります。

123451 は BIN-VAR の PICTURE 節より高い精度を持つため、この MOVE ステートメントを TRUNC(OPT) オプションでコンパイルしてはなりません。TRUNC(OPT) を使用した場合も、結果は -7621 になります。これは、10 進数の切り捨てを行わないことによって、最高のパフォーマンスが得られたからです。

TRUNC の例 2

```
01 BIN-VAR      PIC 9(6) USAGE BINARY
   . . .
   MOVE 1234567891 to BIN-VAR
```

次の表に、MOVE 後のデータ項目の値を示します。

データ項目	10 進数	16 進数	表示
送り出し側	1234567891	49I96I02ID3	1234567891
受け取り側 TRUNC(STD)	567891	00I08IAAI53	567891
受け取り側 TRUNC(OPT)	567891	53IAAI08I00	567891

データ項目	10 進数	16 進数	表示
受け取り側 TRUNC(BIN)	1234567891	491961021D3	1234567891

TRUNC(STD) を指定すると、送り出しデータは BINARY 受け取り側の PICTURE 節に適合するように、6 桁の整数に切り捨てられます。

TRUNC(OPT) を指定すると、コンパイラーは送り出しデータの精度が BINARY 受け取り側の PICTURE 節の精度よりも大きくないと想定します。この場合、最も効率のよいコード・シーケンスは、TRUNC(STD) が指定されているものとして切り捨てを行うことです。

TRUNC(BIN) を指定すると、BIN-VAR に割り振られた 2 進数フルワードにすべての送り出しデータが収まるため、切り捨ては行われません。

関連概念

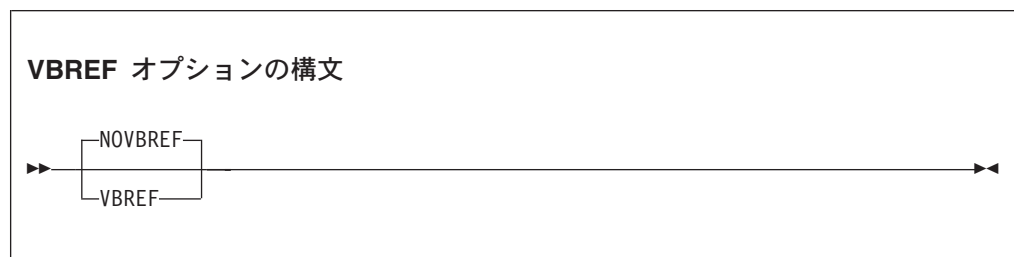
45 ページの『数値データの形式』

関連参照

VALUE 節 (COBOL for AIX 言語解説書)

VBREF

VBREF は、ソース・プログラムの中で使用されるすべての動詞、およびこれらの動詞が使用されている行番号の相互参照を入手するために使用します。また、VBREF はプログラムの中でそれぞれの動詞が使用された回数の合計も出します。



デフォルト: NOVBREF

省略形: なし

コンパイルの効率を高める場合は、NOVBREF を使用してください。

WSCLEAR

WSCLEAR を使用すると、プログラムの初期化時に、そのプログラムの WORKING-STORAGE をクリアして 2 進ゼロにすることができます。ストレージは、VALUE 文節が適用される前にクリアされます。

WSCLEAR オプションの構文



デフォルト: NOWSCLEAR

省略形: なし

ストレージのクリア処理をバイパスするには、NOWSCLEAR を使用します。

パフォーマンスの考慮事項: WSCLEAR を使用するとき、オブジェクト・プログラムのサイズやパフォーマンスが懸念される場合は、OPTIMIZE(FULL) も使用することをお勧めします。これにより、DATA DIVISION から未参照のデータ項目をすべて除去するようコンパイラーに命令が出されるため、初期化の時間が短縮されます。

XREF

XREF は、ソート済みの相互参照リストを入手するために使用します。

XREF オプションの構文



デフォルト: XREF(FULL)

省略形: XINOX

XREF、XREF(FULL)、または XREF(SHORT) を選択できます。サブオプションを何も指定しないで XREF を指定すると、XREF(FULL) が有効です。

プログラム内で参照されるすべてのプログラム名、データ名、およびプロシージャ名、およびそれらが定義されている行番号を表示するリストにセクションが含まれています。外部プログラム名が識別されます。

また、関連コピーブックを取得したファイルでプログラム内の COPY または BASIS ステートメントを相互参照するセクションも含まれています。

ロケール設定によって指定された照合シーケンスの順序で、名前はリストされます。名前が 1 バイト文字であっても、マルチバイト文字 (DBCS など) を含んでいても、この順序が使用されます。

XREF と SOURCE を使用した場合は、データ名とプロシージャー名の相互参照情報が元のソースと同じ行に印刷されます。行番号参照またはその他の情報は、リスト・ページの右側に表示されます。組み込み関数を参照するソース行の右側には、IFN という文字と、その関数の引数が定義されている場所の行番号が印字されます。組み込み参照に含められた情報によって、ID が未定義であるか (UND)、複数回定義されているか (DUP)、項目が暗黙定義であるか (IMP) (特殊レジスターや形象定数など)、プログラム名が外部プログラム名であるか (EXT) がわかります。

XREF と NOSOURCE を使用すると、ソート済みの相互参照リストだけが得られます。

XREF(SHORT) は、相互参照リスト内の明示的に参照されたデータ項目だけを印刷します。XREF(SHORT) は、マルチバイトデータ名とプロシージャー名、および単一バイトの名前に適用されます。

NOXREF を使用すると、このリストは抑止されます。

使用上の注意

- MOVE CORRESPONDING ステートメントで使用されるグループ名は、XREF リストに入れられます。それらのグループの基本名もリストされています。
- データ名の XREF リストでは、文字 M が前に付いている行番号は、そのデータ項目がその行のステートメントによって明示的に変更されたことを示しています。
- XREF リストは追加のストレージを使用します。

関連概念

331 ページの『第 17 章 デバッグ』

関連タスク

342 ページの『リストの入手』

YEARWINDOW

YEARWINDOW を使用すると、COBOL コンパイラーによるウィンドウ表示日付フィールド処理に適用する 100 年ウィンドウ (世紀ウィンドウ) の最初の年を指定することができます。

YEARWINDOW オプションの構文

▶▶—YEARWINDOW(*base-year*)————▶▶

デフォルト: YEARWINDOW(1900)

省略形: YW

base-year は、世紀ウィンドウの最初の年を表します。次のいずれかの値で指定します。

- 1900 から 1999 の間の符号なし 10 進数

これは固定ウィンドウの開始年号を指定します。例えば、YEARWINDOW(1930) は 1930 から 2029 年の世紀ウィンドウを指定します。

- -1 から -99 の負の整数

これは、スライディング・ウィンドウを示します。ウィンドウの最初の年は、現在の年に負の整数を加えて計算されます。例えば、YEARWINDOW(-80) は、世紀ウィンドウの最初の年がプログラム実行時点の年号より 80 年前であることを指定します。

使用上の注意

- YEARWINDOW オプションは、DATEPROC オプションも有効でない限り、効力を持ちません。
- 実行時には、次の 2 つの条件が真でなければなりません。
 - 世紀ウィンドウの開始年号が 1900 年代の年号である。
 - 現在の年号がコンパイル単位の世紀ウィンドウ内にある。

例えば、現在の年号が 2008 年で、DATEPROC オプションが有効な場合に、YEARWINDOW(1900) というオプションを指定すると、プログラムは終了し、エラー・メッセージが出されます。

ZWB

ZWB を使用してコンパイルすると、コンパイラーは、実行時に符号付きゾーン 10 進数 (DISPLAY) フィールドを英数字基本フィールドと比較する前に、そのフィールドから符号を除去します。

ZWB オプションの構文



デフォルト: ZWB

省略形: なし

ゾーン 10 進数項目がスケール項目である場合 (すなわち、記号 P をその PICTURE スtring内を含んでいる場合)、比較の際にその項目を使用しても、ZWB の影響を受けることはありません。そのような項目では常に、英数字フィールドとの比較が行われる前に符号が除去されます。

ZWB はプログラムの実行方法に影響します。同じ COBOL ソース・プログラムでも、このオプションの設定によって結果が異なることがあります。

NOZWB は、入力数字フィールドで SPACES をテストする場合に使用します。

第 15 章 コンパイラー指示ステートメント

プログラムのコンパイルを指示するには、いくつかのコンパイラー指示ステートメントと 1 つのディレクティブが役に立ちます。

以下は、コンパイラー指示ステートメントおよびディレクティブです。

*CONTROL (*CBL) ステートメント

このコンパイラー指示ステートメントは、出力の作成を抑制するかまたは可能にするかを選択します。キーワードの *CONTROL と *CBL は同義語です。

>>CALLINTERFACE 指示

このコンパイラー指示は、引数記述子を生成するかどうかなど、呼び出しのインターフェース規約を指定します。>>CALLINTERFACE を使用して指定された規約は、別の >>CALLINTERFACE 指定が行われるまで有効です。>>CALLINT は、>>CALLINTERFACE の略語です。

>>CALLINTERFACE を使用できるのは、PROCEDURE DIVISION の中だけです。

>>CALLINTERFACE ディレクティブの構文と使用法は、CALLINT コンパイラー・オプションと類似しています。例外は次のとおりです。

- この指示構文には括弧が含まれません。
- この指示は、後述のように、選択した呼び出しに適用できます。
- この指示構文には、キーワード DESCRIPTOR とその変形が含まれます。

サブオプションなしで >>CALLINT を指定すると、使用される呼び出し規約が CALLINT コンパイラー・オプションで区切られます。

DESCRIPTOR のみ: >>CALLINT ディレクティブは、以下の形式の場合以外はコメントとして扱われます。

- >>CALLINT SYSTEM DESCRIPTOR、または同等に >>CALLINT DESCRIPTOR
- >>CALLINT SYSTEM NODESCRIPTOR、または同等に >>CALLINT NODESCRIPTOR

これらのディレクティブは、DESCRIPTOR のオン/オフを切り替えます。SYSTEM は無視されます。

SYSTEM の代わりに OPTLINK または CDECL (COBOL for Windows の有効な値) をコーディングすると、コンパイラーが I レベルの診断を生成し、ディレクティブ全体 (最初のキーワードだけでなく) が無視されます。

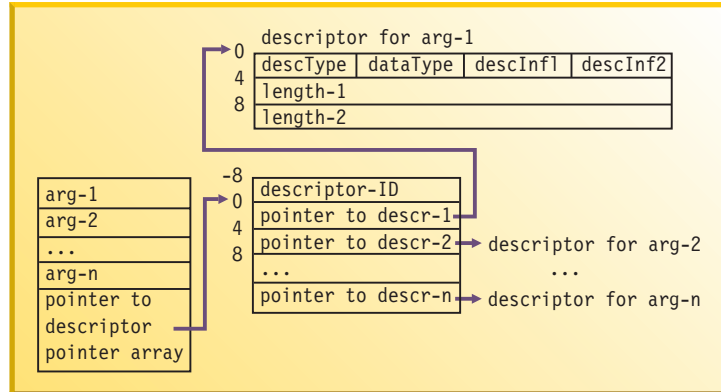
>>CALLINT 指示を指定する場所は、COBOL プロシージャ・ステートメントが指定可能な場所であればどこでも構いません。例えば、次の構文は有効です。

```
MOVE 3 TO  
>>CALLINTERFACE SYSTEM  
RETURN-CODE.
```

>>CALLINT の影響は、現行のプログラムに限定されます。ネストされたプログラム、または同じバッチ内でコンパイルされたプログラムは、>>CALLINT コンパイラー指示で指定された規則ではなく、CALLINT コンパイラー・オプションを使用して指定された呼び出し規約を継承します。

>>CALLINT SYSTEM DESCRIPTORを使用して呼び出されるルーチンを記述する場合、引数受け渡しの仕組みは次のようになります。

CALL "PROGRAM1" USING arg-1, arg-2, ... arg-n



pointer to descr-n

特定の引数に対する記述子を指します。引数に対する記述子が存在しない場合は 0 になります。

descriptor-ID

このバージョンの記述子を識別するには、COBDESC0 に設定します。これにより、記述子の入力形式が将来変更される場合にも対応できるようになります。

descType

PICTURE X(n) を使用した USAGE DISPLAY、あるいは PICTURE G(n) または N(n) を使用した USAGE DISPLAY-1 の基本データ項目の場合は、X'02' (descElmt) に設定します。それ以外 (数値フィールド、構造体、テーブル) の場合はすべて、X'00' に設定します。

dataType

次のように設定します。

- descType = X'00' の場合: dataType = X'00'
- descType = X'02' で、USAGE が DISPLAY の場合: dataType = X'02' (typeChar)
- descType = X'02' で、USAGE が DISPLAY-1 の場合: dataType = X'09' (typeGChar)

descInf1

常に X'00' に設定します。

descInf2

次のように設定します。

- descType = X'00' の場合: descInf2 = X'00'
- descType = X'02' の場合:
 - CHAR(EBCDIC) オプションが有効で、USAGE 文節内の NATIVE オプションで引数が定義されていない場合: descInf2 = X'40'
 - それ以外の場合: descInf2 = X'00'

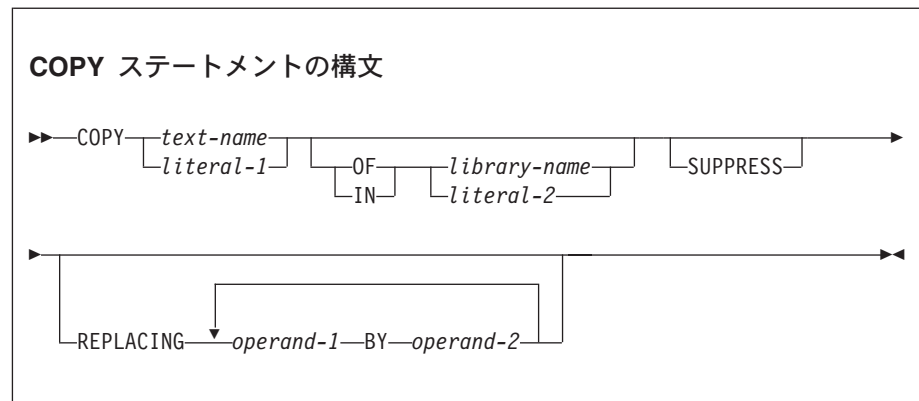
length-1

引数記述子には、固定長引数の引数長か、可変長項目の現行の長さが入ります。

length-2

引数が可変長項目の場合は、その引数の最大長を示します。固定長引数の場合、length-2 と length-1 は等しくなります。

COPY ステートメント



このコンパイラ指示ステートメントは、事前に作成されたテキストを COBOL プログラムに入れます。事前に作成されたテキストを含む *text-name* (コピーブックの名前) を指定する必要があります。例えば、COPY *my-text* のようになります。 *text-name* に修飾子として *library-name* を付けることができます。例えば、COPY *my-text of inventory-lib* のようになります。 *text-name* が修飾されない場合は、SYSLIB の *library-name* が想定されます。 *library-name* および *text-name* に影響を与える事項は次のとおりです。

library-name

library-name をリテラルとして指定した場合、リテラルの内容は実際のパスとして扱われます。ユーザー定義語として *library-name* を指定した場合は、名前が環境変数として使用され、この環境変数の値がコピーブックの位置を指定するパスに使用されます。複数のパス名を指定するには、各パス名をコロン (:) またはセミコロン (;) で区切ります。

library-name を指定しない場合、このパスは『*text-name*』に説明するとおり使用されます。

text-name

text-name をユーザー定義語として指定した場合、処理は *text-name* に対応する環境変数が設定されているかどうかによって異なります。環境変数が設定されている場合、環境変数の値はコピーブックのファイル名 (およびパス名) として使用されます。

次の 3 つの条件がすべて満たされる場合は、 *text-name* が絶対パスとして扱われます。

- *library-name* は使用されません。
- *text-name* がリテラル、または環境変数である。

- 最初の文字が *'* である。

例えば、次のパスは絶対パスとして扱われます。

```
COPY "/mycpylib/mytext.cpy"
```

text-name に対応する環境変数が設定されていない場合、コピーブックは次の名前で検索されます。

1. 接尾部が *.cpy* の *text-name*
2. 接尾部が *.cbl* の *text-name*
3. 接尾部が *.cob* の *text-name*
4. 接尾部のない *text-name*

例えば、COPY MyCopy は次の順序で検索を行います。

1. MYCOPY.cpy (前述のとおり、指定されたすべてのパス内)
2. MYCOPY.cbl (前述のとおり、指定されたすべてのパス内)
3. MYCOPY.cob (前述のとおり、指定されたすべてのパス内)
4. MYCOPY (前述のとおり、指定されたすべてのパス内)

COBOL は、その名前がリテラルに含まれている場合 (「MyCopy」) 以外、デフォルトでは *library-name* と *text-name* を大文字にします。この例では、MyCopy と MYCOPY は同じではありません。ファイル名が大/小文字混合 (MyCopy.cbl など) の場合は、*text-name* を COPY ステートメントでリテラルとして定義します。

-I オプション

それ以外の場合 (*library-name* も *text-name* もパスを示さない場合)、検索パスは -I オプションに依存します。

COPY A と COPY A OF MYLIB を等しくするには、-I\$MYLIB を指定します。

上記の規則に基づくと、COPY "/X/Y" はルート・ディレクトリー内で検索され、COPY "X/Y" は現行ディレクトリー内で検索されます。

COPY A OF SYSLIB は COPY A と同等です。-I オプションは、*library-name* 修飾が明示的に指定された COPY ステートメントや、ライブラリー名が SYSLIB のステートメントには影響しません。

library-name と *text-name* の両方を指定すると、コンパイラーは、*library-name* の末尾が *'* でない場合に、2 つの値の間にパス区切り文字 (*/*) を挿入します。例えば、COPY MYCOPY OF MYLIB を次のように設定します。

```
export MYCOPY=MYPDS(MYMEMBER)
export MYLIB=MYFILE
```

この場合、結果として MYFILE/MYPDS(MYMEMBER) が作成されます。

text-name をユーザー定義語として指定した場合は、ローカル・ファイルにアクセスすることができ、メインフレームのソースを変更せずに z/OS 上の PDS メンバーにアクセスすることもできます。以下に例を示します。

```
COPY mycopybook
```

この例では、環境変数 *mycopybook* が h/mypds(mycopy) に設定されているときは、次のようになります。

- h は特定のホストに割り当てられます。
- mypds は z/OS PDS データ・セット名です。
- mycopy は PDS メンバー名です。

NFS (ネットワーク・ファイル・システム) を使用して、AIX から z/OS ファイルにアクセスすることができます。NFS により、AIX パス名を使用して z/OS ファイルにアクセスすることができます。ただし、NFS では、z/OS 命名規則に従うためにパス区切り文字が「.」に変換されます。環境変数に値を割り当てるときはこのことに注意して、名前の形式を必ず適切なものにしてください。例えば、次のように設定します。

```
export MYCOPY=(MYMEMBER)
export MYLIB=M/MYFILE/MYPDS
```

結果として次のパスが作成されるため、これは機能しません。

```
M/MYFILE/MYPDS/(MYMEMBER)
```

これは次のようになります。

```
M.MYFILE.MYPDS.(MYMEMBER)
```

PROCESS (CBL) ステートメント

このコンパイラ指示ステートメントは、最も外側の IDENTIFICATION DIVISION ヘッダーの前に置かれるもので、プログラムのコンパイル時に使用されるコンパイラ・オプションを指定します。

関連タスク

7 ページの『ソース・リストのヘッダーの変更』

236 ページの『コマンド行からのコンパイル』

238 ページの『PROCESS (CBL) ステートメントによるコンパイラ・オプションの指定』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

CALLINTERFACE ディレクティブ (*COBOL for AIX* 言語解説書)

CBL (PROCESS) ステートメント (*COBOL for AIX* 言語解説書)

*CONTROL (*CBL) ステートメント (*COBOL for AIX* 言語解説書)

COPY ステートメント (*COBOL for AIX* 言語解説書)

第 16 章 ランタイム・オプション

サポートされているランタイム・オプションを次の表に示します。

表 38. ランタイム・オプション

オプション	説明	デフォルト	省略形
『CHECK』	エラー検査のフラグを立てます。	CHECK(ON)	CH
326 ページの『DEBUG』	USE FOR DEBUGGING 宣言で指定された COBOL デバッグ・セクションがアクティブかどうかを指定します。	NODEBUG	なし
326 ページの『ERRCOUNT』	重大度 1 (W レベル) の条件が何回発生すると実行単位が異常終了するかを指定します。	ERRCOUNT(20)	なし
327 ページの『FILEMODE』	COBOL for AIX プログラムによって作成された STL ファイルと VSAM ファイルが大容量形式内部表現を持つのか小容量形式内部表現を持つのかを指定します。これによって、STL ファイルと VSAM ファイルの最大サイズ制限が決定されます。	FILEMODE(LARGE)	FM(L S)
328 ページの『FILESYS』	ASSIGN または環境変数のいずれかを通して、明示的なファイル・システム選択が行われないファイルに使用されるファイル・システムを指定します。	FILESYS(VSA)	なし
328 ページの『TRAP』	COBOL が例外を代行受信するかどうかを指定します。	TRAP(ON)	なし
329 ページの『UPSI』	COBOL ルーチンを使用するアプリケーションに対して、8 つの UPSI スイッチのオン/オフを設定します。	UPSI(00000000)	なし

CHECK

CHECK により、エラー検査にフラグが設定されます。COBOL では、索引、添え字、参照変更範囲によって、エラー検査が行われます。

CHECK オプションの構文

▶▶—CHECK(—ON
 OFF—)—▶▶

デフォルト: CHECK(ON)

省略形: CH

ON ランタイム検査を実行することを指定します。

OFF ランタイム検査を実行しないことを指定します。

使用上の注意: コンパイル時に NOSSRANGE が有効だった場合は、CHECK(ON) が無効になります。

パフォーマンスの考慮事項: SSRANGE を使用して COBOL プログラムをコンパイルした後、アプリケーションのテストまたはデバッグを行わない場合は、CHECK(OFF) を指定するとパフォーマンスが向上します。

DEBUG

DEBUG は、USE FOR DEBUGGING 宣言で指定された COBOL デバッグ・セクションがアクティブかどうかを指定します。

DEBUG オプションの構文



デフォルト: NODEBUG

DEBUG デバッグ・セクションをアクティブにします。

NODEBUG

デバッグ・セクションを抑制します。

パフォーマンスについての考慮事項: パフォーマンスを高めるには、このオプションをデバッグ時だけに使用してください。

ERRCOUNT

ERRCOUNT は、重大度 1 (W レベル) の条件が何回発生すると実行単位が異常終了するかを指定します。

ERRCOUNT オプションの構文



デフォルト: ERRCOUNT(20)

number は、この実行単位の実行中に、発生し得る重大度 1 の条件の数です。条件数が *number* を超えると、実行単位が異常終了します。

重大度 2 (E レベル) 以上の条件が発生すると、ERRCOUNT オプションの値に関係なく、実行単位が終了します。

FILEMODE

FILEMODE は、OPEN OUTPUT ステートメントを使用して作成された STL ファイルと VSAM ファイルが持つメタデータ (内部制御情報) が大容量形式なのか小容量形式なのかを指定します。これによって、これらの最大サイズが決定されます。

FILEMODE オプションの構文

```
▶▶ FILEMODE( [LARGE | SMALL] ) ▶▶
```

デフォルト: FILEMODE(LARGE)

省略形: FM(L|S)

LARGE STL ファイルと VSAM ファイルに対する OPEN OUTPUT ステートメントにより大容量形式のファイルが作成されることを指定します。このようなファイルには、基本的にサイズの制限がありません。

SMALL STL ファイルと VSAM ファイルに対する OPEN OUTPUT ステートメントにより小容量形式のファイルが作成されることを指定します。小容量形式のファイルの最大サイズは 2GB です。

大容量形式のファイルおよび小容量形式のファイルは、32 ビットと 64 ビットのどちらの COBOL for AIX プログラムを使用しても処理できます。ファイル処理は、ADDR コンパイラー・オプションの値に影響を受けません。

COBOL for AIX バージョン 2 のプログラムは、小容量形式のファイルのみを作成できます。バージョン 2 のプログラムは、より新しいランタイム・ライブラリーを使用して実行される場合でも、FILEMODE ランタイム・オプションを無視します。

FILEMODE は、行順次ファイルや RSD ファイルの処理には影響しません。これらのファイルには最大サイズ制限がありません。

関連概念

130 ページの『2 GB を超える STL ファイルおよび VSAM ファイル』

関連参照

125 ページの『VSAM ファイル・システム』

127 ページの『STL ファイル・システム』

131 ページの『ファイル・システム要約』

266 ページの『ADDR』

コンパイラー限界値 (COBOL for AIX 言語解説書)

FILESYS

FILESYS は、ASSIGN ステートメントまたは環境変数のいずれかを通して、明示的なファイル・システム選択が行われないファイルに使用されるファイル・システムを指定します。このオプションは、順次ファイル、相対ファイル、および索引付きファイルに適用されます。

FILESYS オプションの構文

```
▶▶—FILESYS( [VSA | STL | RSD] )————▶▶
```

Default is: FILESYS(VSA)

VSA ファイル・システムは VSAM です。Encina SFS ファイルを使用したい場合は、VSAM ファイル・システムを指定します。

STL ファイル・システムは STL です。

RSD ファイル・システムは RSD です。

使用されるのはファイル名識別子の最初の 3 つの文字だけで、この識別子は大/小文字を区別しません。例えば、以下の例はすべて、VSAM に対して有効な指定です。

- FILESYS(VSA)
- FILESYS(vSAM)
- FILESYS(vsa)

関連タスク

121 ページの『ファイルの識別』

関連参照

228 ページの『ランタイム環境変数』

TRAP

TRAP は、COBOL が例外を代行受信するかどうかを指示します。

TRAP オプションの構文

```
▶▶—TRAP( [ON | OFF] )————▶▶
```

デフォルト: TRAP(ON)

TRAP(OFF) が有効であり、なおかつ例外条件を処理する独自のトラップ・ハンドラーを使用しない場合は、これらの条件が発生すると、オペレーティング・システムによるデフォルトのアクションが実行されます。例えば、プログラムが無許可の場所にデータを格納しようとする、デフォルトのシステム・アクションとして、メッセージが出されて処理が終了します。

ON COBOL による例外の代行受信をアクティブにします。

OFF COBOL による例外の代行受信を非アクティブにします。

使用上の注意

- TRAP(OFF) を使用するのには、プログラムの例外を COBOL が処理する前に分析する必要があります。必要がある場合だけにしてください。
- 非 CICS 環境で TRAP(OFF) を指定すると、例外ハンドラーが設定されません。
- (例外診断の目的で) TRAP(OFF) を使用して実行すると、COBOL が TRAP(ON) を必要とするため、多数の副次作用が発生する可能性があります。TRAP(OFF) を使用して実行すると、ソフトウェアで発生した条件、プログラム・チェック、または異常終了が検出されない場合でも、副次作用が発生する可能性があります。TRAP(OFF) が有効な場合にプログラム・チェックまたは異常終了が検出されると、次の副次作用が発生する可能性があります。
 - COBOL によって取得されたリソースが解放されない。
 - COBOL によってオープンされたファイルがクローズしない。このため、レコードが失われる可能性があります。
 - メッセージまたはダンプ出力が生成されない。

このような条件が発生すると、実行単位が異常終了します。

UPSI

UPSI は、COBOL ルーチンを使用するアプリケーションに対して、8 つの UPSI スイッチのオン/オフを設定します。

UPSI オプションの構文



```
▶▶—UPSI( [nnnnnnnn] )————▶▶
```

デフォルト: UPSI(00000000)

それぞれの n は、UPSI スイッチ (0 から 7) のうちの 1 つを表します。左端の n が 1 番目のスイッチを表しています。それぞれの n は 0 (オフ) または 1 (オン) のいずれかになります。

第 17 章 デバッグ

アプリケーションのプログラム動作における問題の原因を判別するには、ソース言語デバッグと対話式デバッグの 2 つの方法を使用することができます。

ソース言語デバッグの場合、COBOL は、デバッグを容易にするいくつかの言語エレメント、コンパイラー・オプション、およびリスト出力を提供します。

対話式デバッグの場合は、Debugger for AIX を使用できます。

関連タスク

- 『ソース言語によるデバッグ』
- 336 ページの『コンパイラー・オプションを使用したデバッグ』
- 341 ページの『デバッガーの使用』
- 342 ページの『リストの入手』
- 353 ページの『オフセット情報を含むメッセージによるデバッグ』
- 353 ページの『ユーザー出口のデバッグ』
- 354 ページの『アセンブラー・ルーチンのデバッグ』

ソース言語によるデバッグ

さまざまな COBOL 言語機能を使用して、プログラムの障害の原因を正確に示すことができます。

障害のあるプログラムが既に実動中の大規模なアプリケーションの一部である場合 (ソース更新を除外する) は、プログラムの障害部分をシミュレートするような小さいテスト・ケースを作成してください。テスト・ケースでは、以下の問題の検出に役立つようなデバッグ機能をコーディングしてください。

- プログラム・ロジックのエラー
- 入出力エラー
- データ型のミスマッチ
- 初期化されていないデータ
- プロシーチャーの問題

関連タスク

- 332 ページの『プログラム・ロジックのトレース』
- 332 ページの『入出力エラーの検出および処理』
- 333 ページの『データの妥当性検査』
- 333 ページの『初期化されていないデータの検出』
- 334 ページの『プロシーチャーに関する情報の生成』

関連参照

ソース言語のデバッグ (COBOL for AIX 言語解説書)

プログラム・ロジックのトレース

プログラムのロジックは、DISPLAY ステートメントを追加することによってトレースしてください。

例えば、問題が EVALUATE ステートメントまたは 1 組のネストされた IF ステートメントにあると判断した場合は、それぞれのパスで DISPLAY ステートメントを使用して、ロジック・フローを調べます。問題の原因が数値の計算方法にあると判断した場合は、DISPLAY ステートメントを使用して、いくつかの中間結果の値を検査することができます。

プログラムの中で明示範囲終了符号を使用してステートメントを終了させるようにすると、ロジックはより明確であり、したがってトレースしやすくなります。

例えば、特定のルーチンが開始して終了したかどうかを判別する場合は、プログラムに次のようなコードを挿入してみてください。

```
DISPLAY "ENTER CHECK PROCEDURE"  
    .  
    . (checking procedure routine)  
    .  
DISPLAY "FINISHED CHECK PROCEDURE"
```

ルーチンが正しく作動していることを確認したら、次のいずれかの方法で DISPLAY ステートメントを使用不可にします。

- 各 DISPLAY ステートメントの行の 7 桁目にアスタリスクを置き、コメント行に変換する。
- 各 DISPLAY ステートメントの 7 桁目に D を置き、コメント行に変換する。これらのステートメントを再活動化したい場合は、ENVIRONMENT DIVISION に WITH DEBUGGING MODE 節を含めると、7 桁目の D は無視され、DISPLAY ステートメントが実施されます。

プログラムを実動に移す前に、使用したすべてのデバッグ・エイドを削除するか使用不可にしてから、プログラムを再コンパイルします。プログラムはより効果的に実行され、使用するストレージは小さくなります。

関連概念

21 ページの『範囲終了符号』

関連参照

DISPLAY ステートメント (*COBOL for AIX 言語解説書*)

入出力エラーの検出および処理

ファイル状況キーは、プログラムのエラーが、ストレージ・メディアで起こっている入出力エラーによるものかどうかを判別するのに役立ちます。

ファイル状況キーをデバッグ・エイドとして使用するためには、各入出力ステートメントの後で、状況キーの値がゼロ以外かどうか検査します。値がゼロでない (エラー・メッセージで報告される) 場合には、プログラム内の入出力プロシージャのコーディングを調べます。状況キーの値に基づいてエラーを訂正するためのプロシージャを組み込むこともできます。

問題がプログラムの入出力プロシージャにあると判断した場合は、USE EXCEPTION/ERROR 宣言を組み込んで、問題のデバッグに役立てることができます。その後、ファイルのオープンに失敗すると、適切な EXCEPTION/ERROR 宣言が実行されます。適切な宣言とは、ファイルに固有なもの、あるいはオープン属性 (INPUT、OUTPUT、I-O、または EXTEND) 用に提供されたものです。

各 USE AFTER STANDARD ERROR ステートメントを、PROCEDURE DIVISION 中の DECLARATIVES キーワードに続くセクションにコーディングします。

関連タスク

173 ページの『ERROR 宣言のコーディング』

174 ページの『ファイル状況キーの使用』

関連参照

状況キー (COBOL for AIX 言語解説書)

データの妥当性検査

プログラムが非数値データに対して算術を実行しようとしているか、または入力レコードの誤ったデータ型を受け取ろうとしている可能性がある場合、クラス・テスト (クラス条件) を使用してデータ型を妥当性検査してください。

クラス・テストを使用すると、データ項目の内容が、ALPHABETIC、ALPHABETIC-LOWER、ALPHABETIC-UPPER、DBCS、KANJI、または NUMERIC のいずれであるかを検査できます。データ項目が暗黙的または明示的に USAGE NATIONAL として記述されている場合、クラス・テストは、指定された文字クラスに関連した文字の国別文字表現を検査します。

関連タスク

90 ページの『条件式のコーディング』

207 ページの『有効な DBCS 文字に関するテスト』

関連参照

クラス条件 (COBOL for AIX 言語解説書)

初期化されていないデータの検出

問題の原因がテーブルまたはデータ項目のフィールドに残された残余データにあると考えられるときは、INITIALIZE または SET ステートメントを使用して、テーブルまたはデータ項目を初期化してください。

問題が起きたり起きなかったりし、しかも同一のデータで起きるとは限らない場合、スイッチが初期化されていないが多くの場合正しい値 (0 または 1) に偶然に設定されることが原因であると考えられます。SET ステートメントを使用してスイッチを初期化するようにすれば、初期化されていないスイッチが問題の原因であると判断できるか、考えられる原因からそのスイッチを除外することができます。

関連参照

INITIALIZE ステートメント (COBOL for AIX 言語解説書)

SET ステートメント (COBOL for AIX 言語解説書)

プロシージャーに関する情報の生成

プログラムまたはテスト・ケースおよびその実行方法に関する情報を生成するには、USE FOR DEBUGGING 宣言をコーディングします。この宣言を使用すると、ステートメントをプログラムに組み込んで、プログラムの実行時にいつそれらのステートメントを実行しなければならないかを指示することができます。

例えば、プロシージャーが何度実行されるかを決定するには、デバッグ・プロシージャーを USE FOR DEBUGGING 宣言に組み込み、カウンターを使用して、制御がそのプロシージャーに渡される回数の記録をとることができます。カウンター技法を使用して、次のような項目を検査できます。

- PERFORM ステートメントが実行される回数。特定のルーチンが使用されているかどうか、および制御構造が正しいかどうか。
- ループ・ルーチンが実行される回数、ループが実行されているかどうか、およびループの回数が正確かどうか。

プログラムに、デバッグ行かデバッグ・ステートメント、またはその両方を入れることができます。

デバッグ行 は、桁 7 の D で識別されているステートメントです。プログラムのデバッグ行をアクティブにするには、ENVIRONMENT DIVISION の SOURCE-COMPUTER 行に WITH DEBUGGING MODE 節をコーディングする必要があります。この節が含まれていないと、デバッグ行はコメントとしてしか扱われません。

デバッグ・ステートメント とは、PROCEDURE DIVISION の DECLARATIVES セクションにコーディングされたステートメントです。それぞれの USE FOR DEBUGGING 宣言は別個のセクションにコーディングしなければなりません。デバッグ・ステートメントは、次のようにコーディングしてください。

- DECLARATIVES セクション内のみ。
- ヘッダー USE FOR DEBUGGING の後に置く。
- 最外部のプログラムだけに置く (ネストされたプログラムでは無効です)。デバッグ・ステートメントが、ネストされたプログラムに含まれているプロシージャーによって起動されることはありません。

プログラムでデバッグ・ステートメントを使用するには、WITH DEBUGGING MODE 節を指定し、さらに、DEBUG ランタイム・オプションを使用する必要があります。

オプションに関する制約事項:

- THREAD オプションを指定してコンパイルするプログラムでは、USE FOR DEBUGGING 宣言を使用できません。
- USE FOR DEBUGGING 宣言は、WITH DEBUGGING MODE 節が指定されていると、TEST コンパイラー・オプションと相互に排他的な関係にあります。USE FOR DEBUGGING 宣言と WITH DEBUGGING MODE 節が存在していると、TEST オプションは取り消されます。

335 ページの『例: USE FOR DEBUGGING』

関連参照

SOURCE-COMPUTER 段落 (COBOL for AIX 言語解説書)

デバッグ行 (COBOL for AIX 言語解説書)
デバッグ・セクション (COBOL for AIX 言語解説書)
DEBUGGING 宣言 (COBOL for AIX 言語解説書)

例: USE FOR DEBUGGING

この例は、以下のプログラム・セグメントは、DISPLAY ステートメントと USE FOR DEBUGGING 宣言を使用してプログラムをテストする際に必要となるステートメントの種類を示しています。

DISPLAY ステートメントは、情報を端末または出力ファイルに書き込みます。USE FOR DEBUGGING 宣言は、ルーチンが実行される回数を示すカウンターと一緒に使用されます。

```
Environment Division.  
  . . .  
Data Division.  
  . . .  
Working-Storage Section.  
  . . . (other entries your program needs)  
01 Trace-Msg    PIC X(30) Value " Trace for Procedure-Name : ".  
01 Total        PIC 9(9)  Value 1.  
  . . .  
Procedure Division.  
Declaratives.  
Debug-Declaratives Section.  
  Use For Debugging On Some-Routine.  
Debug-Declaratives-Paragraph.  
  Display Trace-Msg, Debug-Name, Total.  
End Declaratives.  
  
Main-Program Section.  
  . . . (source program statements)  
  Perform Some-Routine.  
  . . . (source program statements)  
  Stop Run.  
Some-Routine.  
  . . . (whatever statements you need in this paragraph)  
  Add 1 To Total.  
Some-Routine-End.
```

プロシージャー `Some-Routine` が実行されるたびに、DECLARATIVES SECTION の DISPLAY ステートメントがこのメッセージを出します。

```
Trace For Procedure-Name : Some-Routine 22
```

メッセージの終わりにある番号 22 は、データ項目 `Total` の累算された値で、`Some-Routine` が実行された回数を示しています。デバッグ宣言内のステートメントは、名前を指定されたプロシージャーが実行される前に実行されます。

DISPLAY ステートメントを使用して、プログラムの実行をトレースし、プログラム中のフローを示すこともできます。これを行うには、DISPLAY ステートメントから `Total` を除去し、DECLARATIVES SECTION の USE FOR DEBUGGING を次のように変更します。

```
USE FOR DEBUGGING ON ALL PROCEDURES.
```

これで、最外部プログラムのそれぞれの非デバッグ・プロシージャーが実行される前にメッセージが表示されるようになります。

コンパイラー・オプションを使用したデバッグ

ある種のコンパイラー・オプションは、プログラム内のエラーの検出、プログラム内の各種エレメントの検出、リストの取得、およびデバッグのためのプログラムの準備に役立ちます。

コンパイラー・オプション (括弧内に示されているもの) を使用して、以下のエラーを検出することができます。

- 重複データ名のような構文エラー (NOCOMPILE)
- セクションの欠落 (SEQUENCE)
- 無効な添え字値 (SSRANGE)

コンパイラー・オプションを使用して、プログラム内にある以下のエレメントを検出することができます。

- エラー・メッセージおよび関連するエラーの発生場所 (FLAG)
- プログラム・エンティティ定義および参照。 (XREF)
- DATA DIVISION 内のデータ項目 (MAP)
- 動詞参照 (VBREF)

ソースのコピー (SOURCE) または生成されたコードのリスト (LIST) を取得できません。

TEST コンパイラー・オプションを使用して、デバッグできるようプログラムを準備します。

関連タスク

『コーディング・エラーの検出』

337 ページの『行シーケンス問題の検出』

337 ページの『有効範囲の検査』

338 ページの『診断するエラーのレベルの選択』

340 ページの『プログラム・エンティティ定義および参照の検出』

341 ページの『データ項目のリスト』

342 ページの『リストの入手』

関連参照

263 ページの『第 14 章 コンパイラー・オプション』

コーディング・エラーの検出

条件付きでコンパイルしたり、構文検査のみを行ったりする場合は、NOCOMPILE オプションを使用してください。SOURCE オプションと一緒に使用すると、NOCOMPILE は、コーディングの間違い (欠落している定義、正しく定義されていないデータ項目、重複するデータ名など) を見つけるのに役立つリストを作成します。

構文のみの検査: プログラムの構文検査のみを行い、オブジェクト・コードを生成しないようにするには、サブオプションなしの NOCOMPILE を使用してください。一緒に SOURCE オプションを指定すると、コンパイラーはリストを作成します。

NOCOMPILE を指定すると、いくつかのコンパイラー・オプションが抑制されます。詳細については、COMPILE オプションに関する下記の関連参照を参照してください。

条件付きコンパイル: 条件付きでコンパイルするには、NOCOMPILE(x) (ここで、 x はエラーの重大度レベルの 1 つです) を使用してください。エラーすべてが x より低い重大度である場合に、プログラムはコンパイルされます。使用できる重大度レベルは、S (重大)、E (エラー)、および W (警告) であり、この順序で低くなります。

レベル x またはそれ以上のエラーが発生した場合、コンパイルは停止し、プログラムの構文検査のみが行われます。

関連参照

276 ページの『COMPILE』

行シーケンス問題の検出

順序どおりになっていないステートメントを見つけるには、SEQUENCE コンパイラー・オプションを使用してください。シーケンス中断は、ソース・プログラムのセクションが移動または削除されたことを表します。

SEQUENCE を使用すると、コンパイラーはソース・ステートメント番号を検査し、昇順になっているかどうかを調べます。順序どおりになっていないステートメント番号の横には、2 つのアスタリスクが入れます。これらのステートメントの合計数はソース・リストに続く診断の最初の行として印刷されます。

関連参照

303 ページの『SEQUENCE』

有効範囲の検査

SSRANGE コンパイラー・オプションを使用して、アドレスが適切な範囲内にあるかどうかを調べます。

SSRANGE を使用すると、以下のアドレスが検査されます。

- 添え字付きまたは指標付きデータ参照: 所要のエレメントの有効アドレスが指定されたテーブルの最大境界内にあるかどうか。
- 可変長データ参照 (OCCURS DEPENDING ON 節を含むデータ項目への参照): 実際の長さが正であるかどうか、そしてグループ・データ項目に対して定義された最大長より短いかどうか。
- 参照変更データ参照: オフセットと長さが正であるかどうか。オフセットと長さの合計がデータ項目の最大長より短いかどうか。

SSRANGE オプションが有効である場合、以下の両方の条件が真であれば、実行時に検査が行われることになります。

- 指標付き、添え字付き、可変長、または参照変更データ項目が含まれている COBOL ステートメントが実行される。
- CHECK ランタイム・オプションが ON である。

参照されたデータが入っているデータ項目の範囲外にあるアドレスが生成された場合は、エラー・メッセージが生成され、プログラムは実行を停止します。メッセージでは、参照されたテーブルまたは ID、およびエラーが発生した行番号が識別されます。エラーの原因となった参照のタイプによっては、追加情報が提供されます。

与えられたデータ参照内のすべての添え字、指標、または参照修飾子がリテラルであり、データ項目の外側を参照する結果になる場合は、SSRANGE コンパイラー・オプションの設定値に関係なく、コンパイル時にエラーが診断されます。

パフォーマンスの考慮: SSRANGE が指定されると、パフォーマンスは少し低下することがあります。これは、それぞれの添え字付きまたは指標付き項目を検査するために必要なオーバーヘッドが余分にかかるためです。

関連参照

309 ページの『SSRANGE』

597 ページの『パフォーマンスに関連するコンパイラー・オプション』

診断するエラーのレベルの選択

FLAG コンパイラー・オプションを使用すると、コンパイル時に診断するエラーのレベルを指定すること、およびエラー・メッセージをリストに組み込みかどうかを指示することができます。すべてのエラーが通知されるようにするには、FLAG(I) または FLAG(I,I) を使用してください。

最初のパラメーターには、発行される構文エラー・メッセージのうち最も重大度レベルの低いものを指定してください。オプションとして、2 番目のパラメーターには、ソース・リストに組み込む構文メッセージのうち最も重大度レベルの低いものを指定します。この重大度レベルは、最初のパラメーターのレベルと同じかそれ以上でなければなりません。両方のパラメーターを指定する場合は、一緒に SOURCE コンパイラー・オプションも指定する必要があります。

表 39. コンパイラー・メッセージの重大度レベル

重大度レベル	結果のメッセージ
U (回復不能)	U (メッセージのみ)
S (重大)	すべての S および U メッセージ
E (エラー)	すべての E、S、および U メッセージ
W (警告)	すべての W、E、S、および U メッセージ
I (通知)	すべてのメッセージ

2 番目のパラメーターを指定すると、コンパイラーがエラーを検出するために利用できる情報が十分ある時点で、構文エラー・メッセージ (U レベルのメッセージを除く) がソース・リストに組み込まれます。ライブラリー・コンパイラー・フェーズで出されるメッセージ以外のすべての組み込みメッセージは、それらが参照するステートメントの直後に続きます。エラーがあるステートメントの番号もメッセージに示されます。組み込みメッセージは、ソース・リストの終わりに出される残りの診断メッセージで繰り返されます。

NOSOURCE コンパイラー・オプションを指定した場合は、構文エラー・メッセージはリストの終わりにだけ入れられます。回復不能エラーに関するメッセージはソース・リストに組み込まれません。この重大度のエラーはコンパイルを終了させるからです。

『例: 組み込みメッセージ』

関連タスク

243 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照

242 ページの『コンパイル・エラー・メッセージの重大度コード』

243 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

286 ページの『FLAG』

例: 組み込みメッセージ

次の例は、FLAG オプションに 2 番目のパラメーターを指定することによって生成される組み込みメッセージを示しています。要約の中のメッセージのいくつかは複数の COBOL ステートメントに適用されます。

```

LineID  PL SL  ----+*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7---+--8 Map and Cross Reference
...
000977      /
000978      *****
000979      ***      I N I T I A L I Z E      P A R A G R A P H      **
000980      ***      Open files. Accept date, time and format header lines.      **
000981      IA4690***      Load location-table.      **
000982      *****
000983      100-initialize-paragraph.
000984      move spaces to ws-transaction-record      IMP 339
000985      move spaces to ws-commuter-record      IMP 315
000986      move zeroes to commuter-zipcode      IMP 326
000987      move zeroes to commuter-home-phone      IMP 327
000988      move zeroes to commuter-work-phone      IMP 328
000989      move zeroes to commuter-update-date      IMP 332
000990      open input update-transaction-file      203
==000990==> IGYPS2052-S An error was found in the definition of file "LOCATION-FILE". The
reference to this file was discarded.
000991      location-file      192
000992      i-o commuter-file      180
000993      output print-file      216
000994      if loccode-file-status not = "00" or      248
000995      update-file-status not = "00" or      247
000996      updprint-file-status not = "00"      249
000997      1      display "Open Error ..."
000998      1      display " Location File Status = " loccode-file-status      248
000999      1      display " Update File Status = " update-file-status      247
001000      1      display " Print File Status = " updprint-file-status      249
001001      1      perform 900-abnormal-termination      1433
001002      end-if
001003      IA4760      if commuter-file-status not = "00" and not = "97"      240
001004      1      display "100-OPEN"
001005      1      move 100 to comp-code      230
001006      1      perform 500-vsam-error      1387
001007      1      display "Commuter File Status (OPEN) = "
001008      1      commuter-file-status      240
001009      1      perform 900-abnormal-termination      1433
001010      IA4790      end-if
001011      accept ws-date from date      UND
==001011==> IGYPS2121-S "WS-DATE" was not defined as a data-name. The statement was discarded.
001012      IA4810      move corr ws-date to header-date      UND 463
==001012==> IGYPS2121-S "WS-DATE" was not defined as a data-name. The statement was discarded.
001013      accept ws-time from time      UND
==001013==> IGYPS2121-S "WS-TIME" was not defined as a data-name. The statement was discarded.
001014      IA4830      move corr ws-time to header-time      UND 457
==001014==> IGYPS2121-S "WS-TIME" was not defined as a data-name. The statement was discarded.
001015      IA4840      read location-file      192
...

```

```

LineID  Message code  Message text
192  IGYDS1050-E  File "LOCATION-FILE" contained no data record descriptions.
      The file definition was discarded.
899  IGYPS2052-S  An error was found in the definition of file "LOCATION-FILE".
      The reference to this file was discarded.
      Same message on line: 990
1011 IGYPS2121-S  "WS-DATE" was not defined as a data-name. The statement was discarded.
      Same message on line: 1012
1013 IGYPS2121-S  "WS-TIME" was not defined as a data-name. The statement was discarded.
      Same message on line: 1014
1015 IGYPS2053-S  An error was found in the definition of file "LOCATION-FILE".
      This input/output statement was discarded.
      Same message on line: 1027
1026 IGYPS2121-S  "LOC-CODE" was not defined as a data-name. The statement was discarded.
1209 IGYPS2121-S  "COMMUTER-SHIFT" was not defined as a data-name. The statement was discarded.
      Same message on line: 1230
1210 IGYPS2121-S  "COMMUTER-HOME-CODE" was not defined as a data-name. The statement was discarded.
      Same message on line: 1231
1212 IGYPS2121-S  "COMMUTER-NAME" was not defined as a data-name. The statement was discarded.
      Same message on line: 1233
1213 IGYPS2121-S  "COMMUTER-INITIALS" was not defined as a data-name. The statement was discarded.
      Same message on line: 1234
1223 IGYPS2121-S  "WS-NUMERIC-DATE" was not defined as a data-name. The statement was discarded.
Messages Total Informational Warning Error Severe Terminating
Printed: 19
* Statistics for COBOL program FLAGOUT:
* Source records = 1755
* Data Division statements = 279
* Procedure Division statements = 479
Locale = en_US.ISO8859-1 (1)
End of compilation 1, program FLAGOUT, highest severity: Severe.
Return code 12

```

(1) コンパイラーが使用したロケール

プログラム・エンティティー定義および参照の検出

XREF(FULL) コンパイラー・オプションを使用すると、データ名、プロシージャ名、またはプログラム名が定義および参照されている場所を見つけることができます。また、コピーブックを取得したファイルへの、COPY または BASIS ステートメントの相互参照を作成するためにも使用します。

ソート済み相互参照には、そのデータ名、プロシージャ名、またはプログラム名が定義されている行番号およびそのデータ名、プロシージャ名、またはプログラム名へのすべての参照の行番号が入られます。

明示的に参照されているデータ項目だけを含める場合は、XREF(SHORT) オプションを使用します。

XREF (FULL または SHORT) と SOURCE オプションの両方を使用すると、変更された相互参照がソース・リストの右側に印刷されます。この組み込み相互参照は、データ名またはプロシージャ名が定義されている行番号を示します。

詳細については、XREF コンパイラー・オプションに関する下の関連参照を参照してください。

- 349 ページの『例: XREF 出力: データ名相互参照』
- 350 ページの『例: XREF 出力: プログラム名相互参照』
- 350 ページの『例: XREF 出力: COPY/BASIS 相互参照』
- 351 ページの『例: 組み込み相互参照』

関連タスク

- 342 ページの『リストの入手』

関連参照

316 ページの『XREF』

データ項目のリスト

MAP コンパイラー・オプションを使用すると、DATA DIVISION の項目と暗黙的に宣言されたすべての項目のリストを作成することができます。

MAP オプションを使用すると、圧縮された MAP 情報を含んでいる組み込み MAP 要約が、COBOL ソース・データ宣言の右側に生成されます。XREF データと組み込み MAP 要約の両方が同じ行にあるときは、組み込み要約の方が先に印刷されます。

MAP リストおよび組み込み MAP 要約の各部分は、ソース全体を通じ、*CONTROL MAPINOMAP (または *CBL MAPINOMAP) ステートメントを使用して、選択または抑制することができます。以下に例を示します。

```
*CONTROL NOMAP
  01 A
  02 B
*CONTROL MAP
```

346 ページの『例: MAP 出力』

関連タスク

342 ページの『リストの入手』

関連参照

293 ページの『MAP』

デバッガーの使用

COBOL for AIX はデバッガーを備えています。TEST コンパイラー・オプションを使用すれば、デバッガーによって実行可能プログラムをステップスルーできるように、COBOL プログラムを準備することができます。

あるいは、cob2 コマンド (またはそのバリエーションのいずれか) の -g オプションを使用して、デバッガーを使用するためにプログラムを準備することもできます。

デバッガーを呼び出して実行可能ファイルを検査するには、コマンド `idebug filename` を発行します (ここで、*filename* はプログラムの名前です)。プログラムが -g オプションでコンパイルされた場合は、デバッガーからシンボリックおよび行番号情報が提供されます。

関連タスク

236 ページの『コマンド行からのコンパイル』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

310 ページの『TEST』

リストの入手

コンパイラー・オプションを使用して適切なコンパイラー・リストを要求することによって、デバッグに必要な情報を入手してください。

重要: コンパイラーによって作成されるリストは、プログラミング・インターフェースではなく、容易に変更できるものです。

表 40. コンパイラー・オプションとリストの対応

用途	リスト	内容	コンパイラー・オプション
<p>プログラムに有効なオプションのリスト、プログラムの内容に関する統計、およびコンパイルに関する診断メッセージを検査する。</p> <p>コンパイル時に、有効なロケールを検査する。</p>	短縮リスト	<ul style="list-style-type: none"> プログラムに有効なオプションのリスト プログラムの内容に関する統計 コンパイルに関する診断メッセージ¹ <p>有効なロケールを示すロケール行</p>	NOSOURCE、NOXREF、NOVBREF、NOMAP、NOLIST
プログラムのテストおよびデバッグを援助する。プログラムのデバッグ後にレコードを得る。	ソース・リスト	ソースのコピー	306 ページの『SOURCE』
特定のデータ項目を見つける。再入可能性または最適化を考慮した後の最終ストレージ割り振りを調べる。プログラムが定義されている場所を見つけ、その属性を検査する。	DATA DIVISION 項目のマッピング	<p>すべての DATA DIVISION 項目および暗黙的に宣言されたすべての項目</p> <p>組み込みマップ要約 (DATA DIVISION 内の、データ宣言が含まれている行のリストの右マージン)</p> <p>ネストされたプログラム・マップ (ネストされたプログラムが含まれているプログラム)</p>	293 ページの『MAP』 ²
名前が定義、参照、または変更されている場所を調べる。プロシージャーが参照されているコンテキスト (例えば、動詞が PERFORM ブロックで使用されたかどうか) を判別する。コピーブックの取得元のファイルを判別する。	名前がソートされた相互参照リスト。COPY/BASIS ステートメントおよびコピーブック・ファイルのソートされた相互参照リスト	<p>データ名、プロシージャー名、プログラム名。これらの名前への参照。</p> <p>COPY/BASIS テキスト名とライブラリー名、および関連コピーブックを取得したファイル</p> <p>埋め込まれた変更済み相互参照は、データ名およびプロシージャー名が定義された行番号を提供します</p>	316 ページの『XREF』 ^{2,3}

表 40. コンパイラー・オプションとリストの対応 (続き)

用途	リスト	内容	コンパイラー・オプション
プログラム内の障害のある動詞を見つける。または、プログラムの実行時に移動されたデータ項目のストレージのアドレスを調べる。	コンパイラーによって生成される PROCEDURE DIVISION コードおよびアセンブラ・コード ³	生成されたコード	291 ページの『LIST』 ^{2,4}
特定の動詞のインスタンスを見つける。	アルファベット順の動詞	使用されたそれぞれの動詞、各動詞が使用された回数、各動詞が使用された行番号	315 ページの『VBREF』

1. メッセージを除去するには、コンパイル診断情報のレベルを左右するオプション (例えば、FLAG) をオフにしてください。

2. コンパイル済みプログラムの行番号を使用するには、NUMBER コンパイラー・オプションを使用してください。コンパイラーは、ステートメントが読み込まれるときに、桁 1 から 6 にあるソース・ステートメント行番号のシーケンスを検査します。行番号が順序どおりになっていないことがわかると、コンパイラーは先行のステートメントの行番号より 1 だけ大きい値の番号を割り当てます。新しい値には、2 つのアスタリスクのフラグが付けられます。シーケンス・エラーを示す診断メッセージがコンパイル・リストに入れられます。

3. プロシーチャー参照のコンテキストは、行番号の前の文字で示されます。

4. アセンブラー・リストはリスト・ファイル (ソース・プログラムと同じ名前であるがサフィックスが .wlist であるファイル) に書き込まれます。

『例: 短縮リスト』

345 ページの『例: SOURCE および NUMBER 出力』

346 ページの『例: MAP 出力』

346 ページの『例: 組み込みマップ要約』

348 ページの『例: ネストされたプログラム・マップ』

349 ページの『例: XREF 出力: データ名相互参照』

350 ページの『例: XREF 出力: プログラム名相互参照』

350 ページの『例: XREF 出力: COPY/BASIS 相互参照』

351 ページの『例: 組み込み相互参照』

352 ページの『例: VBREF コンパイラー出力』

関連タスク

243 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照

243 ページの『コンパイラー検出エラーに関するメッセージおよびリスト』

例: 短縮リスト

下記リストに示された括弧付きの番号は、リストに続く説明の番号と対応しています。診断メッセージの原因となったエラーのいくつかは、説明を行うために故意に挿入されたものです。

```
PROCESS(CBL) statements: (1)
CBL NOSOURCE,NOXREF,NOVBREF,NOMAP,NOLIST (2)
Options in effect: (3)
NOADATA
  ADDR(32)
  QUOTE
```

```

    ARITH(COMPAT)
    CALLINT(NODESCRIPTOR)
    CHAR(NATIVE)
NOCICS
    COLLSEQ(BINARY)
NOCOMPILE(S)
NOCURRENCY
NODATEPROC
NODIAGTRUNC
NODYNAM
NOEXIT
    FLAG(I,I)
NOFLAGSTD
    FLOAT(NATIVE)
NOLIB
    LINECOUNT(60)
NOLIST
    LSTFILE(LOCALE)
NOMAP
    MAXMEM(2048K)

NOMDECK
    NCOLLSEQ(BINARY)
    NSYMBOL(NATIONAL)
NONUMBER
NOOPTIMIZE
    PGMNAME(LONGUPPER)
NOPROFILE

    SEPOBJ
    SEQUENCE
    SIZE(8388608)
NOSOSI
NOSOURCE
    SPACE(1)
    SPILL(512)
NOSQL
NOSSRANGE
    TERM
NOTEST
NOTHREAD
    TRUNC(STD)
NOVBREF
NOWSCLEAR
NOXREF
    YEARWINDOW(1900)
ZWB

LineID Message code Message text (4)
IGYDS0139-W Diagnostic messages were issued during processing of compiler options. These messages are
located at the beginning of the listing.
193 IGYDS1050-E File "LOCATION-FILE" contained no data record descriptions. The file definition was discarded.
889 IGYPS2052-S An error was found in the definition of file "LOCATION-FILE". The reference to this file
was discarded.
Same message on line: 983
993 IGYPS2121-S "WS-DATE" was not defined as a data-name. The statement was discarded.
Same message on line: 994
995 IGYPS2121-S "WS-TIME" was not defined as a data-name. The statement was discarded.
Same message on line: 996
997 IGYPS2053-S An error was found in the definition of file "LOCATION-FILE". This input/output statement
was discarded.
Same message on line: 1009
1008 IGYPS2121-S "LOC-CODE" was not defined as a data-name. The statement was discarded.
1219 IGYPS2121-S "COMMUTER-SHIFT" was not defined as a data-name. The statement was discarded.
Same message on line: 1240
1220 IGYPS2121-S "COMMUTER-HOME-CODE" was not defined as a data-name. The statement was discarded.
Same message on line: 1241
1222 IGYPS2121-S "COMMUTER-NAME" was not defined as a data-name. The statement was discarded.
Same message on line: 1243
1223 IGYPS2121-S "COMMUTER-INITIALS" was not defined as a data-name. The statement was discarded.
Same message on line: 1244
1233 IGYPS2121-S "WS-NUMERIC-DATE" was not defined as a data-name. The statement was discarded.
Messages Total Informational Warning Error Severe Terminating (5)
Printed: 21 2 1 18
* Statistics for COBOL program SLISTING: (6)
* Source records = 1765
* Data Division statements = 277
* Procedure Division statements = 513
Locale = en_US.ISO8859-1 (7)
End of compilation 1, program SLISTING, highest severity: Severe. (8)
Return code 12

```

- (1) PROCESS (または CBL) ステートメントで指定されたオプションに関するメッセージ。このメッセージは、オプションが指定されていない場合には表示されません。

- (2) PROCESS (または CBL) ステートメントの中にコーディングされたオプション。
- (3) このコンパイルの開始時のオプションの状況。
- (4) プログラム診断メッセージ。最初のメッセージは、ライブラリー・フェーズ診断 (ある場合) に言及しています。ライブラリー・フェーズの診断は、常時リストの先頭に示されます。
- (5) このプログラムの診断メッセージのカウントで、重大度レベルによってグループ化されたもの。
- (6) プログラム SLISTING のプログラム統計。
- (7) コンパイラーが使用したロケール。
- (8) コンパイル単位のプログラム統計。バッチ・コンパイルを実行する (1 回のコンパイルで複数の最外部 COBOL プログラムを実行する) 場合、戻りコードは、コンパイル全体について最高レベルのメッセージ重大度です。

例: SOURCE および NUMBER 出力

次に示されているリストの部分では、プログラマーは 2 つのステートメントに順序どおりでない番号を付けています。リストに示された注釈番号は、後続の説明の番号と対応しています。

```

(1)
LineID  PL SL  ----+*A-1-B-+----2-+----3-+----4-+----5-+----6-+----7-|---+----8  Cross-Reference
(2)    (3)  (4)
087000/*****
087100***          D O  M A I N  L O G I C          **
087200***          **
087300*** Initialization. Read and process update transactions until **
087400*** EOE. Close files and stop run.          **
087500*****
087600 procedure division.
087700    000-do-main-logic.
087800        display "PROGRAM SRCOUT - Beginning"
087900        perform 050-create-vsam-master-file.
088150        display "perform 050-create-vsam-master finished".
088151** 088125        perform 100-initialize-paragraph
088200        display "perform 100-initialize-paragraph finished"
088300        read update-transaction-file into ws-transaction-record
088400            at end
1 088500            set transaction-eof to true
088600        end-read
088700        display "READ completed"
088800        perform until transaction-eof
1 088900            display "inside perform until loop"
1 089000            perform 200-edit-update-transaction
1 089100            display "After perform 200-edit  "
1 089200            if no-errors
2 089300                perform 300-update-commuter-record
2 089400                display "After perform 300-update "
1 089650            else
089651** 2 089600                perform 400-print-transaction-errors
2 089700                display "After perform 400-errors "
1 089800            end-if
1 089900            perform 410-re-initialize-fields
1 090000            display "After perform 410-reinitialize"
1 090100            read update-transaction-file into ws-transaction-record
1 090200                at end
2 090300                set transaction-eof to true
1 090400            end-read
1 090500            display "After '2nd READ'  "
090600        end-perform

```

- (1) スケール行では、区域 A、区域 B、およびソース・コード桁番号にラベルを付けます。
- (2) コンパイラーが割り当てるソース・コード行番号。
- (3) プログラム (PL) とステートメント (SL) のネスト・レベル。

(4) プログラムの第 1 から 6 桁 (シーケンス番号域)。

例: MAP 出力

次の例は、MAP オプションからの出力を示しています。その下の説明で使用されている番号は、出力に付けられている番号と対応しています。

```
Data Division Map
(1)
Data Definition Attribute codes (rightmost column) have the following meanings:
  D = Object of OCCURS DEPENDING   G = GLOBAL                       LSEQ= ORGANIZATION LINE SEQUENTIAL
  E = EXTERNAL                     O = Has OCCURS clause           SEQ= ORGANIZATION SEQUENTIAL
  VLO=Variably Located Origin       OG= Group has own length definition  INDX= ORGANIZATION INDEXED
  VL= Variably Located              R = REDEFINES                   REL= ORGANIZATION RELATIVE

(2)   (3) (4)                               (5)   (6)   (7)   (8)
Source Hierarchy and                          Length(Displacement) Data Type   Data Def
LineID Data Name
-----*
  4 PROGRAM-ID IGYTCARA-----*
180 FD COMMUTER-FILE                          File                               INDX
182 1 COMMUTER-RECORD                          80                               Group
183 2 COMMUTER-KEY                            16(0000000)                      Display
184 2 FILLER                                  64(0000016)                      Display
186 FD COMMUTER-FILE-MST                      File                               INDX
188 1 COMMUTER-RECORD-MST                    80                               Group
189 2 COMMUTER-KEY-MST                      16(0000000)                      Display
190 2 FILLER                                  64(0000016)                      Display
192 FD LOCATION-FILE                          File                               SEQ
203 FD UPDATE-TRANSACTION-FILE              File                               SEQ
208 1 UPDATE-TRANSACTION-RECORD             80                               Display
216 FD PRINT-FILE                            File                               SEQ
221 1 PRINT-RECORD                          121                              Display
228 1 WORKING-STORAGE-FOR-IGYTCARA           1                               Display
```

- (1) データ定義属性コードの説明。
- (2) データ項目が定義されたソース行番号。
- (3) レベル定義または番号。コンパイラーは、次の方法でこの番号を生成します。
 - 階層の第 1 レベルは常に 01 です。レベル 02 から 49 としてコーディングした項目のレベルごとに 1 を加えます。
 - レベル番号の 66、77、および 88、そして標識 FD と SD は変更されません。
- (4) ソース・モジュールでソース順序で使用されるデータ名。
- (5) データ項目の長さ。ベース・ロケーター値。
- (6) 収容構造の先頭からの 16 進変位。
- (7) データ型および使用法。
- (8) データ定義属性コード。定義は DATA DIVISION マップの先頭で説明されています。

『例: 組み込みマップ要約』

348 ページの『例: ネストされたプログラム・マップ』

関連参照

347 ページの『MAP 出力で使用される用語および記号』

例: 組み込みマップ要約

次の例は、MAP オプションによって作成される組み込みマップ要約を示しています。この要約は、DATA DIVISION の、データ宣言を含む行のリストの右マージンに現れます。

```

000002          Identification Division.
000003
000004          Program-id.  EMBMAP.

000176          Data division.
000177          File section.
000178
000179
000180          FD  COMMUTER-FILE
000181             record 80 characters.                (1) (2)
000182             01 commuter-record.                  80
000183             05 commuter-key                      PIC x(16).    16(0000000)
000184             05 filler                          PIC x(64).    64(0000016)  . . .
000221          IA1620 01 print-record                pic x(121).   121

000227          Working-storage section.
000228             01 Working-storage-for-EMBMAP        pic x.        1
000229
000230             77 comp-code                          pic S9999 comp. 2
000231             77 ws-type                          pic x(3) value spaces. 3
000232
000233
000234             01 i-f-status-area.
000235             05 i-f-file-status                    pic x(2).    2
000236             88 i-o-successful                    value zeroes. IMP 2(0000000)
000237
000238
000239             01 status-area.
000240             05 commuter-file-status                pic x(2).    (3) 2(0000000)
000241             88 i-o-okay                          value zeroes. IMP
000242             05 commuter-vsam-status.
000243             10 vsam-r15-return-code              pic 9(2) comp. 6(0000002) 2(0000002)
000244             10 vsam-function-code                pic 9(1) comp. 2(0000004)
000245             10 vsam-feedback-code              pic 9(3) comp. 2(0000006)
000246
000247             77 update-file-status                  pic xx.      2
000248             77 loccode-file-status                pic xx.      2
000249             77 updprint-file-status              pic xx.      2

000877          procedure division.
000878             000-do-main-logic.
000879             display "PROGRAM EMBMAP - Beginning".
000880             perform 050-create-vsam-master-file.  931

```

- (1) データ項目の長さ (10 進数)
 - (2) ベース・ロケータ値の先頭からの 16 進変位。
 - (3) 特殊定義記号:
 - UND** ユーザー名が未定義です。
 - DUP** ユーザー名が 1 回を超えて定義されています。
 - IMP** 暗黙的に定義された名前 (特殊レジスターや形象定数など)。
 - IFN** 組み込み関数参照。
 - EXT** 外部参照。
- * NOCOMPILE オプションが有効なため、プログラム名が未解決です。

MAP 出力で使用される用語および記号

次の表は、MAP コンパイラー・オプションによって作成されるリストで使用される用語および記号を説明しています。

表 41. MAP 出力で使用される用語および記号

用語	説明
ALPHABETIC	英字 (PICTURE A)
ALPHA-EDIT	英字編集
AN-EDIT	英数字編集
BINARY	バイナリー (USAGE BINARY、COMPUTATIONAL、または COMPUTATIONAL-5)
COMP-1	単精度内部浮動小数点 (USAGE COMPUTATIONAL-1)
COMP-2	倍精度内部浮動小数点 (USAGE COMPUTATIONAL-2)

表 41. MAP 出力で使用される用語および記号 (続き)

用語	説明
DBCS	DBCS (USAGE DISPLAY-1)
DBCS-EDIT	DBCS 編集
DISP-FLOAT	表示浮動小数点 (USAGE DISPLAY)
DISPLAY	英数字 (PICTURE X)
DISP-NUM	ゾーン 10 進数 (USAGE DISPLAY)
DISP-NUM-EDIT	数字編集 (USAGE DISPLAY)
FD	ファイル定義
FUNCTION-PTR	外部呼び出し可能関数を指すポインター (USAGE FUNCTION-POINTER)
GROUP	英数字固定長グループ
GRP-VARLEN	英数字可変長グループ
INDEX	指標 (USAGE INDEX)
INDEX-NAME	指標名
NATIONAL	カテゴリー国別 (USAGE NATIONAL)
NAT-EDIT	国別編集 (USAGE NATIONAL)
NAT-FLOAT	国別浮動小数点 (USAGE NATIONAL)
NAT-GROUP	国別グループ (GROUP-USAGE NATIONAL)
NAT-GRP-VARLEN	国別可変長グループ (GROUP-USAGE NATIONAL)
NAT-NUM	国別 10 進数 (USAGE NATIONAL)
NAT-NUM-EDIT	国別数字編集 (USAGE NATIONAL)
OBJECT-REF	オブジェクト参照 (USAGE OBJECT REFERENCE)
PACKED-DEC	内部 10 進数 (USAGE PACKED-DECIMAL または COMPUTATIONAL-3)
POINTER	ポインター (USAGE POINTER)
PROCEDURE-PTR	外部呼び出し可能プログラムを指すポインター (USAGE PROCEDURE-POINTER)
SD	ソート・ファイル定義
01 から 49、77	データ記述に関するレベル番号
66	RENAMES に関するレベル番号
88	条件名に関するレベル番号

例: ネストされたプログラム・マップ

この例は、MAP コンパイラー・オプションを指定することによって作成される、ネストされたプロシーチャーのマップを示しています。括弧内の番号は、後続の注釈に対応しています。

Nested Program Map

(1)

Program Attribute codes (rightmost column) have the following meanings:

C = COMMON

I = INITIAL

U = PROCEDURE DIVISION USING...

(2)	(3)	(4)	(5)
Source LineID	Nesting Level	Program Name from PROGRAM-ID paragraph	Program Attributes
2		NESTED.	
12	1	X1.	
20	2	X11	
27	2	X12	
35	1	X2.	

- (1) プログラム属性コードの説明
- (2) プログラムが定義されたソース行番号
- (3) プログラムのネストの深さ
- (4) プログラム名
- (5) プログラム属性コード

例: XREF 出力: データ名相互参照

次の例は、XREF コンパイラー・オプションによって作成される、データ名のソート済み相互参照を示しています。括弧内の番号は、後続の注釈に対応しています。

An "M" preceding a data-name reference indicates that the data-name is modified by this reference.

(1)	(2)	(3)
Defined	Cross-reference of data-names	References
264	ABEND-ITEM1	
265	ABEND-ITEM2	
347	ADD-CODE	1126 1192
381	ADDRESS-ERROR.	M1156
280	AREA-CODE.	1266 1291 1354 1375
382	CITY-ERROR	M1159

(4)
Context usage is indicated by the letter preceding a procedure-name reference. These letters and their meanings are:

- A = ALTER (procedure-name)
- D = GO TO (procedure-name) DEPENDING ON
- E = End of range of (PERFORM) through (procedure-name)
- G = GO TO (procedure-name)
- P = PERFORM (procedure-name)
- T = (ALTER) TO PROCEED TO (procedure-name)
- U = USE FOR DEBUGGING (procedure-name)

(5)	(6)	(7)
Defined	Cross-reference of procedures	References
877	000-DO-MAIN-LOGIC	
943	050-CREATE-STL-MASTER-FILE . . .	P879
995	100-INITIALIZE-PARAGRAPH . . .	P881
1471	1100-PRINT-I-F-HEADINGS. . . .	P926
1511	1200-PRINT-I-F-DATA.	P928
1573	1210-GET-MILES-TIME.	P1540
1666	1220-STORE-MILES-TIME.	P1541
1682	1230-PRINT-SUB-I-F-DATA. . . .	P1562
1706	1240-COMPUTE-SUMMARY	P1563
1052	200-EDIT-UPDATE-TRANSACTION. .	P890
1154	210-EDIT-THE-REST.	P1145
1189	300-UPDATE-COMMUTER-RECORD . .	P893
1237	310-FORMAT-COMMUTER-RECORD . .	P1194 P1209

```

1258 320-PRINT-COMMUTER-RECORD. . . P1195 P1206 P1212 P1222
1318 330-PRINT-REPORT . . . . . P1208 P1232 P1286 P1310 P1370
1342 400-PRINT-TRANSACTION-ERRORS . P896

```

データ名の相互参照:

- (1) その名前が定義されている行番号。
- (2) データ名。
- (3) その名前が使用されている行番号。M が行番号の前に置かれている場合は、データ項目がその位置で明示的に変更されたことを意味します。

プロシージャ参照の相互参照:

- (4) プロシージャ参照のコンテキスト取扱コードの説明。
- (5) そのプロシージャ名が定義されている行番号。
- (6) プロシージャ名。
- (7) そのプロシージャが参照されている行番号およびそのプロシージャのコンテキスト取扱コード。

『例: XREF 出力: プログラム名相互参照』

『例: XREF 出力: COPY/BASIS 相互参照』

351 ページの『例: 組み込み相互参照』

例: XREF 出力: プログラム名相互参照

次の例は、XREF コンパイラー・オプションによって作成される、プログラム名のソート済み相互参照を示しています。括弧内の番号は、後続の注釈に対応しています。

(1) Defined	(2) Cross-reference of programs	(3) References
EXTERNAL	EXTERNAL1.	25
2	X.	41
12	X1.	33 7
20	X11.	25 16
27	X12.	32 17
35	X2.	40 8

- (1) そのプログラム名が定義されている行番号。プログラムが外部の場合は、定義行番号の代わりに EXTERNAL という語が表示されます。
- (2) プログラム名。
- (3) そのプログラムが参照されている行番号。

例: XREF 出力: COPY/BASIS 相互参照

次の例は、の XREF コンパイラー・オプションで生成された関連コピーブックのライブラリー名およびファイル名に対するコピーブックのソート済み相互参照を示しています。括弧内の番号は、後続の注釈に対応しています。

COPY/BASIS cross-reference of text-names, library names and file names

(1) Text-name	(1) Library-name	(2) File name
	(3)	(3) (4)

```

"realxrealzlongxlo> 'thislongdirecto> <toryname/realxrealzlongxname.cpy
"realxrealzlongxlo> SYSLIB (default) (5) ./cbldir1/realxrealzlongxname.cpy
"copyA.cpy" SYSLIB (default) ./cbldir1/copyA.cpy
'./copydir2/copyM.cbl' SYSLIB (default) ./copydir2/copyM.cbl
'/copyB.cpy' SYSLIB ./cbldir1/copyB.cpy
'/copydir/copyM.cbl' SYSLIB ./cbldir1/copydir/copyM.cbl
'cbldir1/copyC.cpy' ALTDD2 ./cbldir1/copyC.cpy
'copydir/copyM.cbl' SYSLIB ./cbldir1/copydir/copyM.cbl
'copydir2/copyM.cbl' SYSLIB (default) ./copydir2/copyM.cbl
'copydir3/stuff.cpy' ALTDD2 ./copydir3/stuff.cpy
'stuff.cpy' ALTDD ./copydir3/stuff.cpy
OTHERDD ALTDD2 ./cbldir1/other.cob
REALXLONGXLONGYNAMEX SYSLIB (default) ./REALXLONGXLONGYNAMEX
. . .

```

(5)

./ = /afs/stllp.sanjose.ibm.com/usr1/cobdev/tmross/stuff/subdir

Note: Some names were truncated. > = truncated on right < = truncated on left

(1) ソースの COPY ステートメントからのものです。たとえば、上の相互参照の 5 番目の項目に対応する COPY ステートメントは次のようなものです。

```
COPY '/copyB.cpy' Of SYSLIB
```

(2) COPY メンバーをコピーした元のファイルの完全修飾パス

(3) 長いテキスト名またはライブラリー名の右側の切り捨ては、大なり記号 (>) で示されます。

(4) 長いファイル名の左側の切り捨ては、小なり記号 (<) で示されます。

(5) 相互参照では、ファイル名の現行作業ディレクトリー部分は ./ で表されます。相互参照の下に、現行作業ディレクトリー名を展開した完全なディレクトリー名が示されます。

例: 組み込み相互参照

次の例は、ソース・リストに組み込まれる変更済み相互参照を示しています。この相互参照は XREF コンパイラー・オプションによって作成されます。

```

LineID  PL SL  ----+*A-1-B--+----2-----3-----4-----5-----6-----7-|--+-----8  Map and Cross Reference
. . .
000878          procedure division.
000879          000-do-main-logic.
000880          display "PROGRAM IGYTCARA - Beginning".
000881          perform 050-create-vsam-master-file.          932 (1)
000882          perform 100-initialize-paragraph.            984
000883          read update-transaction-file into ws-transaction-record 204 340
000884          at end
000885          1      set transaction-eof to true            254
000886          end-read.

. . .
000984          100-initialize-paragraph.
000985          move spaces to ws-transaction-record          IMP 340 (2)
000986          move spaces to ws-commuter-record          IMP 316
000987          move zeroes to commuter-zipcode            IMP 327
000988          move zeroes to commuter-home-phone         IMP 328
000989          move zeroes to commuter-work-phone         IMP 329
000990          move zeroes to commuter-update-date       IMP 333
000991          open input update-transaction-file        204
000992          location-file                             193
000993          i-o commuter-file                         181
000994          output print-file                         217

. . .
001442          1100-print-i-f-headings.
001443
001444          open output print-file.                    217
001445
001446          move function when-compiled to when-comp.   IFN 698 (2)
001447          move when-comp (5:2) to compile-month.    698 640
001448          move when-comp (7:2) to compile-day.     698 642
001449          move when-comp (3:2) to compile-year.     698 644
001450
001451          move function current-date (5:2) to current-month. IFN 649
001452          move function current-date (7:2) to current-day. IFN 651
001453          move function current-date (3:2) to current-year. IFN 653
001454
001455          write print-record from i-f-header-line-1    222 635
001456          after new-page.                            138
. . .

```

(1) プログラム内のデータ名またはプロシージャ名の定義の行番号。

- (2) 特殊定義記号:
- UND** ユーザー名が未定義です。
 - DUP** ユーザー名が 1 回を超えて定義されています。
 - IMP** 暗黙的に定義された名前 (特殊レジスターや形象定数など)。
 - IFN** 組み込み関数参照。
 - EXT** 外部参照。
- * NOCOMPILE オプションが有効なため、プログラム名が未解決です。

例: VBREF コンパイラー出力

次の例は、プログラム内のすべての動詞のアルファベット順のリストと、各動詞が参照されている場所を示しています。このリストは、VBREF コンパイラー・オプションによって作成されます。

(1)	(2)	(3)
2	ACCEPT	101 101
2	ADD	129 130
1	CALL	140
5	CLOSE	90 94 97 152 153
20	COMPUTE150 164 164 165 166 166 166 166 167 168 168 169 169 170 171 171
		171 172 172 173
2	CONTINUE106 107
2	DELETE96 119
47	DISPLAY88 90 91 92 92 93 94 94 94 95 96 96 97 99 99 100 100 100 100
		103 109 117 117 118 119 138 139 139 139 139 139 139 140 140 140
		140 143 148 148 149 149 149 152 152 152 153 162
2	EVALUATE116 155
47	IF88 90 93 94 94 95 96 96 97 99 100 103 105 105 107 107 107 109
		110 111 111 112 113 113 113 113 114 114 115 115 116 118 119 124
		124 126 127 129 132 133 134 135 136 148 149 152 152
183	MOVE90 93 95 98 98 98 98 98 99 100 101 101 102 104 105 105 106 106
		107 107 108 108 108 108 108 108 109 110 111 112 113 113 114
		114 114 115 115 116 116 117 117 117 118 118 118 119 119 120 121
		121 121 121 121 121 121 121 121 121 122 122 122 122 122 123 123
		123 123 123 123 123 124 124 124 125 125 125 125 125 125 126
		126 126 126 126 127 127 127 127 128 128 129 129 130 130 130
		131 131 131 131 131 132 132 132 132 132 133 133 133 133 133
		134 134 134 134 134 135 135 135 135 135 135 136 136 137 137
		137 137 138 138 138 138 141 141 142 142 144 144 144 144 145 145
		145 145 146 149 150 150 150 151 151 155 156 156 157 157 158 158
		159 159 160 160 161 161 162 162 162 168 168 168 169 169 170 171
		171 172 172 173 173
5	OPEN93 95 99 144 148
62	PERFORM88 88 88 88 89 89 89 91 91 91 91 93 93 94 94 95 95 95 95 96
		96 96 97 97 97 100 100 101 102 104 109 109 111 116 116 117 117
		117 118 118 118 118 119 119 119 120 120 124 125 127 128 133 134
		135 136 136 137 150 151 151 153 153
8	READ88 89 96 101 102 108 149 151
1	REWRITE118
4	SEARCH106 106 141 142
46	SET88 89 101 103 104 105 106 108 108 136 141 142 149 150 151 152 154
		155 156 156 156 156 157 157 157 157 158 158 158 158 159 159 159
		159 160 160 160 160 161 161 161 161 162 162 164 164
2	STOP92 143
4	STRING123 126 132 134
33	WRITE94 116 129 129 129 129 129 130 130 130 130 145 146 146 146 146 147
		147 151 165 165 166 166 167 174 174 174 174 174 174 175 175

- (1) その動詞がプログラムで使用されている回数。
- (2) 動詞。
- (3) その動詞が使用されている行番号。

オフセット情報を含むメッセージによるデバッグ

いくつかの IWZ メッセージには、失敗したプログラムの特定の行の識別に使用できるオフセット情報が含まれています。

この情報を使用するには、次のようにします。

1. LIST オプションを使用してプログラムをコンパイルします。次のステップにより、接尾部が .wlist のアセンブラー・リスト・ファイルが作成されます。
2. オフセット情報を含むメッセージを取得する場合は、COBOL プログラムに対するオフセット情報を検索します。次の例では、プログラムは IWZ903 で、対応する 16 進オフセットは 0x14c です (太字で強調表示されています)。

```
IWZ903S The system detected a data exception.  
        Message routine called from offset 0x38 of routine iwzWriteERRmsg.  
        iwzWriteERRmsg called from offset 0xa0 of routine _iwzcBCD_ADD_Pckd.  
        _iwzcBCD_ADD_Pckd called from offset 0x14c of routine IWZ903.  
IWZ901S Program exits due to severe or critical error.
```

3. .wlist ファイル内で 16 進オフセットを確認します。16 進オフセットの左側は、COBOL ステートメント番号です。次の例では、0x14c に対応するステートメント番号は 35 です (太字で強調表示されています)。

```
35 | 000144 addi    39000004  1    LI      gr8=4  
35 | 000148 b1     4BFFFEB9  0    CALL   gr3=_iwzcBCD_ADD_Pckd  
35 | 00014C ori     60000000  1  
35 | 000150 fmr.   FC000091  0
```

4. プログラム・リスト内でこのステートメントを確認します。

関連参照

291 ページの『LIST』

ユーザー出口のデバッグ

ユーザー出口ルーチンをデバッグするには、cob2 ではなく、メインのコンパイラー・モジュール上でデバッガーを使用します。(メインのコンパイラー・モジュールは cob2 によって起動される個別のプロセスです。デバッガーは、1 つのプロセスしかデバッグできません。)

以下の手順を実行します。

1. cob2 とともに -# オプションを使用して、cob2 がメイン・コンパイラー・モジュールを呼び出す方法と、渡されるオプションを調べます。例えば、次のコマンドは、IWZRMGUX ユーザー出口を使用して *pgmname.cb1* をコンパイルし、。

```
cob2 -c -q"EXIT(ADEXIT(IWZRMGUX))" pgmname.cb1
```

このコマンドを次のように修正します。

```
cob2 -# -c -q"EXIT(ADEXIT(IWZRMGUX))" pgmname.cb1
```

結果として、以下が表示されます (IGYCCOB2 はご使用のユーザー出口を呼び出します)。

```
IGYCCOB2 -q"EXIT(ADEXIT(IWZRMGUX))" pgmname.cb1
```

2. ユーザー出口を次のようにデバッグします。

```
idebug IGYCCOB2 -q"EXIT(ADEXIT(IWZRMGUX))" pgmname.cb1
```

デバッガーは、デバッグ情報を使用してユーザー出口を構築する場合に、ユーザー出口の開始時に自動的に停止します。

アセンブラー・ルーチンのデバッグ

アセンブラー・ルーチンをデバッグするには、「Disassembly (分解)」ビューを使用します。アセンブラー・ルーチンにはデバッグ情報がないため、デバッガーは自動的にこのビューに移動します。

「Disassembly (分解)」ビュー内で、逆アセンブルされたステートメントにブレークポイントを設定するには、接頭部域内をダブルクリックします。デフォルトでは、デバッガーは開始時に、最初のデバッグ可能ステートメントを検出するまで実行します。(デバッグ可能かどうかを問わず) アプリケーション内の最初の命令でデバッガーを停止させるには、`-i` オプションを使用する必要があります。以下に例を示します。

```
idebug -i progname
```

第 4 部 データベースへのアクセス

第 18 章 DB2 環境用のプログラミング	357
DB2 coprocessor	357
SQL ステートメントのコーディング	358
DB2 coprocessor を用いた SQL INCLUDE の使 用	359
SQL ステートメントでのバイナリー項目の使用	359
SQL ステートメントの成否の判断	360
コンパイル前の DB2 の起動	360
SQL オプションを使用したコンパイル	360
DB2 サブオプションの分離	361
パッケージ名およびバインド・ファイル名の使用	361
第 19 章 COBOL プログラムの開発 (CICS の場 合)	363
CICS のもとで実行する COBOL プログラムのコー ディング	364
CICS のもとでのシステム日付の取得	365
CICS での動的呼び出し	366
SFS データへのアクセス	367
CICS での COBOL および C/C++ 間の呼び出し	367
CICS プログラムのコンパイルおよび実行	368
I 組み込みの CICS 変換プログラム	369
CICS プログラムのデバッグ	369
第 20 章 Open Database Connectivity (ODBC)	371
ODBC と組み込み SQL の比較	371
バックグラウンド	372
ODBC 対応ソフトウェアのインストールおよび構成	372
COBOL からの ODBC 呼び出しのコーディング: 概要	372
ODBC に適したデータ型の使用	372
ODBC 呼び出しにおける引数としてのポインテ ーの受け渡し	373
例: ODBC 呼び出しにおける引数としてのポ インターの受け渡し	374
ODBC 呼び出しにおける関数戻り値へのアクセ ス	375
ODBC 呼び出しにおけるビットのテスト	375
ODBC API 用の COBOL コピーブックの使用	376
例: ODBC コピーブックを使用したサンプル・ プログラム	378
例: ODBC プロシージャ用のコピーブック	379
例: ODBC データ定義用のコピーブック	382
COBOL 用に切り捨てまたは省略される ODBC 名	382
ODBC 呼び出しを行うプログラムのコンパイルおよ びリンク	384

第 18 章 DB2 環境用のプログラミング

一般に、COBOL プログラムのコーディングは、プログラムから DB2 データベースにアクセスするかどうかに関係なく同じになります。しかし、DB2 データの検索、更新、挿入、および削除を行い、さらにその他の DB2 サービスを使用するためには、SQL ステートメントを使用しなければなりません。

DB2 と通信するには、以下のステップを実行します。

- EXEC SQL および END-EXEC ステートメントで区切って、必要なすべての SQL ステートメントをコーディングします。
- DB2 を開始します (まだ開始されていない場合)。
- SQL コンパイラー・オプションを使用してコンパイルします。
- NODYNAM コンパイラー・オプションを使用してコンパイルします (アプリケーションが DB2 独立型プリコンパイラーを使用してコンパイルされている場合)。

COBOL 動的呼び出しによってロードされる COBOL ライブラリーで EXEC SQL ステートメントを使用する場合は、1 つ以上の EXEC SQL ステートメントがメインプログラムにはいってなければなりません。(呼び出し元の DB2 API は COBOL 動的呼び出しではロードできません。)

関連概念

『DB2 coprocessor』

関連タスク

358 ページの『SQL ステートメントのコーディング』

360 ページの『コンパイル前の DB2 の起動』

360 ページの『SQL オプションを使用したコンパイル』

関連参照

280 ページの『DYNAM』

DB2 SQL リファレンス 第 1 巻

DB2 SQL リファレンス 第 2 巻

DB2 coprocessor

DB2 coprocessor を使用すると、別個のプリコンパイラーを使用しなくても、組み込み SQL ステートメントが含まれたソース・プログラムをコンパイラーが処理します。

コンパイラーは、ソース・プログラムで SQL ステートメントを検出すると、DB2 coprocessor とインターフェースします。この coprocessor が、SQL ステートメントに対して適切なアクションを取り、それらのために生成する固有 COBOL ステートメントをコンパイラーに指示します。

プリコンパイラーを使用するときに適用される COBOL 言語の使用に関する制約事項は、DB2 coprocessor を使用するときには適用されません。

- SQL ステートメントで使用されているホスト変数を識別するのに EXEC SQL BEGIN DECLARE SECTION および EXEC SQL END DECLARE SECTION ステートメントを使用する必要はありません。
- 複数のネストなし COBOL プログラムが入っているソース・ファイルをバッチでコンパイルできます。
- ソース・プログラムはネストされたプログラムを含むことができます。
- ソース・プログラムは、オブジェクト指向 COBOL 言語拡張機能を含むことができます。

DB2 coprocessor を使用するプログラムをコンパイルするには、SQL コンパイラー・オプションを指定する必要があります。

関連タスク

360 ページの『SQL オプションを使用したコンパイル』

関連参照

308 ページの『SQL』

SQL ステートメントのコーディング

SQL ステートメントは、EXEC SQL と END-EXEC で区切らなければなりません。EXEC SQL および END-EXEC 区切り文字はそれぞれ 1 行の中で完結している必要があります。複数行にわたって継続させることはできません。

さらに、以下の特別なステップを実行する必要があります。

- EXEC SQL INCLUDE ステートメントをコーディングして、最外部プログラムの WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION に SQL 通信域 (SQLCA) を組み込んでください。再帰的プログラムや THREAD コンパイラー・オプションを使用するプログラムの場合には、LOCAL-STORAGE をお勧めします。
- SQL ステートメントで使用するすべてのホスト変数を WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、または LINKAGE SECTION に宣言する。ただし、EXEC SQL BEGIN DECLARE SECTION および EXEC SQL END DECLARE SECTION を指定する必要はありません。

制約事項: オブジェクト指向クラスまたはメソッドで SQL ステートメントを使用することはできません。

ラージ・オブジェクト (BLOB や CLOB など) および複合 SQL の場合でも SQL ステートメントを使用できます。

関連タスク

359 ページの『DB2 coprocessor を用いた SQL INCLUDE の使用』

359 ページの『SQL ステートメントでのバイナリー項目の使用』

360 ページの『SQL ステートメントの成否の判断』

DB2 coprocessor を用いた SQL INCLUDE の使用

SQL コンパイラー・オプションを使用すると、SQL INCLUDE ステートメントは、ネイティブの COBOL COPY ステートメント (使用される検索パスおよびファイル・サフィックスを含む) とまったく同様に扱われます。

したがって、次の 2 行は同様に扱われます。(EXEC SQL INCLUDE ステートメントの終了を示すピリオドが必要です。)

```
EXEC SQL INCLUDE name END-EXEC.  
COPY name.
```

SQL INCLUDE ステートメント内の *name* は、COPY *text-name* と同じ規則に従い、REPLACING 句を持たない COPY *text-name* ステートメントとまったく同様に処理されます。

COBOL は、SQL INCLUDE 処理に対して DB2 環境変数 DB2INCLUDE を使用しません。SQL INCLUDE 処理に DB2INCLUDE 環境変数を使用する場合は、ホーム・ディレクトリーまたは AIX コマンド・シェルのプロンプトで、この環境変数を、.profile ファイルに入っている COBOL SYSLIB 環境変数の設定と連結することができます。例えば、次のように指定します。

```
export SYSLIB=$DB2INCLUDE:$SYSLIB
```

関連参照

319 ページの『第 15 章 コンパイラー指示ステートメント』
COPY ステートメント (COBOL for AIX 言語解説書)

SQL ステートメントでのバイナリー項目の使用

EXEC SQL ステートメントで指定するバイナリー・データ項目の場合は、USAGE COMP-5 としてか、USAGE BINARY、COMP、または COMP-4 として宣言することができます。

バイナリー・データ項目を USAGE BINARY、COMP、または COMP-4 として宣言する場合は、TRUNC(BIN) オプションを使用します。(この技法は、個々のデータ項目で USAGE COMP-5 を使用するよりも、パフォーマンスへの効果が大きくなる場合があります。) 代わりに、TRUNC(OPT) または TRUNC(STD) が有効な場合は、コンパイラーはその項目を受け入れますが、10 進数切り捨て規則のため、そのデータは無効なことがあります。切り捨てがデータの妥当性に影響を与えないようにしなければなりません。

関連概念

45 ページの『数値データの形式』

関連参照

312 ページの『TRUNC』

SQL ステートメントの成否の判断

DB2 は、SQL ステートメントの実行を終了すると、戻りコードを SQLCA 構造体の SQLCODE および SQLSTATE フィールドに入れて送り、操作が成功したか失敗したかを示します。プログラムは戻りコードをテストし、必要なアクションを取らなければなりません。

コンパイル前の DB2 の起動

コンパイラーは DB2 コプロセッサと連動するため、プログラムをコンパイルする前に DB2 を起動する必要があります。

コンパイルを行うためにターゲット・データベースに接続する場合は、コンパイルを開始する前に接続するか、またはコンパイラーに接続を実行させることができます。コンパイラーに接続を実行させるには、次のいずれかの方法でデータベースを指定します。

- SQL オプション内に DATABASE サブオプションを使用する。
- DB2DBDFT 環境変数でデータベース名を指定する。

SQL オプションを使用したコンパイル

SQL コンパイラー・オプションで指定されたオプション・ストリングは、DB2 コプロセッサで使用可能になります。ストリングの内容を表示するのは DB2 コプロセッサだけです。

例えば、次の cob2 コマンドは、データベース名 SAMPLE と DB2 オプション USER および USING をコプロセッサに渡します。

```
cob2 -q"sql('database sample user myname using mypassword')" mysql.cb1. . .
```

次のオプションは、プリコンパイラーにとっては意味があり使用されますが、コプロセッサには無視されます。

- MESSAGES
- NOLINEMACRO
- OPTLEVEL
- OUTPUT
- SQLCA
- TARGET
- WCHARTYPE

関連タスク

361 ページの『DB2 サブオプションの分離』

361 ページの『パッケージ名およびバインド・ファイル名の使用』

関連参照

308 ページの『SQL』

DB2 コマンド解説書 (プリコンパイル)

DB2 サブオプションの分離

複数の SQL オプション指定が連結されているので、(1 つの CBL ステートメントに収まらない可能性がある) 別々の DB2 サブオプションを複数の CBL ステートメントに分離できます。

サブオプション・ストリングに組み込まれるオプションは累積されます。コンパイラーは、複数のソースからこれらのサブオプションを、指定された順に連結します。例えば、ソース・ファイル mypgm.cbl に以下のコードが含まれているとします。

```
cb1 . . . SQL("string2") . . .  
cb1 . . . SQL("string3") . . .
```

コマンド `cob2 mypgm.cbl -q:"SQL('string1')"` を発行すると、コンパイラーは次のサブオプション・ストリングを DB2 coprocessor に渡します。

```
"string1 string2 string3"
```

連結ストリングはシングル・スペースで区切られます。コンパイラーが同じ SQL サブオプションの複数インスタンスを検出した場合は、連結ストリングの中の最後のサブオプション指定が有効になります。コンパイラーは、連結 DB2 サブオプション・ストリングの長さを 4 KB に限定しています。

パッケージ名およびバインド・ファイル名の使用

SQL オプションで指定できるサブオプションは、package name と bind file name の 2 つです。これらの名前を指定しない場合は、ソース・ファイル名 (非バッチ・コンパイルの場合) または最初のプログラム (バッチ・コンパイルの場合) に基づいてデフォルト名が作成されます。

バッチ・コンパイルの後続のネストなしプログラムについては、各プログラムの PROGRAM-ID に基づいて名前が設定されます。

パッケージ名の場合は、ベース名 (ソース・ファイル名または PROGRAM-ID) が次のように変更されます。

- 8 文字を超える名前は 8 文字に切り詰められる。
- 小文字は大文字に変換される。
- A から Z、0 から 9、または _ (下線) 以外の文字はすべて 0 に変更される。
- 先頭文字が英字でない場合は A に変換される。

したがって、ベース名が 9123aB-cd の場合、パッケージ名は A123AB0C となります。

バインド・ファイル名の場合は、ベース名に接尾部 .bnd が追加されます。明示的に指定しない限り、ファイル名は現行ディレクトリーと相対します。

第 19 章 COBOL プログラムの開発 (CICS の場合)

CICS アプリケーションは COBOL で記述することができ、TXSeries を使用して AIX ワークステーション上で実行することができます。

CICS で実行する COBOL アプリケーションを作成するには、以下の手順を実行します。

1. CICS 管理者が領域の環境ファイルを変更して、ランタイム・ディレクトリーを含むように環境変数 COBPATH、COBLIB、および LIBPATH を設定してあることを確認します。また、ランタイム・ディレクトリーに対するアクセス権が CICS 領域に付与されていることを確認します。

環境ファイルは /var/cics_regions/xxxxxxx/environment です (xxxxxxx は領域の名前です)。

2. エディターを使用して次のタスクを実行し、アプリケーションを作成します。
 - COBOL ステートメントと CICS コマンドを使用して、プログラムをコーディングします。
 - COBOL コピーブックを作成します。
 - プログラムが使用する CICS 画面マップを作成します。
3. コマンド cicsmap を使用して、画面マップを処理します。
4. cicstcl コマンドのいずれかの形式を使用して、CICS コマンドを変換し、プログラムをコンパイルしてリンクします。
 - 組み込みの CICS 変換プログラムを利用するには、cicstcl -p を使用します。
 - 分離型の CICS 変換プログラムを利用するには、-p フラグなしで cicstcl を使用します。
5. CICS 領域に対するアプリケーションのリソース (トランザクション、アプリケーション・プログラム、ファイルなど) を定義します。これらのアクションを実行するには、CICS 管理者権限が必要です。
6. cicsterm コマンドを使用するなどして、CICS 領域にアクセスします。
7. アプリケーションと関連付けられた 4 文字のトランザクション ID を入力して、アプリケーションを実行します。

関連概念

369 ページの『組み込みの CICS 変換プログラム』

関連タスク

364 ページの『CICS のもとで実行する COBOL プログラムのコーディング』

368 ページの『CICS プログラムのコンパイルおよび実行』

369 ページの『CICS プログラムのデバッグ』

TXSeries for Multiplatforms: CICS Application Programming Guide

CICS のもとで実行する COBOL プログラムのコーディング

一般に、CICS 環境では COBOL 言語がサポートされています。ただし、TXSeries で実行する COBOL プログラムをコーディングするには、注意すべき制約事項と考慮事項がいくつかあります。

制約事項:

- オブジェクト指向プログラミング、および Java とのインターオペラビリティはサポートされていません。COBOL クラス定義およびメソッドは、CICS 環境では実行できません。
- ソース・プログラムには、ネスト済みプログラムを含めてはなりません。
- TXSeries で実行される COBOL プログラムは 32 ビットでなければなりません。

変数名として EXEC、CICS、END-EXEC を使用しないでください。また、メインプログラムに対してユーザー指定のパラメーターを使用しないでください。さらに、次の COBOL 言語エレメントのいずれも使用しないことをお勧めします。

- ENVIRONMENT DIVISION の FILE-CONTROL 記入項目
- DATA DIVISION の FILE SECTION
- USE 宣言部分 (USE FOR DEBUGGING を除く)

また、CICS 環境では、次の COBOL ステートメントを使用することもお勧めできません。

- ACCEPT format 1
- CLOSE
- DELETE
- DISPLAY UPON CONSOLE、DISPLAY UPON SYSPUNCH
- MERGE
- OPEN
- READ
- REWRITE
- SORT
- START
- STOP *literal*
- WRITE

ACCEPT ステートメントの一部の形式を除き、メインフレーム CICS では上記に示す COBOL 言語エレメントをサポートしません。これらの言語エレメントを使用する場合は、次の制限に注意してください。

- プログラムをメインフレーム CICS 環境に完全に移植することはできません。
- CICS 障害が発生した場合は、上記のステートメントを使用して更新されたリソースに対してバックアウト (失敗したタスクに関連するリソースを復元すること) を実行することができません。

制限事項: 分離または統合された CICS 変換プログラムによって変換され、TXSeries 上で動作する COBOL プログラムでサポートされる zSeries ホスト・データ形式はありません。

関連タスク

『CICS のもとでのシステム日付の取得』
366 ページの『CICS での動的呼び出し』
367 ページの『SFS データへのアクセス』
367 ページの『CICS での COBOL および C/C++ 間の呼び出し』

関連参照

266 ページの『ADDR』
627 ページの『付録 C. zSeries ホスト・データ形式についての考慮事項』

CICS のもとでのシステム日付の取得

CICS プログラムでシステム日付を検索するには、形式 2 の ACCEPT ステートメントまたは CURRENT-DATE 組み込み関数を使用してください。

以下の形式 2 の ACCEPT ステートメントを CICS 環境で使用すると、システム日付を入手することができます。

- ACCEPT *identifier-2* FROM DATE (2 桁年)
- ACCEPT *identifier-2* FROM DATE YYYYMMDD
- ACCEPT *identifier-2* FROM DAY (2 桁年)
- ACCEPT *identifier-2* FROM DAY YYYYDDD
- ACCEPT *identifier-2* FROM DAY-OF-WEEK (1 桁の整数。1 は月曜日を表します。)

次に示す形式 2 の ACCEPT ステートメントを CICS 環境で使用すると、システム時刻を入手することができます。

- ACCEPT *identifier-2* FROM TIME

あるいは、CURRENT-DATE 組み込み関数を使用できます。この組み込み関数も時刻を提供できます。

これらの方法は、CICS 環境と非 CICS 環境の両方で使用できます。

CICS プログラムでは、形式 1 の ACCEPT ステートメントは使用しないでください。

関連タスク

35 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』

関連参照

CURRENT-DATE (*COBOL for AIX* 言語解説書)

CICS での動的呼び出し

CICS 環境では、CALL *identifier* ステートメントを使用して、動的呼び出しをすることができます。ただし、COBPATH 環境変数を正しく設定する必要があります。また、呼び出されたモジュールの名前が正しいことも確認する必要があります。

以下では、alpha が CICS ステートメントを含む COBOL プログラムである例を考えます。

```
WORKING-STORAGE SECTION.  
01 WS-COMMAREA PIC 9 VALUE ZERO.  
77 SUBPNAME PIC X(8) VALUE SPACES  
.  
PROCEDURE DIVISION.  
MOVE 'alpha' TO SUBPNAME.  
CALL SUBPNAME USING DFHEIBLK, DFHCOMMAREA, WS-COMMAREA.
```

CICS 制御ブロック DFHEIBLK および DFHCOMMAREA (上記に示される) を alpha に渡す必要があります。

alpha のソースはファイル alpha.ccp 内にあります。コマンド cicstcl は、alpha.ccp の変換、コンパイル、およびリンクに使用されます。COBOL でのデフォルトは大文字の名前です。したがって、PGMNAME(MIXED) オプションを使用してこのデフォルトを変更する場合以外は、ソース・ファイル ALPHA.ccp (alpha.ccp ではない) を指定して、ALPHA.ibm cob (alpha.ibm cob ではない) を作成する必要があります。

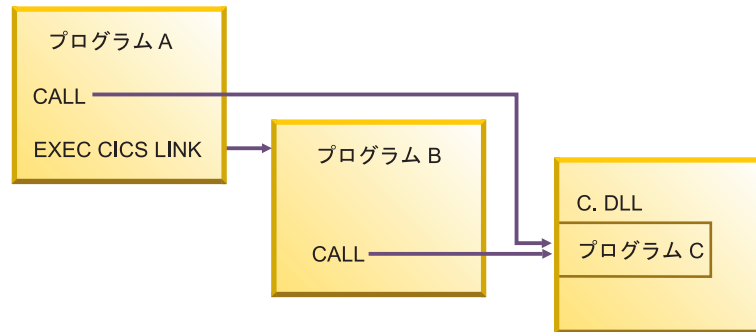
CICS 領域が green と呼ばれていると仮定します。その場合、ファイル ALPHA.ibm cob を /var/cics_regions/green/bin にコピーする必要があり、AIX SMIT ツールを使用して CICS プログラムとして ALPHA を定義する必要があります。インストール・スタッフは、ファイル /var/cics_regions/green/environment に次の行を追加する必要があります。

```
COBPATH=/var/cics_regions/green/bin
```

次に、スタッフは CICS の green 領域をシャットダウンしてから、再起動する必要があります。呼び出し先プログラムをその他のディレクトリー内に動的に入れる場合は、インストール・スタッフは COBPATH にそのディレクトリーを追加することと、CICS サーバーにそのディレクトリーへのアクセス権があることを確認してください。

共用ライブラリーの考慮事項: 1 つ以上の COBOL プログラムが含まれる共用ライブラリーは、同一 CICS トランザクション内の複数の実行単位で使用しないでください。さもないと、予測不能な結果が生じます。次の図は、2 つの異なる実行単位から同一のサブプログラムが呼び出された場合の CICS トランザクションを示しています。

- プログラム A は (C.so 内 の) プログラム C を呼び出します。
- プログラム A は EXEC CICS LINK コマンドを使用してプログラム B にリンクします。この組み合わせが、同じトランザクション内の新しい実行単位になります。
- プログラム B は (C.so 内 の) プログラム C を呼び出します。



プログラム A と B はプログラム C の同一コピーを共用するため、コピーの状態が変化すると両方のプログラムに影響を与えます。

CICS 環境では、実行単位内で初回呼び出し時にのみ、共用ライブラリー内のプログラムが初期化 (WSCLEARコンパイラー・オプションと VALUE 文節の両方の初期化) されます。1 つの COBOL サブプログラムが複数回呼び出される場合、呼び出し元のメインプログラムが同一であってもなくても、サブプログラムは初回呼び出し時にのみ初期化されます。それぞれのメインプログラムからの初回呼び出し時に、サブプログラムを初期化する必要がある場合は、サブプログラムの個々のコピーを各呼び出し側プログラムと静的にリンクします。呼び出しのたびにサブプログラムを初期化する必要がある場合は、次のいずれかの方法を使用してください。

- 再初期化するデータを WORKING-STORAGE SECTION ではなく、サブプログラムの LOCAL-STORAGE SECTION に入れる。これは、WSCLEAR コンパイラー・オプションによる初期化ではなく、VALUE 文節による初期化にのみ影響します。
- CANCEL を使用して、サブプログラムの使用後ごとにサブプログラムをキャンセルする。これにより、次の呼び出し時には、そのプログラムが初期状態になります。
- サブプログラムに INITIAL 属性を追加する。

SFS データへのアクセス

ユーザーのプログラムは、CICS で実行されていなければ、VSAM ファイル・システムを使用して SFS (TXSeries によって使用されるデフォルトのファイル・システム) にアクセスできます。

関連タスク

122 ページの『Encina SFS ファイルの識別』

CICS での COBOL および C/C++ 間の呼び出し

呼び出される側のプログラムに CICS コマンドが含まれていない場合に限り、COBOL から C/C++ プログラムへの呼び出し、または C/C++ プログラムから COBOL プログラムへの呼び出しを CICS で行うことができます。(呼び出し側プログラムに CICS コマンドを含めることはできません。)

COBOL プログラムは、C/C++ プログラムに CICS コマンドが含まれているかどうかに関係なく、EXEC CICS LINK または EXEC CICS XCTL コマンドを C/C++ プログラムに発行することができます。したがって、COBOL プログラムが、CICS コマン

ドを含む C/C++ プログラムを呼び出す場合は、COBOL CALL ステートメントではなく EXEC CICS LINK または EXEC CICS XCTL を使用してください。

関連タスク

508 ページの『COBOL および C/C++ プログラム間の呼び出し』

CICS プログラムのコンパイルおよび実行

COBOL for AIX TXSeries プログラムでは、スレッド・セーフ・バージョンの COBOL ランタイム・ライブラリーを使用する必要があります。これらのプログラムは、cob2_r コマンドと NOTHREAD コンパイラー・オプション (デフォルト) を使用してコンパイルします。

TRUNC(BIN) は、CICS で実行する COBOL プログラムに推奨されるコンパイラー・オプションです。ただし、BINARY、COMP、および COMP-4 データ項目の値が切り捨てられていない値で、PICTURE の指定に準拠することが確実な場合は、TRUNC(OPT) を使用することにより、プログラムのパフォーマンスが向上する可能性があります。

EXEC CICS コマンド引数として、BINARY、COMP、または COMP-4 データ項目の代わりに COMP-5 データ項目を使用することができます。TRUNC(BIN) が有効な場合、COMP-5 データ項目は、BINARY、COMP、または COMP-4 データ項目と同様に扱われます。

CICS Client を使用するプログラムには、PGMNAME(MIXED) コンパイラー・オプションを使用する必要があります。

COBOL プログラムを 分離型または組み込みの CICS 変換プログラムで変換する際に、DYNAM、NOLIB、THREAD、または ADDR(64) コンパイラー・オプションを使用しないでください。それ以外の COBOL コンパイラー・オプションはすべてサポートされます。

ランタイム・オプション: ASSIGN 文節によって特定のファイル・システムが選択されていない場合に使用するファイル・システムを指定するには、FILESYS ランタイム・オプションを使用します。

関連概念

369 ページの『組み込みの CICS 変換プログラム』

関連タスク

236 ページの『コマンド行からのコンパイル』

TXSeries for Multiplatforms: CICS Application Programming Guide

関連参照

263 ページの『第 14 章 コンパイラー・オプション』

265 ページの『矛盾するコンパイラー・オプション』

328 ページの『FILESYS』

組み込みの CICS 変換プログラム

CICS コンパイラー・オプションを指定して COBOL プログラムをコンパイルする場合、COBOL コンパイラーは組み込みの CICS 変換プログラムと連動して、ソース・プログラム内のネイティブ COBOL ステートメントと組み込みの CICS ステートメントの両方を処理します。

コンパイラーは、CICS ステートメントを検出したとき、およびソース・プログラム内の重要な地点で、組み込みの CICS 変換プログラムとインターフェースをとります。変換プログラムは適切な処置を行ってから、通常は、生成するネイティブ言語ステートメントを指示してコンパイラーに制御を戻します。

cicstcl コマンドを使用して COBOL プログラムをコンパイルする場合は、-p フラグを指定して組み込みの CICS 変換プログラムを使用します。cicstcl -p コマンドは、CICS コンパイラー・オプションの適切なサブオプションでコンパイラーを呼び出します。

組み込みの CICS ステートメントは、分離型の変換も引き続き可能ですが、組み込みの CICS 変換プログラムを使用することを推奨します。分離型の変換プログラムを使用する場合に適用される一部の制約は、組み込みの変換プログラムを使用する場合には適用されません。組み込みの変換プログラムを使用することにはいくつかの利点があります。

- Debugger for AIX を使用して、分離型の CICS 変換プログラムによって提供される拡張ソースではなく、元のソースをデバッグすることができます。
- コピーブック内の EXEC CICS ステートメントステートメントを個別に変換する必要がありません。
- 変換済みで未コンパイル・バージョンのソース・プログラムに対応する中間ファイルが不要です。
- 出力リストは 2 つではなく 1 つだけ作成されます。
- REPLACE ステートメントを EXEC CICS ステートメントに影響させることができます。
- CICS ステートメントが含まれているプログラムをバッチでコンパイルすることができます。

関連タスク

364 ページの『CICS のもとで実行する COBOL プログラムのコーディング』

368 ページの『CICS プログラムのコンパイルおよび実行』

関連参照

273 ページの『CICS』

CICS プログラムのデバッグ

組み込みの変換プログラムを使用して CICS プログラムをコンパイルする場合、分離型の CICS 変換プログラムが提供する拡張ソースをデバッグするのではなく、元のソース・レベルでプログラムをデバッグできます。

| 分離型の CICS 変換プログラムを使用する場合は、まず CICS プログラムを
| COBOL に変換します。次いで結果の COBOL プログラムを、他の COBOL プログ
| ラムと同様の方法でデバッグすることができます。

COBOL for AIX に付属の Debugger for AIX を使用して、CICS プログラムをデバ
ッグすることも可能です。この場合は、分散デバッガーが使用するシンボリック情
報を生成するようコンパイラーに指示する必要があります。

関連概念

331 ページの『第 17 章 デバッグ』

369 ページの『組み込みの CICS 変換プログラム』

関連タスク

236 ページの『コマンド行からのコンパイル』

TXSeries for Multiplatforms: CICS Application Programming Guide

第 20 章 Open Database Connectivity (ODBC)

Open Database Connectivity (ODBC) は、アプリケーションが構造化照会言語 (SQL) を使用して複数のデータベース管理システムにアクセスできるようにするための、アプリケーション・プログラミング・インターフェース (API) に関する仕様です。COBOL アプリケーションに ODBC インターフェースを使用すると、ODBC インターフェースをサポートするデータベースおよびファイル・システムに動的にアクセスできるようになります。

ODBC を使用すると、1 つのアプリケーションから多数の異種データベース管理システムにアクセスできるため、インターオペラビリティを最大限に高めることができます。このため、特定種のデータ・ソースに限定せずに、アプリケーションの開発、コンパイル、および出荷までを行うことができます。ユーザーはその後、データベース・ドライバーを追加して、アプリケーションに必要なデータベース管理システムとリンクすることができます。

ODBC インターフェースを使用すると、アプリケーションはドライバー・マネージャーを通して呼び出しを行うようになります。このドライバー・マネージャーは、アプリケーションの接続先となるデータベース・サーバーに必要なドライバーを動的にロードします。その後は、このドライバーが呼び出しを受け入れ、指定されたデータ・ソース (データベース) へ SQL を送信し、結果を戻します。

ODBC と組み込み SQL の比較

データベース・アクセス用の組み込み SQL を使用する COBOL アプリケーションは、特定のデータベース用のプリコンパイラーまたはコプロセッサによって処理する必要があり、ターゲット・データベースが変更された場合は再コンパイルする必要があります。ODBC は呼び出しインターフェースなので、組み込み SQL を使用した場合は異なり、ターゲット・データベースをコンパイル時に指定することはありません。複数のデータベース用に複数のアプリケーション・バージョンを用意する必要がないだけでなく、ターゲットとするデータベースをアプリケーションが動的に決定することができます。

ODBC の利点は次のとおりです。

- ODBC は、使用するデータベース・サーバーの種類を問わず、一貫性のあるインターフェースを提供します。
- 複数の同時接続を行うことができます。
- アプリケーションが稼働する各データベースとアプリケーションをバインドする必要がありません。COBOL for AIX はこのバインドを自動的に行いますが、自動的にバインドするのは、1 つのデータベースに対してのみです。接続先データベースを実行時に動的に選択する場合は、別のデータベースにバインドする追加ステップを実行する必要があります。

組み込み SQL には、次のような利点もあります。

- 静的 SQL は一般に、動的 SQL よりもパフォーマンスが高くなります。静的 SQL は実行時に準備する必要がないため、処理とネットワーク・トラフィックの両方を軽減することができます。
- 静的 SQL の場合、データベース管理者はユーザーに対し、使用するテーブルやビューそれぞれに対してアクセス権を与えるのではなく、パッケージに対するアクセス権を与えるだけで済みます。

バックグラウンド

X/Open Company と SQL Access Group は共同で、X/Open Call Level Interface という呼び出し可能 SQL インターフェースに関する仕様を作成しました。このインターフェースの目的は、アプリケーションがどのデータベース・ベンダーのプログラミング・インターフェースにも依存しないようにして、アプリケーションの移植性を高めることです。

ODBC はもともと、Microsoft® が X/Open CLI の先行ドラフトに基づいて、Microsoft オペレーティング・システム用に開発したものでした。それ以降は、他社ベンダーが UNIX システムなどの他のプラットフォーム上で動作する ODBC ドライバーを提供してきました。

ODBC 対応ソフトウェアのインストールおよび構成

ODBC を COBOL for AIX でデータ・アクセスできるようにするには、ODBC ドライバー・マネージャーおよびドライバーをインストールし、インストール済み環境に必要な ODBC データベース・ドライバーを追加し、RDBMS クライアント (DB2 UDB、Oracle 7 SQL*NET など) をインストールする必要があります。

COBOL からの ODBC 呼び出しのコーディング: 概要

COBOL プログラムから ODBC にアクセスするには、ODBC に適切なデータ型を使用し、引数を渡す方法、関数の戻り値にアクセスする方法、ビットをテストする方法などを理解する必要があります。IBM COBOL for AIX には、ODBC 呼び出しを支援するコピーブックが用意されています。

関連タスク

『ODBC に適したデータ型の使用』

373 ページの『ODBC 呼び出しにおける引数としてのポインターの受け渡し』

375 ページの『ODBC 呼び出しにおける関数戻り値へのアクセス』

375 ページの『ODBC 呼び出しにおけるビットのテスト』

376 ページの『ODBC API 用の COBOL コピーブックの使用』

384 ページの『ODBC 呼び出しを行うプログラムのコンパイルおよびリンク』

ODBC に適したデータ型の使用

ODBC API で指定されたデータ型は、API 定義の ODBC C タイプによって定義されます。示された ODBC C タイプの引数に対応する COBOL データ宣言を示します。

表 42. ODBC C タイプおよび対応する COBOL 宣言

ODBC C タイプ	COBOL 形式	説明
SQLSMALLINT	COMP-5 PIC S9(4)	符号付き短整数 (2 バイトの 2 進数)
SQLUSMALLINT	COMP-5 PIC 9(4)	符号なし短整数 (2 バイトの 2 進数)
SQLINTEGER	COMP-5 PIC S9(9)	符号付き長整数 (4 バイトの 2 進数)
SQLUIINTEGER	COMP-5 PIC 9(9)	符号なし長整数 (4 バイトの 2 進数)
SQLREAL	COMP-1	浮動小数点 (4 バイト)
SQLFLOAT	COMP-2	浮動小数点 (8 バイト)
SQLDOUBLE	COMP-2	浮動小数点 (8 バイト)
SQLCHAR	POINTER	符号なし文字を指すポインター
SQLHDBC	POINTER	接続ハンドル
SQLHENV	POINTER	環境ハンドル
SQLHSTMT	POINTER	ステートメント・ハンドル
SQLHWND	POINTER	ウィンドウ・ハンドル

COBOL の符号なし文字を指すポインターは、ヌル終了文字列を指すポインターになります。ポインター項目のターゲットを PIC X(n) と定義します。n には、ヌル終了フィールドを表すのに十分な大きさの値を入れます。ODBC API によっては、ヌル終了文字列を引数として渡す必要があります。

zSeries ホスト・データ・フォーマットは使用しないでください。ODBC API は、パラメーターがネイティブ・フォーマットであることを予期します。

関連タスク

『ODBC 呼び出しにおける引数としてのポインターの受け渡し』
375 ページの『ODBC 呼び出しにおける関数戻り値へのアクセス』

ODBC 呼び出しにおける引数としてのポインターの受け渡し

ODBC が受け入れ可能なデータ型の 1 つに対するポインター引数を指定する場合、ポインター BY REFERENCE のターゲットを渡すか、ターゲット項目を指すポインター項目を定義してそれを BY VALUE で渡すか、ターゲットの ADDRESS OF を BY VALUE で渡すかのいずれかを実行する必要があります。

例えば、関数が次のように定義されているとします。

```
RETCODE SQLSomeFunction(PSomeArgument)
```

ここで、PSomeArgument は、SomeArgument を指す引数として定義されています。この引数を SQLSomeFunction に渡すには、次のいずれかの方法を使用します。

- SomeArgument BY REFERENCE を渡す。

```
CALL "SQLSomeFunction" USING BY REFERENCE SomeArgument
```

SomeArgument が入力引数の場合は、代わりに USING BY CONTENT SomeArgument を使用することができます。

- SomeArgument を指すポインター・データ項目 PSomeArgument を定義する。

```
SET PSomeArgument TO ADDRESS OF SomeArgument  
CALL "SQLSomeFunction" USING BY VALUE PSomeArgument
```

- ADDRESS OF SomeArgument BY VALUE を渡す。

```
CALL "SQLSomeFunction" USING BY VALUE ADDRESS OF SomeArgument
```

最後の方法を使用できるのは、ターゲット引数 SomeArgument が LINKAGE SECTION 内のレベル 01 項目の場合のみです。このような場合は、次のいずれかの方法で、アドレス可能性を SomeArgument に設定することができます。

- ポインターまたは ID を使用して明示的に設定する。次に例を示します。

```
SET ADDRESS OF SomeArgument TO a-pointer-data-item
SET ADDRESS OF SomeArgument TO ADDRESS OF an-identifier
```

- SomeArgument を引数として、ODBC 関数呼び出しの実行元プログラムへ渡すことで、暗黙的に設定する。

『例: ODBC 呼び出しにおける引数としてのポインターの受け渡し』

例: ODBC 呼び出しにおける引数としてのポインターの受け渡し

次のサンプル・プログラム部分は、SQLAllocHandle 関数の呼び出し方法を示しています。

```

. . .
WORKING-STORAGE SECTION.
    COPY "odbc3.cpy".
. . .
01 SQL-RC      COMP-5    PIC S9(4).
01 Henv        POINTER.
. . .
PROCEDURE DIVISION.
. . .
    CALL "SQLAllocHandle"
        USING      By VALUE      sql=handle-env
                  sql=null-handle
        By REFERENCE Henv
        RETURNING  SQL-RC
        IF SQL-RC NOT = (SQL-SUCCESS or SQL-SUCCESS-WITH-INFO)
        THEN
            DISPLAY "SQLAllocHandle failed."
. . .
        ELSE
. . .
. . .

```

SQLConnect 関数の呼び出し例を次に示します。

例 1:

```

. . .
CALL "SQLConnect" USING BY VALUE      ConnectionHandle
                       BY REFERENCE  ServerName
                       BY VALUE      SQL-NTS
                       BY REFERENCE  UserIdentifier
                       BY VALUE      SQL-NTS
                       BY REFERENCE  AuthenticationString
                       BY VALUE      SQL-NTS
                       RETURNING      SQL-RC
. . .

```

例 2:

```

. . .
SET Ptr-to-ServerName          TO ADDRESS OF ServerName
SET Ptr-to-UserIdentifier      TO ADDRESS OF UserIdentifier
SET Ptr-to-AuthenticationString TO ADDRESS OF AuthenticationString

```

```

CALL "SQLConnect" USING BY VALUE      ConnectionHandle
                                       Ptr-to-ServerName
                                       SQL-NTS
                                       Ptr-to-UserIdentifier
                                       SQL-NTS
                                       Ptr-to-AuthenticationString
                                       SQL-NTS
                                       RETURNING      SQL-RC
. . .

```

例 3:

```

. . .
CALL "SQLConnect" USING BY VALUE      ConnectionHandle
                                       ADDRESS OF ServerName
                                       SQL-NTS
                                       ADDRESS OF UserIdentifier
                                       SQL-NTS
                                       ADDRESS OF AuthenticationString
                                       SQL-NTS
                                       RETURNING      SQL-RC
. . .

```

例 3 では、Servername、UserIdentifier、および AuthenticationString をレベル 01 項目として LINKAGE SECTION 内で定義する必要があります。

BY REFERENCE または BY VALUE 句は、別の BY REFERENCE、BY VALUE、または BY CONTENT 句によってオーバーライドされるまで、すべての引数に適用されます。

ODBC 呼び出しにおける関数戻り値へのアクセス

ODBC 呼び出しの関数戻り値を指定するには、CALL ステートメントの RETURNING 句を使用します。

```

CALL "SQLAllocEnv" USING BY VALUE Phenv RETURNING SQL-RC
IF SQL-RC NOT = SQL-SUCCESS
  THEN
    DISPLAY "SQLAllocEnv failed."
    . . .
  ELSE
    . . .
END-IF

```

ODBC 呼び出しにおけるビットのテスト

ODBC API によっては、ビット・マスクを設定し、ビットを照会する必要があります。ライブラリー・ルーチン iwzODBCTestBits を使用してビットを照会することができます。

このルーチンを次のように呼び出します。

```
CALL "iwzODBCTestBits" USING identifier-1, identifier-2 RETURNING identifier-3
```

identifier-1

フィールドはテストされます。これは、2 バイトまたは 4 バイトの 2 進数フィールド (つまり、USAGE COMP-5 PIC 9(4) または PIC 9(9)) でなければなりません。

identifier-2

テスト対象のビットを選択するビット・マスク・フィールドです。これは、*identifier-1* と同じ USAGE および PICTURE を使用して定義する必要があります。

identifier-3

テストの戻り値です (USAGE COMP-5 PIC S9(4) と定義します)。

- 0** *identifier-2* で選択されたどのビットも、*identifier-1* で ON になっていない。
- 1** *identifier-2* で選択されたすべてのビットが、*identifier-1* で ON になっている。
- 1** *identifier-2* で選択されたビットのうち、1 つ以上のビットが *identifier-1* に対して ON になっていて、1 つ以上のビットが OFF になっている。
- 100** 無効な入力引数が検出された (8 バイトの 2 進数フィールドが *identifier-1* として使用されているなど)。

| odbc2.cpy および odbc3.cpy コピーブック内で定義されたビット・マスクを使用す
| る COBOL 算術式で、1 つのフィールドに複数のビットを設定することができます
| です。例えば、次のステートメントでは、InfoValue フィールド内に、
SQL-CVT-CHAR、SQL-CVT-NUMERIC、および SQL-CVT-DECIMAL に対するビットを設定
| することができます。

```
COMPUTE InfoValue = SQL-CVT-CHAR + SQL-CVT-NUMERIC + SQL-CVT-DECIMAL
```

InfoValue を設定したら、これを 2 番目の引数として、iwzTestBits 関数に渡すこと
| ことができます。

| 上記の算術式の各オペランドは、odbc2.cpy または odbc3.cpy コピーブックで定義
| されたとおりに、結合解除ビットを表します。結果として、加算では意図したビッ
| トが ON になるように設定されます。ただし演算子は論理 OR ではないので、この
| ような算術式ではビットが繰り返されることがないように注意してください。例え
| ば、次のコードを使用すると、InfoValue 上に意図した SQL-CVT-CHAR ビットが指
| 定されなくなります。

```
COMPUTE InfoValue = SQL-CVT-CHAR + SQL-CVT-NUMERIC + SQL-CVT-DECIMAL + SQL-CVT-CHAR
```

ODBC API 用の COBOL コピーブックの使用

IBM COBOL for AIX ではコピーブックが提供されており、これによって COBOL プログラムからの ODBC 呼び出しを使用して、ODBC ドライバー対応のデータベースに容易にアクセスできるようになります。これらのコピーブックは、そのまま使用することも、変更を加えて使用することも可能です。

以下で紹介するコピーブックは、ODBC Version 3.0 に対応しています。ただし、Version 2.x のコピーブックも付属しているため、ODBC Version 2.x のアプリケーション開発が必要な場合は、Version 3.0 のコピーブックの代わりに使用することができます。

表 43. ODBC コピーブック

ODBC Version 3.0 対応のコピーブック	ODBC Version 2.x 対応のコピーブック	説明	場所
odbc3.cpy	odbc2.cpy	シンボルおよび定数	COBOL 用の include フォルダ
odbc3d.cpy	odbc2d.cpy	DATA DIVISION 定義	COBOL 用の samples フォルダ内の odbc フォルダ
odbc3p.cpy	odbc2p.cpy	PROCEDURE DIVISION ステートメント	COBOL 用の samples フォルダ内の odbc フォルダ

SYSLIB 環境変数に include および odbc フォルダのパスを組み込んで、コンパイラがコピーブックを確実に使用できるようにします。

odbc3.cpy は、ODBC API に対して記述された定数値のシンボルを定義します。このコピーブックは、ODBC API への呼び出しに使用される定数と、ODBC ガイドで指定されたシンボルをマップします。このコピーブックを使用して、引数と関数の戻り値を指定およびテストすることができます。

odbc3p.cpy は、ODBC の初期化、エラーの処理、およびクリーンアップに一般に使用される関数 (SQLAllocEnv、SQLAllocConnect、SQLAllocStmt、SQLFreeStmt、SQLDisconnect、SQLFreeConnect、SQLFreeEnv) 向けに用意されている COBOL ステートメントを使用できるようにします。

odbc3d.cpy には、WORKING-STORAGE SECTION (または LOCAL-STORAGE SECTION) で odbc3.cpy によって使用されるデータ宣言が含まれています。

これらのコピーブックでは、COBOL 固有の調整がいくつか行われています。

- 下線 (_) はハイフン (-) で置き換えられます。例えば、SQL_SUCCESS は SQL-SUCCESS として指定されます。
- 30 文字を超える名前は切り捨てられるか、または 30 文字に短縮されます (下記の関連参照を参照)。

コピーブック odbc3.cpy を組み込むには、次のように COPY ステートメントを DATA DIVISION 内に指定します。

- プログラムの場合は、COPY ステートメントを WORKING-STORAGE SECTION 内 (プログラムがネストされている場合は最外部プログラム内) に指定します。
- ODBC 呼び出しを行う各メソッドの場合は、COPY ステートメントを、(クラス定義の WORKING-STORAGE SECTION ではなく) メソッドの WORKING-STORAGE SECTION 内に指定します。

378 ページの『例: ODBC コピーブックを使用したサンプル・プログラム』

382 ページの『例: ODBC データ定義用のコピーブック』

379 ページの『例: ODBC プロシージャ用のコピーブック』

関連参照

382 ページの『COBOL 用に切り捨てまたは省略される ODBC 名』

例: ODBC コピーブックを使用したサンプル・プログラム

以下の例は、3 つの ODBC コピーブックの使用を示しています。

```
cb1  pgmname(mixed)
*****
* ODBC3EG.CBL *
*-----*
* Sample program using ODBC3, ODBC3D, and ODBC3P copybooks *
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. "ODBC3EG".
DATA DIVISION.

WORKING-STORAGE SECTION.
* copy ODBC API constant definitions
COPY "odbc3.cpy" SUPPRESS.
* copy additional definitions used by ODBC3P procedures
COPY "odbc3d.cpy".
* arguments used for SQLConnect
01 ServerName          PIC X(10) VALUE Z"Oracle7".
01 ServerNameLength    COMP-5 PIC S9(4) VALUE 10.
01 UserId              PIC X(10) VALUE Z"TEST123".
01 UserIdLength        COMP-5 PIC S9(4) VALUE 10.
01 Authentication      PIC X(10) VALUE Z"TEST123".
01 AuthenticationLength COMP-5 PIC S9(4) VALUE 10.
PROCEDURE DIVISION.
Do-ODBC SECTION.
Start-ODBC.
    DISPLAY "Sample ODBC 3.0 program starts"
* allocate henv & hdbc
PERFORM ODBC-Initialization
* connect to data source
CALL "SQLConnect" USING BY VALUE      Hdbc
                       BY REFERENCE ServerName
                       BY VALUE      ServerNameLength
                       BY REFERENCE UserId
                       BY VALUE      UserIdLength
                       BY REFERENCE Authentication
                       BY VALUE      AuthenticationLength
                       RETURNING      SQL-RC
IF SQL-RC NOT = SQL-SUCCESS
    MOVE "SQLConnect" to SQL-stmt
    MOVE SQL-HANDLE-DBC to DiagHandleType
    SET DiagHandle to Hdbc
    PERFORM SQLDiag-Function
END-IF
* allocate hstmt
PERFORM Allocate-Statement-Handle
*****
* add application specific logic here *
*****
* clean-up environment
PERFORM ODBC-Clean-Up.
* End of sample program execution
DISPLAY "Sample COBOL ODBC program ended"
GOBACK.
* copy predefined COBOL ODBC calls which are performed
COPY "odbc3p.cpy".
*****
* End of ODBC3EG.CBL: Sample program for ODBC 3.0 *
*****
```

例: ODBC プロシージャ用のコピーブック

このサンプルでは、ODBC の初期化、クリーンアップ、およびエラー処理を行うためのプロシージャを示しています。

```
*****
* ODBC3P.CPY *
*-----*
* Sample ODBC initialization, clean-up and error handling *
* procedures (ODBC Ver 3.0) *
*****
*** Initialization functions SECTION *****
ODBC-Initialization SECTION.
*
  Allocate-Environment-Handle.
  CALL "SQLAllocHandle" USING
      BY VALUE SQL-HANDLE-ENV
      BY VALUE SQL-NULL-HANDLE
      BY REFERENCE Henv
      RETURNING SQL-RC
  IF SQL-RC NOT = SQL-SUCCESS
  MOVE "SQLAllocHandle for Env" TO SQL-stmt
  MOVE SQL-HANDLE-ENV to DiagHandleType
  SET DiagHandle to Henv
  PERFORM SQLDiag-Function
  END-IF.
*
  Set-Env-Attr-to-Ver30-Behavior.
  CALL "SQLSetEnvAttr" USING
      BY VALUE Henv
      BY VALUE SQL-ATTR-ODBC-VERSION
      BY VALUE SQL-OV-ODBC3
      or SQL-OV-ODBC2 *
      for Ver 2.x behavior *
      BY VALUE SQL-IS-UIINTEGER
      RETURNING SQL-RC
  IF SQL-RC NOT = SQL-SUCCESS
  MOVE "SQLSetEnvAttr" TO SQL-stmt
  MOVE SQL-HANDLE-ENV to DiagHandleType
  SET DiagHandle to Henv
  PERFORM SQLDiag-Function
  END-IF.
*
  Allocate-Connection-Handle.
  CALL "SQLAllocHandle" USING
      By VALUE SQL-HANDLE-DBC
      BY VALUE Henv
      BY REFERENCE Hdbc
      RETURNING SQL-RC
  IF SQL-RC NOT = SQL-SUCCESS
  MOVE "SQLAllocHandle for Connection" to SQL-stmt
  MOVE SQL-HANDLE-ENV to DiagHandleType
  SET DiagHandle to Henv
  PERFORM SQLDiag-Function
  END-IF.
*** SQLAllocHandle for statement function SECTION *****
Allocate-Statement-Handle SECTION.
Allocate-Stmt-Handle.
CALL "SQLAllocHandle" USING
      By VALUE SQL-HANDLE-STMT
      BY VALUE Hdbc
      BY REFERENCE Hstmt
      RETURNING SQL-RC
  IF SQL-RC NOT = SQL-SUCCESS
  MOVE "SQLAllocHandle for Stmt" TO SQL-stmt
  MOVE SQL-HANDLE-DBC to DiagHandleType
```

```

        SET DiagHandle to Hdbc
        PERFORM SQLDiag-Function
    END-IF.
*** Cleanup Functions SECTION *****
ODBC-Clean-Up SECTION.
*
    Free-Statement-Handle.
    CALL "SQLFreeHandle" USING
        BY VALUE SQL-HANDLE-STMT
        BY VALUE Hstmt
        RETURNING      SQL-RC
    IF SQL-RC NOT = SQL-SUCCESS
        MOVE "SQLFreeHandle for Stmt" TO SQL-stmt
        MOVE SQL-HANDLE-STMT to DiagHandleType
        SET DiagHandle to Hstmt
        PERFORM SQLDiag-Function
    END-IF.
*
    SQLDisconnect-Function.
    CALL "SQLDisconnect" USING
        BY VALUE Hdbc
        RETURNING      SQL-RC
    IF SQL-RC NOT = SQL-SUCCESS
        MOVE "SQLDisconnect" TO SQL-stmt
        MOVE SQL-HANDLE-DBC to DiagHandleType
        SET DiagHandle to Hdbc
        PERFORM SQLDiag-Function
    END-IF.
*
    Free-Connection-Handle.
    CALL "SQLFreeHandle" USING
        BY VALUE SQL-HANDLE-DBC
        BY VALUE Hdbc
        RETURNING      SQL-RC
    IF SQL-RC NOT = SQL-SUCCESS
        MOVE "SQLFreeHandle for DBC" TO SQL-stmt
        MOVE SQL-HANDLE-DBC to DiagHandleType
        SET DiagHandle to Hdbc
        PERFORM SQLDiag-Function
    END-IF.
*
    Free-Environment-Handle.
    CALL "SQLFreeHandle" USING
        BY VALUE SQL-HANDLE-ENV
        BY VALUE Henv
        RETURNING      SQL-RC
    IF SQL-RC NOT = SQL-SUCCESS
        MOVE "SQLFreeHandle for Env" TO SQL-stmt
        MOVE SQL-HANDLE-ENV to DiagHandleType
        SET DiagHandle to Henv
        PERFORM SQLDiag-Function
    END-IF.
*** SQLDiag function SECTION *****
SQLDiag-Function SECTION.
SQLDiag.
    MOVE SQL-RC TO SAVED-SQL-RC
    DISPLAY "Return Value = " SQL-RC
    IF SQL-RC = SQL-SUCCESS-WITH-INFO
        THEN
            DISPLAY SQL-stmt " successful with information"
        ELSE
            DISPLAY SQL-stmt " failed"
    END-IF
*
- get number of diagnostic records - *
CALL "SQLGetDiagField"
    USING          BY VALUE      DiagHandleType
                  DiagHandle
                  0

```

```

                SQL-DIAG-NUMBER
                BY REFERENCE DiagRecNumber
                BY VALUE    SQL-IS-SMALLINT
                BY REFERENCE OMITTED
            RETURNING    SQL-RC
        IF SQL-RC = SQL-SUCCESS or SQL-SUCCESS-WITH-INFO
        THEN
*           - get each diagnostic record - *
            PERFORM WITH TEST AFTER
                VARYING DiagRecNumber-Index FROM 1 BY 1
                UNTIL DiagRecNumber-Index > DiagRecNumber
                or    SQL-RC NOT =
                    (SQL-SUCCESS or SQL-SUCCESS-WITH-INFO)
*           - get a diagnostic record - *
            CALL "SQLGetDiagRec"
                USING
                    BY VALUE    DiagHandleType
                    DiagHandle
                    DiagRecNumber-Index
                BY REFERENCE DiagSQLState
                    DiagNativeError
                    DiagMessageText
                BY VALUE    DiagMessageBufferLength
                BY REFERENCE DiagMessageTextLength
            RETURNING    SQL-RC
        IF SQL-RC = SQL-SUCCESS OR SQL-SUCCESS-WITH-INFO
        THEN
            DISPLAY "Information from diagnostic record number"
                " " DiagRecNumber-Index " for "
                SQL-stmt ":"
            DISPLAY " SQL-State = " DiagSQLState-Chars
            DISPLAY " Native error code = " DiagNativeError
            DISPLAY " Diagnostic message = "
                DiagMessageText(1:DiagMessageTextLength)
        ELSE
            DISPLAY "SQLGetDiagRec request for " SQL-stmt
                " failed with return code of: " SQL-RC
                " from SQLError"
            PERFORM Termination
        END-IF
        END-PERFORM
    ELSE
*           - indicate SQLGetDiagField failed - *
            DISPLAY "SQLGetDiagField failed with return code of: "
                SQL-RC
        END-IF
        MOVE Saved-SQL-RC to SQL-RC
        IF Saved-SQL-RC NOT = SQL-SUCCESS-WITH-INFO
        PERFORM Termination
        END-IF.
*** Termination Section*****
Termination Section.
Termination-Function.
    DISPLAY "Application being terminated with rollback"
    CALL "SQLTransact" USING BY VALUE henv
                                hdbc
                                SQL-ROLLBACK
                                RETURNING    SQL-RC
    IF SQL-RC = SQL-SUCCESS
    THEN
        DISPLAY "Rollback successful"
    ELSE
        DISPLAY "Rollback failed with return code of: "
            SQL-RC
    END-IF

```

```

STOP RUN.
*****
* End of ODBC3P.CPY      *
*****

```

例: ODBC データ定義用のコピーブック

以下のサンプルでは、メッセージ・テキストや戻りコードなどの項目に対するデータ定義を示しています。

```

*****
* ODBC3D.CPY              (ODBC Ver 3.0)                *
*-----*
* Data definitions to be used with sample ODBC function calls *
* and included in Working-Storage or Local-Storage Section *
*****
* ODBC Handles
01 Henv                POINTER          VALUE NULL.
01 Hdbc                POINTER          VALUE NULL.
01 Hstmt               POINTER          VALUE NULL.
* Arguments used for GetDiagRec calls
01 DiagHandleType     COMP-5 PIC 9(4).
01 DiagHandle         POINTER.
01 DiagRecNumber      COMP-5 PIC 9(4).
01 DiagRecNumber-Index COMP-5 PIC 9(4).
01 DiagSQLState.
  02 DiagSQLState-Chars PIC X(5).
  02 DiagSQLState-Null PIC X.
01 DiagNativeError    COMP-5 PIC S9(9).
01 DiagMessageText    PIC X(511) VALUE SPACES.
01 DiagMessageBufferLength COMP-5 PIC S9(4) VALUE 511.
01 DiagMessageTextLength COMP-5 PIC S9(4).
* Misc declarations used in sample function calls
01 SQL-RC             COMP-5 PIC S9(4) VALUE 0.
01 Saved-SQL-RC      COMP-5 PIC S9(4) VALUE 0.
01 SQL-stmt           PIC X(30).
*****
* End of ODBC3D.CPY      *
*****

```

COBOL 用に切り捨てまたは省略される ODBC 名

次の表に、30 文字を超える ODBC 名と、それに対応する COBOL 名を示します。

表 44. COBOL 用に切り捨てまたは省略される ODBC 名

ODBC C #define シンボル > 30 文字	対応する COBOL 名
SQL_AD_ADD_CONSTRAINT_DEFERRABLE	SQL-AD-ADD-CONSTRAINT-DEFER
SQL_AD_ADD_CONSTRAINT_INITIALLY_DEFERRED	SQL-AD-ADD-CONSTRAINT-INIT-DEF
SQL_AD_ADD_CONSTRAINT_INITIALLY_IMMEDIATE	SQL-AD-ADD-CONSTRAINT-INIT-IMM
SQL_AD_ADD_CONSTRAINT_NON_DEFERRABLE	SQL-AD-ADD-CONSTRAINT-NON-DEFE
SQL_AD_CONSTRAINT_NAME_DEFINITION	SQL-AD-CONSTRAINT-NAME-DEFINIT
SQL_AT_CONSTRAINT_INITIALLY_DEFERRED	SQL-AT-CONSTRAINT-INITIALLY-DE
SQL_AT_CONSTRAINT_INITIALLY_IMMEDIATE	SQL-AT-CONSTRAINT-INITIALLY-IM
SQL_AT_CONSTRAINT_NAME_DEFINITION	SQL-AT-CONSTRAINT-NAME-DEFINIT
SQL_AT_CONSTRAINT_NON_DEFERRABLE	SQL-AT-CONSTRAINT-NON-DEFERRAB
SQL_AT_DROP_TABLE_CONSTRAINT_CASCADE	SQL-AT-DROP-TABLE-CONSTRAINT-C

表 44. COBOL 用に切り捨てまたは省略される ODBC 名 (続き)

ODBC C #define シンボル > 30 文字	対応する COBOL 名
SQL_AT_DROP_TABLE_CONSTRAINT_RESTRICT	SQL-AT-DROP-TABLE-CONSTRAINT-R
SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL-C-INTERVAL-MINUTE-TO-SECON
SQL_CA_CONSTRAINT_INITIALLY_DEFERRED	SQL-CA-CONSTRAINT-INIT-DEFER
SQL_CA_CONSTRAINT_INITIALLY_IMMEDIATE	SQL-CA-CONSTRAINT-INIT-IMMED
SQL_CA_CONSTRAINT_NON_DEFERRABLE	SQL-CA-CONSTRAINT-NON-DEFERRAB
SQL_CA1_BULK_DELETE_BY_BOOKMARK	SQL-CA1-BULK-DELETE-BY-BOOKMAR
SQL_CA1_BULK_UPDATE_BY_BOOKMARK	SQL-CA1-BULK-UPDATE-BY-BOOKMAR
SQL_CDO_CONSTRAINT_NAME_DEFINITION	SQL-CDO-CONSTRAINT-NAME-DEFINI
SQL_CDO_CONSTRAINT_INITIALLY_DEFERRED	SQL-CDO-CONSTRAINT-INITIALLY-D
SQL_CDO_CONSTRAINT_INITIALLY_IMMEDIATE	SQL-CDO-CONSTRAINT-INITIALLY-I
SQL_CDO_CONSTRAINT_NON_DEFERRABLE	SQL-CDO-CONSTRAINT-NON-DEFERRA
SQL_CONVERT_INTERVAL_YEAR_MONTH	SQL-CONVERT-INTERVAL-YEAR-MONT
SQL_CT_CONSTRAINT_INITIALLY_DEFERRED	SQL-CT-CONSTRAINT-INITIALLY-DE
SQL_CT_CONSTRAINT_INITIALLY_IMMEDIATE	SQL-CT-CONSTRAINT-INITIALLY-IM
SQL_CT_CONSTRAINT_NON_DEFERRABLE	SQL-CT-CONSTRAINT-NON-DEFERRAB
SQL_CT_CONSTRAINT_NAME_DEFINITION	SQL-CT-CONSTRAINT-NAME-DEFINIT
SQL_DESC_DATETIME_INTERVAL_CODE	SQL-DESC-DATETIME-INTERVAL-COD
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQL-DESC-DATETIME-INTERVAL-PRE
SQL_DL_SQL92_INTERVAL_DAY_TO_HOUR	SQL-DL-SQL92-INTERVAL-DAY-TO-H
SQL_DL_SQL92_INTERVAL_DAY_TO_MINUTE	SQL-DL-SQL92-INTERVAL-DAY-TO-M
SQL_DL_SQL92_INTERVAL_DAY_TO_SECOND	SQL-DL-SQL92-INTERVAL-DAY-TO-S
SQL_DL_SQL92_INTERVAL_HOUR_TO_MINUTE	SQL-DL-SQL92-INTERVAL-HR-TO-M
SQL_DL_SQL92_INTERVAL_HOUR_TO_SECOND	SQL-DL-SQL92-INTERVAL-HR-TO-S
SQL_DL_SQL92_INTERVAL_MINUTE_TO_SECOND	SQL-DL-SQL92-INTERVAL-MIN-TO-S
SQL_DL_SQL92_INTERVAL_YEAR_TO_MONTH	SQL-DL-SQL92-INTERVAL-YR-TO-MO
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1	SQL-FORWARD-ONLY-CURSOR-ATTR1
SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2	SQL-FORWARD-ONLY-CURSOR-ATTR2
SQL_GB_GROUP_BY_CONTAINS_SELECT	SQL-GB-GROUP-BY-CONTAINS-SELEC
SQL_ISV_CONSTRAINT_COLUMN_USAGE	SQL-ISV-CONSTRAINT-COLUMN-USAG
SQL_ISV_REFERENTIAL_CONSTRAINTS	SQL-ISV-REFERENTIAL-CONSTRAINT
SQL_MAXIMUM_CATALOG_NAME_LENGTH	SQL-MAXIMUM-CATALOG-NAME-LENGT
SQL_MAXIMUM_COLUMN_IN_GROUP_BY	SQL-MAXIMUM-COLUMN-IN-GROUP-B
SQL_MAXIMUM_COLUMN_IN_ORDER_BY	SQL-MAXIMUM-COLUMN-IN-ORDER-B
SQL_MAXIMUM_CONCURRENT_ACTIVITIES	SQL-MAXIMUM-CONCURRENT-ACTIVIT
SQL_MAXIMUM_CONCURRENT_STATEMENTS	SQL-MAXIMUM-CONCURRENT-STAT
SQL_SQL92_FOREIGN_KEY_DELETE_RULE	SQL-SQL92-FOREIGN-KEY-DELETE-R
SQL_SQL92_FOREIGN_KEY_UPDATE_RULE	SQL-SQL92-FOREIGN-KEY-UPDATE-R
SQL_SQL92_NUMERIC_VALUE_FUNCTIONS	SQL-SQL92-NUMERIC-VALUE-FUNCTI
SQL_SQL92_RELATIONAL_JOIN_OPERATORS	SQL-SQL92-RELATIONAL-JOIN-OPER
SQL_SQL92_ROW_VALUE_CONSTRUCTOR	SQL-SQL92-ROW-VALUE-CONSTRUCTO

表 44. COBOL 用に切り捨てまたは省略される ODBC 名 (続き)

ODBC C #define シンボル > 30 文字	対応する COBOL 名
SQL_TRANSACTION_ISOLATION_OPTION	SQL-TRANSACTION-ISOLATION-OPTI

ODBC 呼び出しを行うプログラムのコンパイルおよびリンク

ODBC 呼び出しを行うプログラムを、コンパイラ・オプション PGMNAME(MIXED) を有効にしてコンパイルする必要があります。(ODBC の入り口点は、大文字小文字を区別します。)

関連参照

299 ページの『PGMNAME』

第 5 部 XML と COBOL の連携

第 21 章 XML 入力の処理	387
COBOL での XML パーサー	388
XML 文書へのアクセス	389
XML 文書の構文解析	389
XML を処理するためのプロシージャの作成	391
XML-EVENT	392
XML-CODE	393
XML-TEXT および XML-NTEXT	393
XML テキストの COBOL データ項目への変換	394
例: XML の処理用プログラム	395
構文解析の出力例	397
XML 文書のエンコード方式	397
XML 入力文書エンコード	398
コード・ページの指定	399
XML マークアップ内の EBCDIC コード・ページ依存文字	400
UTF-8 でエンコードされた XML 文書の構文解析	400
XML PARSE の例外処理	401
XML パーサーによるエラーの処理方法	403
コード・ページの矛盾の処理	404
XML 構文解析の終了	404
第 22 章 XML 出力の生成	407
XML 出力の生成	407
生成される XML 出力のエンコードの制御	411
XML GENERATE 例外の処理	412
例: XML の生成	413
プログラム XGFX	414
プログラム Pretty	415
プログラム XGFX からの出力	417
XML 出力の拡張	418
例: XML 出力の拡張	419
例: エレメントまたは属性名のハイフンを下線に変換する	421

第 21 章 XML 入力の処理

XML PARSE ステートメントを使用すると、COBOL プログラム内で XML 入力を処理できます。

XML PARSE ステートメントは、COBOL ランタイムの一部である高速 XML パーサーとの間の COBOL 言語インターフェースです。

XML 入力を処理するには、XML パーサーとの間で制御を受け渡しする必要があります。このような制御の受け渡しを開始するには、XML PARSE ステートメントを使用します。このステートメントでは、XML パーサーから制御を受け取り、パーサー・イベントを処理する処理プロシージャを指定します。

処理プロシージャで特殊レジスターを使用して、パーサーと情報を交換します。

XML 入力を処理するには、以下の COBOL 機能を使用します。

- XML PARSE ステートメントは、XML 構文解析を開始して、文書および処理プロシージャを識別します。
- 処理プロシージャは構文解析を制御します。すなわち、XML イベントおよび関連文書フラグメントを受け取って処理し、パーサーに戻って処理を続行します。
- 以下の特殊レジスターは、情報の受け渡しを行います。
 - XML-CODE は、XML 構文解析の状況を受け取り、時として、情報をパーサーに返します。
 - XML-EVENT は、各 XML イベントの名前をパーサーから受け取ります。
 - XML-NTEXT は、国別文字データとして返された XML 文書フラグメントを受け取ります。
 - XML-TEXT は、英数字データとして返された文書フラグメントを受け取ります。

関連概念

388 ページの『COBOL での XML パーサー』

398 ページの『XML 入力文書エンコード』

関連タスク

389 ページの『XML 文書へのアクセス』

389 ページの『XML 文書の構文解析』

400 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

401 ページの『XML PARSE の例外処理』

404 ページの『XML 構文解析の終了』

関連参照

691 ページの『付録 G. XML 参照資料』

Extensible Markup Language (XML)

COBOL での XML パーサー

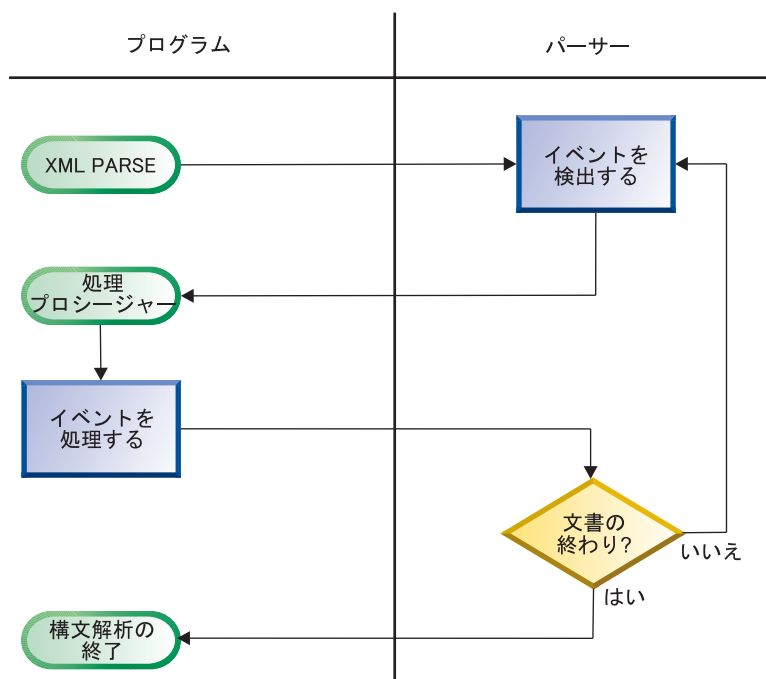
COBOL for AIX ではイベント・ベースのインターフェースが提供されるため、これを使用して XML 文書を構文解析し、さらに COBOL データ構造に変換することができます。

XML パーサーが文書内のフラグメントを検出し、作成した処理プロシージャによってそれらのフラグメントに対する操作が実行されます。フラグメントは XML イベントに関連付けられます。それぞれの XML イベントを処理するために独自の処理プロシージャをコーディングします。この操作の間中、制御がパーサーと処理プロシージャの間を行き来します。

パーサーとの受け渡しを開始するには、XML PARSE ステートメントを使用します。このステートメントに処理プロシージャを指定します。XML PARSE ステートメントを実行すると、構文解析が開始されてパーサーでの処理プロシージャが確立されます。パーサーは、文書の処理中に検出した XML イベントごとに、処理プロシージャに制御を渡します。イベントの処理後、処理プロシージャは自動的に制御をパーサーに戻します。処理プロシージャから正常に制御が返されるごとに、パーサーは XML 文書の分析を続けて次のイベントに報告します。

XML PARSE ステートメントに、構文解析の終了時に制御を渡したい 2 つの命令ステートメントを指定することもできます。1 つは正常終了の場合、もう 1 つは例外条件が存在する場合のためのステートメントです。

次の図は、パーサーとプログラム間で行われる基本的な制御受け渡しの概要を示しています。



通常、構文解析は XML 文書全体が構文解析されるまで継続されます。

XML パーサーが XML 文書を構文解析する際は、XML 文書のさまざまな側面が整形形式になっているかどうかを検査します。文書が整形形式であるのは、XML

specification に記載されている XML 構文規則に準拠し、その他のいくつかの規則 (終了タグの適切な使用、属性名が固有であることなど) に従っている場合です。

関連概念

398 ページの『XML 入力文書エンコード』

関連タスク

『XML 文書へのアクセス』

『XML 文書の構文解析』

391 ページの『XML を処理するためのプロシージャの作成』

401 ページの『XML PARSE の例外処理』

404 ページの『XML 構文解析の終了』

関連参照

700 ページの『XML 準拠』

XML specification

XML 文書へのアクセス

XML PARSE ステートメントを使用して XML 文書を構文解析する前に、その文書をプログラムで使用できるようにしておく必要があります。XML 文書を取得する一般的な方法としては、プログラムに対するパラメーターから取得する方法や、ファイルから文書を読み取る方法があります。

構文解析する XML 文書がファイル内に保管されている場合は、以下に示す通常の COBOL 機能を使用して文書をプログラムのデータ項目に入れてください。

- FILE-CONTROL 記入項目でプログラムに対してファイルを定義します。
- OPEN ステートメントでファイルをオープンします。
- READ ステートメントで、ファイルからすべてのレコードを読み取って、データ項目 (カテゴリー英数字またはカテゴリー国別の基本項目、あるいは英数字グループまたは国別グループ) に入れます。WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION でデータ項目を定義できます。
- (オプション) STRING ステートメントで、個別レコードすべてを 1 つの連続ストリームに結合したり、無関係なブランクを除去したり、可変長レコードを処理したりします。

XML 文書の構文解析

XML 文書を構文解析するには、次のコードの断片で示すように、XML PARSE ステートメントを使用して、構文解析する XML 文書、および構文解析時に発生する XML イベントを扱うための処理プロシージャを指定します。

```
XML PARSE xml-document
  PROCESSING PROCEDURE xml-event-handler
ON EXCEPTION
  DISPLAY 'XML document error ' XML-CODE
  STOP RUN
NOT ON EXCEPTION
  DISPLAY 'XML document was successfully parsed.'
```

END-XML

XML PARSE ステートメントで、まず XML 文書文字ストリームを含む構文データ項目 (上の例では xml-document) を識別します。DATA DIVISION には、文書のエンコードが Unicode UTF-16 である場合には、国別カテゴリーの基本データ項目または国別グループ項目として構文解析データ項目を定義します。それ以外の場合には、英数字グループ項目または基本英数字データ項目として構文解析データ項目を定義します。

- 構文解析データ項目が国別の場合、XML 文書は ビッグ・エンディアン形式の UTF-16 でエンコードする必要があります。
- 構文解析データ項目が英数字である場合には、そのコンテンツは、XML 文書のエンコードに関する、下記の関連参照に説明されている、サポートされているコード・ページのいずれかでエンコードする必要があります。

コード・ページの詳細については、後述の XML 入力文書エンコードに関する概念を参照してください。

次に、文書から検出した XML イベントを処理する処理プロシージャの名前 (上記の例では xml-event-handler) を指定します。

さらにオプションで、以下の句の一方または両方を指定して (上記断片を参照)、構文解析の終了後に行われるアクションを指示できます。

- ON EXCEPTION は、構文解析中に未処理の例外が発生した場合に制御を受け取りません。
- NOT ON EXCEPTION は、それ以外の場合に制御を受け取ります。

XML PARSE ステートメントを終了するには、明示範囲終了符号の END-XML を使用します。END-XML を使用して、条件ステートメント内で ON EXCEPTION 句または NOT ON EXCEPTION 句を使用する XML PARSE ステートメントをネストできます。

パーサーは、XML イベントごとに処理プロシージャに制御を渡します。処理プロシージャの終わりに到達すると、制御はパーサーに戻されます。XML パーサーと処理プロシージャ間での制御の受け渡しは、以下のイベントのいずれかが発生するまで継続します。

- XML 文書全体の構文解析が完了したことが、END-OF-DOCUMENT イベントによって示された場合。
- パーサーが文書内にエラーを検出し、EXCEPTION イベントを通知した場合。この場合、処理プロシージャは、パーサーに制御を戻す前に特殊レジスター XML-CODE をゼロにリセットしません。
- パーサーに戻る前に、XML-CODE 特殊レジスターを -1 に設定して、構文解析プロセスを故意に終了した場合。

特殊レジスター:

XML-EVENT 特殊レジスターを使用して、パーサーが処理プロシージャに渡したイベントを判別します。XML-EVENT には、'START-OF-ELEMENT' などのイベント名が入ります。パーサーは、特殊レジスター XML-TEXT または XML-NTEXT に入っているイベントの内容を渡します。

関連概念

392 ページの『XML-EVENT』

393 ページの『XML-CODE』
 398 ページの『XML 入力文書エンコード』

関連タスク

210 ページの『文字データのコード・ページの指定』
 『XML を処理するためのプロシージャの作成』
 400 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

関連参照

397 ページの『XML 文書のエンコード方式』
 691 ページの『継続を許可する XML PARSE 例外』
 697 ページの『継続を許可しない XML PARSE 例外』
 XML PARSE ステートメント (COBOL for AIX 言語解説書)

XML を処理するためのプロシージャの作成

処理プロシージャには、XML イベントを処理するためのステートメントをコーディングします。

パーサーは、イベントを検出すると、次の表に示す特殊レジスター内の処理プロシージャに情報を渡します。これらの特殊レジスターのコンテンツは、COBOL データ構造の取り込みと、処理の制御に使用します。

これらの特殊レジスターがネストされたプログラムで使用された場合は、最外部のプログラムで GLOBAL として暗黙的に定義されます。

表 45. XML パーサーが使用する特殊レジスター

特殊レジスター	暗黙的な定義および使用法	内容
XML-EVENT ^{1, 3}	PICTURE X(30) USAGE DISPLAY VALUE SPACE	XML イベントの名前
XML-CODE ²	PICTURE S9(9) USAGE BINARY VALUE ZERO	各 XML イベント用の例外コードまたはゼロ
XML-TEXT ^{1, 4}	可変長基本カテゴリ-英数字項目。	XML PARSE ID として英数字項目を指定した場合は、XML 文書のテキスト (パーサーが検出したイベントに対応)
XML-NTEXT ¹	可変長基本カテゴリ-国別項目。	XML PARSE ID として国別項目を指定した場合は、XML 文書のテキスト (パーサーが検出したイベントに対応)

1. この特殊レジスターを受け取りデータ項目として使用することはできません。

2. XML GENERATE ステートメントでも XML-CODE が使用されます。したがって、処理プロシージャ内に XML GENERATE ステートメントがある場合、XML GENERATE ステートメントの前に XML-CODE の値を保存し、XML GENERATE ステートメントの後に保存した値をリストアします。

3. この特殊レジスターの内容は、CHAR コンパイラー・オプション (EBCDIC、NATIVE、または S390) の設定によってエンコードされます。

4. XML-TEXT の内容にソース XML 文書のエンコードが含まれます (CHAR(NATIVE) コンパイラー・オプションが有効な場合は ASCII または UTF-8、CHAR(EBCDIC) が有効な場合は EBCDIC)。

制約事項:

- 処理プロシージャで、XML PARSE ステートメントを直接実行してはなりません。ただし、INVOKE または CALL ステートメントを使用して、処理プロシージャからメソッドまたは最外部のプログラムに制御が渡る場合、ターゲットとなるメソッドまたはプログラムは同一または別の XML PARSE ステートメントを実行できます。複数のスレッドで実行されているプログラムから、同一または別の XML ステートメントを同時に実行することもできます。

- 処理プロシージャの範囲で、GOBACK または EXIT PROGRAM ステートメントを実行してはいけません。ただし、制御がそれぞれ INVOKE または CALL ステートメントで渡されたメソッドまたはプログラムから制御を返す場合を除きます。この場合は、処理プロシージャの範囲で実行されます。

コンパイラーは、各処理プロシージャの最後のステートメントの後に、戻り機構を挿入します。処理プロシージャに STOP RUN ステートメントをコーディングすると、実行単位を終了させることができます。

395 ページの『例: XML の処理用プログラム』

関連概念

『XML-EVENT』

393 ページの『XML-CODE』

393 ページの『XML-TEXT および XML-NTEXT』

関連タスク

197 ページの『国別 (Unicode) 表現との変換』

394 ページの『XML テキストの COBOL データ項目への変換』

関連参照

271 ページの『CHAR』

XML-EVENT

XML 構文解析中に発生するイベントごとに、パーサーは XML-EVENT 特殊レジスターに関連イベント名を設定します。パーサーはその XML-EVENT 特殊レジスターを処理プロシージャに渡します。

イベントによっては、パーサーはイベントに関する追加情報が入った他の特殊レジスターを渡します。ほとんどの場合、パーサーは XML-TEXT または XML-NTEXT 特殊レジスターを、イベントを引き起こした XML フラグメントに設定します。

XML 文書が国別データ項目である場合、またはパーサーが Unicode 文字参照を検出した場合には、パーサーは XML-NTEXT を設定します。それ以外の場合には、パーサーは XML-TEXT を設定します。

パーサーは、文書内でエンコード競合や整形形式エラーを検出すると、XML-EVENT を 'EXCEPTION' に設定し、XML-CODE 特殊レジスターで例外に関する情報を追加します。

XML イベント・セットについて詳しくは、XML-EVENT に関する下記の関連参照を参照してください。

関連概念

393 ページの『XML-CODE』

関連タスク

389 ページの『XML 文書の構文解析』

関連参照

XML-EVENT (COBOL for AIX 言語解説書)

XML-CODE

パーサーから XML PARSE ステートメントに制御が戻されると、特殊レジスター XML-CODE には、パーサー (または XML-CODE を -1 に設定した場合には処理プロシージャ) によって設定された最新の値が入っています。

EXCEPTION イベント以外のすべてのイベントにおいて、XML-CODE の値はゼロです。処理プロシージャが、EXCEPTION 以外のイベントで制御をパーサーに返す前に XML-CODE を -1 に設定した場合、処理はユーザーが開始した COBOL 例外条件で停止します。

EXCEPTION イベントの場合、特殊レジスター XML-CODE は例外コードに設定されません。

エンコード競合例外の場合、処理プロシージャは、XML-CODE をパーサーに返す前に有効な値に初期化することがあります。XML-CODE を他のゼロ以外の値に初期化した場合、または他の例外のために初期化した場合には、パーサーは XML-CODE を元の例外コードに設定します。

XML GENERATE ステートメントでも XML-CODE 特殊レジスターが使用されます。詳細については、XML GENERATE 例外の処理に関する関連タスクを参照してください。

関連タスク

391 ページの『XML を処理するためのプロシージャの作成』

401 ページの『XML PARSE の例外処理』

412 ページの『XML GENERATE 例外の処理』

関連参照

691 ページの『付録 G. XML 参照資料』

XML-CODE (COBOL for AIX 言語解説書)

XML-TEXT および XML-NTEXT

ほとんどの XML イベントで、パーサーは XML-TEXT または XML-NTEXT を関連文書フラグメントに設定します。

通常、XML 文書が英数字データ項目である場合に、パーサーは XML-TEXT を設定します。パーサーは次の場合に、XML-NTEXT を設定します。

- XML 文書が国別データ項目である
- XML 文書が英数字データ項目であり、ATTRIBUTE-NATIONAL-CHARACTER または CONTENT-NATIONAL-CHARACTER イベントが発生した

特殊レジスター XML-TEXT と XML-NTEXT は、相互に排他的です。パーサーが XML-TEXT を設定した場合、XML-NTEXT は空で長さゼロになります。パーサーが XML-NTEXT を設定した場合、XML-TEXT は空で長さゼロになります。

XML-NTEXT 内の国別文字の数を決定するには、LENGTH 組み込み関数 (例: FUNCTION LENGTH(XML-NTEXT)) を使用します。XML-NTEXT 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NTEXT を使用します。国別文字数は、バイト数とは異なります。

XML-TEXT 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-TEXT または LENGTH 組み込み関数を使用します (それぞれバイト数を返します)。

関連概念

392 ページの『XML-EVENT』

393 ページの『XML-CODE』

関連タスク

391 ページの『XML を処理するためのプロシーチャーの作成』

関連参照

XML-TEXT (COBOL for AIX 言語解説書)

XML-NTEXT (COBOL for AIX 言語解説書)

XML テキストの COBOL データ項目への変換

XML データは固定長ではなく、固定形式でもないので、XML データを COBOL データ項目に移動するときは、特別な技法を使用する必要があります。

英数字項目の場合、XML データを COBOL 項目の左端 (デフォルト) に配置するか、右端に配置するかを決める必要があります。データが右端に配置する場合は、COBOL 項目の宣言に JUSTIFIED RIGHT 文節を指定します。

数字の XML 値、特に、'\$1,234.00' または '\$1234' といった「修飾」された通貨値には特別な配慮が必要です。この 2 つのストリングは、XML では同じものを意味しますが、COBOL 送信フィールドとしてはまったく別の宣言となる必要があります。XML データを COBOL データ項目に移動するには、以下の技法のいずれかを使用します。

- 適度に規則性のあるフォーマットの場合は、MOVE を使用して、適切な数字編集項目として再定義した英数字項目に移動します。次に、数字編集項目から移動して編集解除することにより、数字 (操作) 項目への最終的な移動を行います。(規則性のあるフォーマットとは、例えば、小数点以下の桁数が同じで、999 より大きい値にはコンマ区切り文字が付くフォーマットです。)
- 英数字の XML データに対して以下の組み込み関数を使用すると、高度な柔軟性が簡単に実現できます。
 - NUMVAL を使用して、単純な数字を表現している XML データから単純な数値を抽出およびデコードします。
 - NUMVAL-C を使用して、通貨数量を表現している XML データから数値を抽出およびデコードします。

ただし、これらの関数を使用するとパフォーマンスが下がります。

関連タスク

113 ページの『数値への変換 (NUMVAL、NUMVAL-C)』

189 ページの『COBOL での国別データ (Unicode) の使用』

391 ページの『XML を処理するためのプロシーチャーの作成』

例: XML の処理用プログラム

以下の例では、XML PARSE ステートメントおよび処理プロシーチャーの使用方法を示します。

この XML 文書は、構文解析のフローを確認する目的のソースです。プログラムの出力を以下に示します。XML 文書とプログラムの出力結果を比較して、パーサーと処理プロシーチャーとの間の相互作用を確認し、イベントと文書フラグメントを突き合わせてください。

```
Identification division.
  Program-id. XMLSAMPL.

Data division.
  Working-storage section.
*****
* XML document, encoded as initial values of data items. *
*****
  1 xml-document.
    2 pic x(39) value '<?xml version="1.0" encoding="UTF-8"'.
    2 pic x(19) value ' standalone="yes"?>'.
    2 pic x(39) value '<!--This document is just an example-->'.
    2 pic x(10) value '<sandwich>'.
    2 pic x(35) value ' <bread type="baker&apos;s best"/>'.
    2 pic x(41) value ' <?spread please use real mayonnaise ?>'.
    2 pic x(31) value ' <meat>Ham & turkey</meat>'.
    2 pic x(40) value ' <filling>Cheese, lettuce, tomato, etc.'.
    2 pic x(10) value '</filling>'.
    2 pic x(35) value ' <![CDATA[We should add a <relish>'.
    2 pic x(22) value ' element in future!]]>'.
    2 pic x(31) value ' <listprice>$4.99 </listprice>'.
    2 pic x(27) value ' <discount>0.10</discount>'.
    2 pic x(11) value '</sandwich>'.
  1 xml-document-length computational pic 999.

*****
* Sample data definitions for processing numeric XML content. *
*****
  1 current-element pic x(30).
  1 xfr-ed pic x(9) justified.
  1 xfr-ed-1 redefines xfr-ed pic 999999.99.
  1 list-price computational pic 9v99 value 0.
  1 discount computational pic 9v99 value 0.
  1 display-price pic $$9.99.

Procedure division.
  Mainline section.

    XML parse xml-document processing procedure xml-handler
      On exception
        Display 'XML document error ' XML-Code
      Not on exception
        Display 'XML document successfully parsed'
    End-XML

*****
* Process the transformed content and calculate promo price. *
*****
  Display ' '
  Display '-----***** Using information from XML '
    '*****-----'
  Display ' '
  Move list-price to display-price
  Display ' Sandwich list price: ' display-price
  Compute display-price = list-price * (1 - discount)
```

```
Display ' Promotional price: ' display-price
Display ' Get one today!'
```

```
Goback.
```

```
xml-handler section.
```

```
Evaluate XML-Event
```

```
* ==> Order XML events most frequent first
  When 'START-OF-ELEMENT'
    Display 'Start element tag: {' XML-Text '}'
    Move XML-Text to current-element
  When 'CONTENT-CHARACTERS'
    Display 'Content characters: {' XML-Text '}'
* ==> Transform XML content to operational COBOL data item...
  evaluate current-element
  When 'listprice'
* ==> Using function NUMVAL-C...
  Compute list-price = function numval-c(XML-Text)
  When 'discount'
* ==> Using de-editing of a numeric edited item...
  Move XML-Text to xfr-ed
  Move xfr-ed-1 to discount
  End-evaluate
  When 'END-OF-ELEMENT'
    Display 'End element tag: {' XML-Text '}'
    Move spaces to current-element
  When 'START-OF-DOCUMENT'
    Display 'Start of document'
  When 'END-OF-DOCUMENT'
    Display 'End of document.'
  When 'VERSION-INFORMATION'
    Display 'Version: {' XML-Text '}'
  When 'ENCODING-DECLARATION'
    Display 'Encoding: {' XML-Text '}'
  When 'STANDALONE-DECLARATION'
    Display 'Standalone: {' XML-Text '}'
  When 'ATTRIBUTE-NAME'
    Display 'Attribute name: {' XML-Text '}'
  When 'ATTRIBUTE-CHARACTERS'
    Display 'Attribute value characters: {' XML-Text '}'
  When 'ATTRIBUTE-CHARACTER'
    Display 'Attribute value character: {' XML-Text '}'
  When 'START-OF-CDATA-SECTION'
    Display 'Start of CData: {' XML-Text '}'
  When 'END-OF-CDATA-SECTION'
    Display 'End of CData: {' XML-Text '}'
  When 'CONTENT-CHARACTER'
    Display 'Content character: {' XML-Text '}'
  When 'PROCESSING-INSTRUCTION-TARGET'
    Display 'PI target: {' XML-Text '}'
  When 'PROCESSING-INSTRUCTION-DATA'
    Display 'PI data: {' XML-Text '}'
  When 'COMMENT'
    Display 'Comment: {' XML-Text '}'
  When 'EXCEPTION'
    Compute xml-document-length = function length (XML-Text)
    Display 'Exception ' XML-Code ' at offset '
      xml-document-length '.'
  When other
    Display 'Unexpected XML event: ' XML-Event '.'
  End-evaluate
  .
End program XMLSAMPL.
```

構文解析の出力例

以下の出力結果では、構文解析の各イベントが、どの文書フラグメントから発生しているかを確認することができます。

```
Start of document
Version: {1.0}
Encoding: {UTF-8}
Standalone: {yes}
Comment: {This document is just an example}
Start element tag: {sandwich}
Content characters: { }
Start element tag: {bread}
Attribute name: {type}
Attribute value characters: {baker}
Attribute value character: {'}
Attribute value characters: {s best}
End element tag: {bread}
Content characters: { }
PI target: {spread}
PI data: {please use real mayonnaise }
Content characters: { }
Start element tag: {meat}
Content characters: {Ham }
Content character: {&}
Content characters: { turkey}
End element tag: {meat}
Content characters: { }
Start element tag: {filling}
Content characters: {Cheese, lettuce, tomato, etc.}
End element tag: {filling}
Content characters: { }
Start of CDATA: {<![CDATA[
Content characters: {We should add a <relish> element in future!}
End of CDATA: {]]>}
Content characters: { }
Start element tag: {listprice}
Content characters: {$4.99 }
End element tag: {listprice}
Content characters: { }
Start element tag: {discount}
Content characters: {0.10}
End element tag: {discount}
End element tag: {sandwich}
End of document.
XML document successfully parsed

-----+***** Using information from XML *****-----

Sandwich list price: $4.99
Promotional price: $4.49
Get one today!
```

XML 文書のエンコード方式

XML 文書は、下記のサポートされたコード・ページのいずれかでエンコードする必要があります。

以下のように、XML PARSE ステートメントで構文解析する XML 文書はエンコードが必要であり、XML GENERATE ステートメントで作成した XML 文書はエンコードされません。

- 国別データ項目の文書: ビッグ・エンディアン形式の Unicode UTF-16

- ネイティブ英数字データ項目の文書: Unicode UTF-8、または International Components for Unicode (ICU) 変換ライブラリーでサポートされる 1 バイト ASCII コード・ページ

ネイティブ英数字データ項目 は、CHAR(NATIVE) コンパイラー・オプションを有効にしてコンパイルしたカテゴリ英数字データ項目、または NATIVE 句が含まれるデータ記述記入項目を持つカテゴリ英数字データ項目です。

- ホスト英数字データ項目の文書: ICU 変換ライブラリーでサポートされる 1 バイト EBCDIC コード・ページ

ホスト英数字データ項目 は、CHAR(EBCDIC) コンパイラー・オプションを有効にしてコンパイルしたカテゴリ英数字データ項目のうち、NATIVE 句が含まれないデータ記述記入項目を持つカテゴリ英数字データ項目です。

ICU 変換ライブラリーでサポートされるエンコードについては、ICU コンバーター・エクスプローラーに関する下記の関連参照を参照してください。

関連概念

『XML 入力文書エンコード』

関連タスク

399 ページの『コード・ページの指定』

関連参照

271 ページの『CHAR』

International Components for Unicode: Converter Explorer

XML 入力文書エンコード

XML PARSE ステートメントを使用して XML 文書を構文解析するには、文書はサポートされているエンコードでエンコードされている必要があります。

特定の構文解析操作に対してサポートされているエンコードは、XML 文書を含むデータ項目のタイプに依存します。パーサーは、以下のデータ項目とエンコードのタイプをサポートします。

- ビッグ・エンディアン形式で、Unicode UTF-16 でエンコードされた内容を持つカテゴリ国別データ項目
- Unicode UTF-8 またはサポートされる 1 バイトの ASCII コード・ページのいずれかをエンコードされた内容を持つネイティブ英数字データ項目
- サポートされる 1 バイトの EBCDIC コード・ページのいずれかでエンコードされた内容を持つホスト英数字データ項目

サポートされているコード・ページについては、XML 文書のエンコードに関する関連参照を参照してください。

パーサーは、XML 文書の最初の数バイトを検査することによって実際の文書エンコードを判別します。実際の文書エンコードが ASCII または EBCDIC の場合、パーサーは正しく構文解析するために特定のコード・ページ情報を必要とします。この追加のコード・ページ情報は、文書エンコード宣言または外部コード・ページ情報から取得されます。

文書エンコード宣言 は、XML 宣言のオプション部分で、文書の先頭にあります。詳細については、コード・ページの指定に関する関連タスクを参照してください。

ASCII XML 文書の 外部コード・ページ (外部 ASCII コード・ページ) は、現行のランタイム・ロケールによって示されるコード・ページです。EBCDIC XML 文書の外部コード・ページ (外部 EBCDIC コード・ページ) は、次のいずれかになります。

- EBCDIC_CODEPAGE 環境変数で指定されたコード・ページ
- EBCDIC_CODEPAGE 環境変数を設定しなかった場合は、現行のランタイム・ロケールに対して選択されたデフォルトの EBCDIC コード・ページ

指定されたエンコードがサポートされているコード化文字セットのものではない場合、パーサーは構文解析操作を実行する前に XML 例外イベントをシグナル通知します。実際の文書エンコードが指定されたエンコードに一致しない場合には、パーサーは構文解析操作の開始後に該当する XML 例外をシグナル通知します。

サポートされていないコード・ページでエンコードされた XML 文書を構文解析するには、NATIONAL-OF 組み込み関数を使用して、まず文書を国別文字データ (UTF-16) に変換します。特殊レジスタ XML-NTEXT で処理プロシージャに渡されるそれぞれの文書テキスト部分は、DISPLAY-OF 組み込み関数を使用して元のコード・ページに変換することができます。

関連タスク

- 197 ページの『国別 (Unicode) 表現との間の変換』
『コード・ページの指定』
- 400 ページの『UTF-8 でエンコードされた XML 文書の構文解析』
- 401 ページの『XML PARSE の例外処理』

関連参照

- 213 ページの『サポートされるロケールおよびコード・ページ』
- 397 ページの『XML 文書のエンコード方式』
- 691 ページの『継続を許可する XML PARSE 例外』
- 697 ページの『継続を許可しない XML PARSE 例外』

コード・ページの指定

英数字データ項目の XML 文書を構文解析するためのコード・ページを指定する方法を選択できます。

推奨する方法は、文書からエンコード宣言を省略し、代わりに外部コード・ページ情報の指定を使用することです。

エンコード宣言を省略することにより、異機種システム間で XML 文書をより簡単に伝送できるようになります。(エンコード宣言を組み込んだ場合、伝送プロセスで発生するコード・ページ変換を反映するためにエンコード宣言を更新する必要があります。)

エンコード宣言を持たない英数字 XML 文書の構文解析に使用されるコード・ページは、ランタイム・コード・ページです。

代わりに XML 宣言内にエンコード宣言を指定できます。多くの XML 文書は XML 宣言で始まります。以下に例を示します。

```
<?xml version="1.0" encoding="ibm-1140"?>
```

XML パーサーは、先頭バイトが XML 宣言で開始されていない XML 文書を検出すると例外を生成します。

エンコード宣言を指定する場合、ICU 変換ライブラリーでサポートされているいずれかの基本コード・ページ名または別名コード・ページ名を使用してください。コード・ページ名については、ICU コンバーター・エクスプローラーに関する下記の関連参照を参照してください。

XML 構文解析にサポートされているコード・ページについて詳しくは、XML 文書のエンコードに関する下記の関連参照を参照してください。

関連概念

398 ページの『XML 入力文書エンコード』

関連タスク

『UTF-8 でエンコードされた XML 文書の構文解析』

404 ページの『コード・ページの矛盾の処理』

関連参照

213 ページの『サポートされるロケールおよびコード・ページ』

397 ページの『XML 文書のエンコード方式』

International Components for Unicode: Converter Explorer

XML マークアップ内の EBCDIC コード・ページ依存文字

XML マークアップで使用されるいくつかの特殊文字には、さまざまな EBCDIC コード・ページで異なる 16 進表記があります。

次の表に、各種 EBCDIC コード・ページ CCSID の特殊文字とそれぞれの 16 進値を示します。

表 46. さまざまな EBCDIC CCSID 用特殊文字の 16 進値

文字	1047	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149
[X'AD'	X'BA'	X'63'	X'9E'	X'B5'	X'90'	X'4A'	X'B1'	X'90'	X'4A'	X'AE'
]	X'BD'	X'BB'	X'FC'	X'9F'	X'9F'	X'51'	X'5A'	X'BB'	X'B5'	X'5A'	X'9E'
!	X'5A'	X'5A'	X'4F'	X'4F'	X'4F'	X'4F'	X'BB'	X'5A'	X'4F'	X'4F'	X'4F'
	X'4F'	X'4F'	X'BB'	X'BB'	X'BB'	X'BB'	X'4F'	X'4F'	X'BB'	X'BB'	X'BB'
#	X'7B'	X'7B'	X'7B'	X'4A'	X'63'	X'B1'	X'69'	X'7B'	X'B1'	X'7B'	X'7B'

UTF-8 でエンコードされた XML 文書の構文解析

UTF-8 でエンコードされた XML 文書を、一部の追加要件が適用される点を除いて、他の XML 文書を構文解析するのと同じように構文解析することができます。

UTF-8 でエンコードされた XML 文書を構文解析するには、XML 文書の構文解析に対して通常行うような処理で XML PARSE ステートメントをコーディングします。

```
XML PARSE xml-document  
    PROCESSING PROCEDURE xml-event-handler  
    . . .  
END-XML
```

ただし、以下の追加要件に従ってください。

- 構文解析データ項目 (上記の例では xml-document) は、カテゴリー英数字データ項目でなければなりません。CHAR(EBCDIC) コンパイラー・オプションは有効になってはなりません。
- XML 文書が ASCII ではなく UTF-8 として構文解析されるようにするには、以下の条件のうち、1 つ以上の条件が適用されることを確認します。
 - ランタイム・ロケールは UTF-8 ロケールです。
 - UTF-8 を指定する XML エンコード宣言 (encoding="UTF-8") が文書に含まれています。
 - 文書は UTF-8 バイト・オーダー・マークで始まっています。
- x'FFFF' を超える Unicode スカラー値を持つ文字が文書に含まれてはなりません。そのような文字には、文字参照 ("&#xhhhhh;") を使用してください。

パーサーは、英数字特殊レジスター XML-TEXT で XML 文書の断片を返します。

UTF-8 文字 1 文字は、可変個のバイト数でエンコードされます。英数字データに対するほとんどの COBOL 操作では、1 バイトのエンコード (1 文字が 1 バイトでエンコードされるエンコード) を想定しています。UTF-8 文字を英数字データとして操作する場合、データが正常に処理されるようにする必要があります。マルチバイト文字のバイトを分割する可能性がある操作 (参照変更や切り捨てを呼び出す移動操作など) は避けてください。英数字データのマルチバイト文字を処理するために、INSPECT などのステートメントを信頼して使用することはできません。

関連概念

393 ページの『XML-TEXT および XML-NTEXT』

関連タスク

201 ページの『UTF-8 データの処理』

389 ページの『XML 文書の構文解析』

399 ページの『コード・ページの指定』

関連参照

271 ページの『CHAR』

397 ページの『XML 文書のエンコード方式』

XML PARSE ステートメント (COBOL for AIX 言語解説書)

XML PARSE の例外処理

パーサーは、構文解析中に異常またはエラーを検出すると、例外コードを XML-CODE 特殊レジスターに設定します。

XML パーサーが XML-CODE に入れて渡す例外コードが特定範囲内にある場合、処理プロシージャで例外イベントを処理して構文解析を再開できます。

処理プロシージャで例外イベントを処理するには、次の手順に従います。

1. XML-CODE の内容を検査します。
2. 必要に応じて例外を処理します。
3. XML-CODE を、例外が処理されたことを示すゼロに設定します。
4. パーサーに制御を戻します。 これにより、例外条件がなくなります。

このような方法で例外を処理できるのは、XML-CODE に入れて渡される例外コードが以下の範囲の 1 つに含まれる場合 (エンコードの矛盾が検出されたことを示します) のみです。

- 50-99
- 100,001 から 165,535
- 200,001-265,535

XML-CODE に渡される例外コードが 1 から 49 の範囲内にある例外に対して、限られた例外処理を行うことができます。この範囲内の例外が発生した後、パーサーは、戻る前に XML-CODE がゼロに設定されている場合でも、END-OF-DOCUMENT イベントを除き、それ以上標準イベントをシグナル通知しません。XML-CODE をゼロに設定した場合、パーサーは文書の構文解析を続行し、検出した例外をシグナル通知します (これは、文書内で複数のエラーを発見する方法として有効です。)

この範囲の例外発生後の構文解析の終了時に、ON EXCEPTION 句に指定されたステートメントがあればこれに制御が渡され、そうでなければ、XML PARSE ステートメントの終わりに制御が渡されます。特殊レジスター XML-CODE には、パーサーが設定した最新の例外のコードが格納されます。

この他の例外では、パーサーは追加のイベントをシグナル通知せず、ON EXCEPTION 句に指定されているステートメントに制御を渡します。この場合、パーサーに制御を返す前に処理プロシージャで XML-CODE を初期化したとしても、XML-CODE には元の例外番号が入ります。

例外を処理する必要がない場合は、XML-CODE の値を変更しないでパーサーに制御を戻します。パーサーは、ON EXCEPTION 句で指定されたステートメントに制御権を移動します。ON EXCEPTION 句がコーディングされていない場合、制御は XML PARSE ステートメントの終わりに移動します。

構文解析の終了時点までに未処理の例外がなかった場合は、NOT ON EXCEPTION 句に指定されたステートメントに制御が渡されます (構文解析の正常終了)。NOT ON EXCEPTION 句をコーディングしなかった場合、制御は XML PARSE ステートメントの終わりに渡されます。特殊レジスター XML-CODE はゼロに設定されます。

関連概念

393 ページの『XML-CODE』

398 ページの『XML 入力文書エンコード』

403 ページの『XML パーサーによるエラーの処理方法』

関連タスク

391 ページの『XML を処理するためのプロシーチャーの作成』

404 ページの『コード・ページの矛盾の処理』

XML パーサーによるエラーの処理方法

XML 文書の中にエラーがあるのを検出すると、XML パーサーは XML 例外イベントを生成し、制御を処理プロシーチャーに渡します。

パーサーは、以下の情報を特殊レジスターに入れて提供します。

- XML-EVENT に 'EXCEPTION' が設定されている。
- XML-CODE に数値の例外コードが設定されている。例外コードは下記の関連参照の XML PARSE 例外で記述されています。
- XML-TEXT または XML-NTEXT には、例外検出ポイントまでの文書テキストが含まれています。

すべての他の XML 特殊レジスターは空で、長さゼロです。

例外コードの数字が次の範囲のいずれかである場合には、処理プロシーチャーで例外を処理して、構文解析を続行することができます。

- 1-99
- 100,001 から 165,535
- 200,001-265,535

例外コードがその他のゼロ以外の値である場合は、構文解析を続けることはできません。エンコード方式の矛盾の例外 (50 から 99 および 300 から 399) は、文書の構文解析が開始される前にシグナル通知されます。このような例外の場合、XML-TEXT または XML-NTEXT は長さがゼロになるか、文書のエンコード宣言値のみが入ります。

1 から 49 の範囲の例外は XML 仕様に基づく致命的エラーです。したがって、ユーザーが例外を処理しても、パーサーは通常の構文解析を続けることはできません。ただし、パーサーでは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーが検出されるまで、その他のエラーの走査を続けます。このような例外の場合、パーサーでは、END-OF-DOCUMENT イベント以外については、追加の標準イベントをシグナル通知しません。

関連概念

393 ページの『XML-CODE』

398 ページの『XML 入力文書エンコード』

関連タスク

401 ページの『XML PARSE の例外処理』

404 ページの『コード・ページの矛盾の処理』

404 ページの『XML 構文解析の終了』

関連参照

691 ページの『継続を許可する XML PARSE 例外』

697 ページの『継続を許可しない XML PARSE 例外』

コード・ページの矛盾の処理

ご使用の処理プロシージャが、特定の文書エンコード競合の例外を処理できる場合があります。

文書項目が英数字であり、XML-CODE の例外コードが 100,001 から 165,535 までの間にあるか、または 200,001 から 265,535 までの間にある例外イベントは、(エンコード宣言で指定された) 文書のコード・ページが、外部コード・ページ情報と矛盾していることを示しています。

この特殊ケースでは、XML-CODE の値から 100,000 または 200,000 (それぞれ EBCDIC コード・ページか ASCII コード・ページかによって異なる) を減算することで、文書のコード・ページを使用して構文解析を行うようにできます。例えば、XML-CODE が 101,140 に設定されている場合、文書のコード・ページは 1140 です。別の方法では、パーサーに戻る前に XML-CODE をゼロに設定して、外部コード・ページを使用して構文解析することもできます。

パーサーは、コード・ページ矛盾の例外イベント用の処理プロシージャから戻ると、以下の 3 つの処置のいずれかをとります。

- XML-CODE をゼロに設定した場合、パーサーは文書データ項目がネイティブ英数字かホスト英数字項目かによって、外部 ASCII コード・ページまたは外部 EBCDIC コード・ページを使用します。
- XML-CODE に文書のコード・ページ (すなわち、元の XML-CODE 値から 100,000、必要に応じて 200,000 を減算した値) を設定すると、パーサーは文書のコード・ページを使用します。処理プロシージャからの戻り時に XML-CODE が非ゼロ値に設定されていたとき、パーサーが処理を続けるのはこのケースに該当する場合だけです。
- それ以外の場合、パーサーは文書の処理を停止し、例外条件とともに制御を XML PARSE ステートメントに戻します。XML-CODE は、当初に例外イベントへ渡された例外コードに設定されます。

関連概念

393 ページの『XML-CODE』

398 ページの『XML 入力文書エンコード』

403 ページの『XML パーサーによるエラーの処理方法』

関連タスク

401 ページの『XML PARSE の例外処理』

関連参照

691 ページの『継続を許可する XML PARSE 例外』

697 ページの『継続を許可しない XML PARSE 例外』

XML 構文解析の終了

通常の XML イベント (つまり、EXCEPTION 以外のイベント) からパーサーに戻る前に、処理プロシージャで XML-CODE を -1 に設定すると、残りの XML 文書テキストを処理せずに直ちに構文解析を終了することができます。

この技法は、文書を十分に確認済みである場合、あるいは文書に何らかの不規則性があるためにそれ以上処理を続けても意味がないことがわかった場合に使用します。

この場合、パーサーはそれ以上イベント (XML 例外イベントを含む) をシグナル通知しません。制御は、XML PARSE ステートメントの ON EXCEPTION 句が指定されている場合にはその句に渡されます。ON EXCEPTION 句の命令ステートメントでは、XML-CODE が -1 であるかどうかを検査できます。これは、ユーザーが意図的に構文解析を終了したことを示します。ON EXCEPTION 句が指定されていない場合、制御は XML PARSE ステートメントの終わりに渡されます。

また、XML-CODE を変更せずにパーサーに戻ることによって、XML EXCEPTION イベントの発生後に構文解析を終了することができます。この場合、結果は意図的に終了した場合と似ていますが、XML-CODE に元の例外コードが入った状態でパーサーが XML PARSE ステートメントに戻る点は除きます。

関連概念

393 ページの『XML-CODE』

403 ページの『XML パーサーによるエラーの処理方法』

関連タスク

401 ページの『XML PARSE の例外処理』

第 22 章 XML 出力の生成

XML GENERATE ステートメントを使用して、COBOL プログラムから XML 出力を生成させることができます。

XML GENERATE ステートメントでは、ソースおよび出力データ項目を指定します。オプションとして、次のものも指定できます。

- 生成された XML 文字のカウントを受け取るフィールド
- 生成された XML 文書のエンコード
- 生成された文書の名前空間
- 各エレメントの開始および終了タグを修飾する名前空間接頭部 (名前空間を指定した場合)
- 例外発生時に制御を受け取るステートメント

オプションとして、文書で XML 宣言を生成して、適格なソース・データ項目を出力でエレメントとしてではなく属性として表すことができます。

XML-CODE 特殊レジスターを使用して、XML 生成の状況を判別できます。

COBOL データ項目を XML に変換した後、得られた XML 出力をさまざまな方法で使用できます。例えば、Web サービスにそれを配置したりファイルに記述したり、他のプログラムにパラメーターとして渡したりできます。

関連タスク

『XML 出力の生成』

411 ページの『生成される XML 出力のエンコードの制御』

412 ページの『XML GENERATE 例外の処理』

418 ページの『XML 出力の拡張』

関連参照

Extensible Markup Language (XML)

XML GENERATE ステートメント (*COBOL for AIX 言語解説書*)

XML 出力の生成

COBOL データを XML に変換するには、以下の例のように XML GENERATE ステートメントを使用してください。

```
XML GENERATE XML-OUTPUT FROM SOURCE-REC
      COUNT IN XML-CHAR-COUNT
ON EXCEPTION
  DISPLAY 'XML generation error ' XML-CODE
  STOP RUN
NOT ON EXCEPTION
  DISPLAY 'XML document was successfully generated.'
END-XML
```

XML GENERATE ステートメントでは、XML 出力を受け取るデータ項目 (上の例では XML-OUTPUT) をまず識別します。データ項目は、生成された XML 出力を格納でき

る十分な大きさに定義します。通常、データ名の長さに応じて、COBOL ソース・データ・サイズの 5 倍から 10 倍に定義します。

DATA DIVISION では、受信 ID を、英数字 (英数字グループ項目またはカテゴリ英数字の基本項目のどちらか) として、あるいは国別 (国別グループ項目またはカテゴリ国別の基本項目のどちらか) として宣言できます。

次に、XML フォーマットに変換されるソース・データ項目 (この例では SOURCE-REC) を識別します。ソース・データ項目は、英数字グループ項目、国別グループ項目、あるいはクラス英数字または国別の基本データ項目にすることができます。

COBOL データ項目の中には、XML に変換されず無視されるものがあります。XML に変換する英数字グループ項目または国別グループ項目の従属データ項目は、次のような場合は無視されます。

- REDEFINES 節を指定しているか、またはそのような再定義項目に従属しているもの。
- RENAMES 節を指定しているもの。

ソース・データ項目の中の次のような項目も、XML の生成時に無視されます。

- 基本 FILLER (または名前なしの) データ項目
- SYNCHRONIZED データ項目で挿入されている遊びバイト

XML を読みやすくするために余分な空白文字 (例えば、改行または字下げ) が挿入されることはありません。

必要に応じて、COUNT IN 句をコーディングして、XML 出力の生成時に充てんされる XML 文字エンコード・ユニット数を取得できます。受け取る ID のカテゴリが国別である場合、カウントは、UTF-16 文字エンコード・ユニット数です。すべての他のエンコード (UTF-8 を含む) では、カウントはバイト数です。

カウント・フィールドを参照変更長として使用して、生成された XML 出力を含む受け取りデータ項目の一部のみを取得できます。例えば、XML-OUTPUT(1:XML-CHAR-COUNT) は、XML-OUTPUT の最初の XML-CHAR-COUNT 文字位置を参照します。

次のプログラムの抜粋を検討します。

```
01 doc pic x(512).
01 docSize pic 9(9) binary.
01 G.
   05 A pic x(3) value "aaa".
   05 B.
      10 C pic x(3) value "ccc".
      10 D pic x(3) value "ddd".
   05 E pic x(3) value "eee".
   . . .
XML Generate Doc from G
```

上のコードによって、次の XML 文書が生成されます。ここで、A、B、および E は、エレメント G の子エレメントとして表され、C および D は、エレメント B の子エレメントになります。

```
<G><A>aaa</A><B><C>ccc</C><D>ddd</D></B><E>eee</E></G>
```


また、XML GENERATE ステートメントの ATTRIBUTES 句を指定することもできます。ATTRIBUTES 句を使用すると、生成された XML 文書内に含まれる各基本データ項目が (このようなデータ項目が FILLER 以外の名前を持ち、データ記述項目に OCCURS 節がない場合)、子エレメントとしてではなく、すぐ上位のデータ項目に対応する XML エレメントの属性として表されます。

たとえば、上記プログラムの抜粋の XML GENERATE ステートメントが次のようにコーディングされたとします。

```
XML Generate Doc from G with attributes
```

このコードによって次の XML 文書が生成されます。ここで、A および E は、エレメント G の属性として表され、C および D はエレメント B の属性になります。

```
<G A="aaa" E="eee"><B C="ccc" D="ddd"></B></G>
```

オプションとして、XML GENERATE ステートメントの ENCODING 句をコーディングして、生成される XML 文書のエンコードを指定できます。ENCODING 句を使用しなかった場合、文書エンコードは受け取りデータ項目のカテゴリによって決まります。詳細については、生成された XML 出力のエンコードの制御に関する下記の関連タスクを参照してください。

オプションとして、XML-DECLARATION 句をコーディングして、生成された XML 文書にバージョン情報およびエンコード宣言を含んだ XML 宣言を組み込むことができます。受け取りデータ項目のカテゴリによって次のようになります。

- 国別の場合: エンコード・宣言には値 UTF-16 が含まれます (encoding="UTF-16")。
- 英数字の場合: エンコード宣言は、ENCODING 句 (指定されている場合) またはランタイム・ロケールまたは EBCDIC_CODEPAGE 環境変数 (ENCODING 句が指定されていない場合) から派生します。

例えば、下記のプログラムの抜粋では、XML GENERATE の XML-DECLARATION 句を指定して、エンコードを UTF-8 で指定しています。

```
01 Greeting.  
   05 msg pic x(80) value 'Hello, world!'.  
   . . .  
   XML Generate Doc from Greeting  
     with Encoding "UTF-8"  
     with XML-declaration  
   End-XML
```

上のコードによって、次の XML 文書が生成されます。

```
<?xml version="1.0" encoding="UTF-8"?><Greeting><msg>Hello, world!</msg></Greeting>
```

XML-DECLARATION 句をコーディングしなければ、XML 宣言は生成されません。

オプションとして、NAMESPACE 句をコーディングして、生成される XML 文書の名前空間を指定することもできます。名前空間の値は有効な URI (Uniform Resource Identifier) (例: URL (Uniform Resource Locator)) である必要があります。詳細については、下記の URI 構文に関する関連概念を参照してください。

名前空間は、カテゴリが国別または英数字の ID またはリテラルで指定します。

名前空間を指定して名前空間接頭部 (下記) を指定しない場合には、その名前空間は文書のデフォルト名前空間になります。つまり、ルート・エレメントで宣言された名前空間が文書内のルート・エレメントを含む各エレメント名にデフォルトで適用されます。

たとえば、次のデータ定義および XML GENERATE ステートメントを検討してみましょう。

```
01 Greeting.  
   05 msg pic x(80) value 'Hello, world!'.  
01 NS pic x(20) value 'http://example'.  
   . . .  
   XML Generate Doc from Greeting  
     namespace is NS
```

次に示すとおり、結果として得られる XML 文書は、デフォルト名前空間 (`http://example`) を持ちます。

```
<Greeting xmlns="http://example"><msg>Hello, world!</msg></Greeting>
```

名前空間を指定しなければ、生成される XML 文書内のエレメント名は、どの名前空間にも属しません。

オプションとして、NAMESPACE-PREFIX 句をコーディングして、生成される文書内の各エレメントの開始および終了タグに適用する接頭部を指定することもできます。接頭部は、上述のように名前空間を指定した場合にのみ指定できます。

XML GENERATE ステートメントを実行する場合、接頭部の値はコロン (:) なしの有効な XML 名でなければなりません。詳しくは、名前空間に関する下記の関連参照を参照してください。値には末尾スペースを含めることができますが、接頭部の使用前に除去されます。

名前空間接頭部は、カテゴリーが国別または英数字の ID またはリテラルで指定します。

接頭部は、各エレメントの開始および終了タグを修飾するため、短くすることを推奨します。

たとえば、次のデータ定義および XML GENERATE ステートメントを検討してみましょう。

```
01 Greeting.  
   05 msg pic x(80) value 'Hello, world!'.  
01 NS pic x(20) value 'http://example'.  
01 NP pic x(5) value 'pre'.  
   . . .  
   XML Generate Doc from Greeting  
     namespace is NS  
     namespace-prefix is NP
```

次のように、結果として得られる XML 文書は明示的な名前空間 (`http://example`) を持ち、接頭部 `pre` がエレメント `Greeting` および `msg` の開始および終了タグに適用されます。

```
<pre:Greeting xmlns:pre="http://example"><pre:msg>Hello, world!</pre:msg></pre:Greeting>
```

さらに、XML 文書の生成後に制御を受け取るために、以下の句のいずれかまたは両方を指定できます。

- ON EXCEPTION。XML 生成時にエラーが発生したときに制御を受け取る場合。
- NOT ON EXCEPTION。エラーが発生しなかったときに制御を受け取る場合。

XML GENERATE ステートメントを終了するには、明示範囲終了符号の END-XML を使用します。条件ステートメントで ON EXCEPTION または NOT ON EXCEPTION 句を指定している XML GENERATE ステートメントをネストするには、END-XML をコーディングします。

XML への COBOL ソース・レコードの変換が完了するか、またはエラーが発生するまで、XML 生成は継続します。エラーが発生した場合、結果は次のようになります。

- XML-CODE 特殊レジスターには、ゼロ以外の例外コードが含まれます。
- ON EXCEPTION 句が指定されている場合、これに制御が渡されます。指定されていない場合は、XML GENERATE ステートメントの最後に制御が渡されます。

XML 生成時にエラーが発生しなかった場合、XML-CODE 特殊レジスターにはゼロが入り、制御は、NOT ON EXCEPTION 句が指定されている場合はこの句に、指定されていない場合は XML GENERATE ステートメントの最後に渡されます。

413 ページの『例: XML の生成』

関連概念

Uniform Resource Identifier (URI): Generic Syntax

関連タスク

『生成される XML 出力のエンコードの制御』

412 ページの『XML GENERATE 例外の処理』

201 ページの『UTF-8 データの処理』

関連参照

XML GENERATE ステートメント (*COBOL for AIX 言語解説書*)

Extensible Markup Language (XML)

Namespaces in XML 1.0

生成される XML 出力のエンコードの制御

XML GENERATE ステートメントを使用して XML 出力を生成する場合、出力を受け取るデータ項目のカテゴリによって、および XML GENERATE ステートメントの WITH ENCODING 句を使用して文書エンコードを指定することによって、出力のエンコードを制御することができます。

WITH ENCODING *codepage* 句を指定する場合、*codepage* は、COBOL XML 処理に対してサポートされているいずれかのコード・ページを識別するものでなければなりません。詳しくは、XML 文書のエンコードに関する下記の関連参照を参照してください。整数となる場合の *codepage* は、有効な CCSID 番号でなければなりません。クラス英数字または国別となる場合の *codepage* は、International Components for Unicode (ICU) 変換ライブラリーでサポートされるコード・ページ名を識別するものでなければなりません (下記の Converter Explorer 表を参照)。

WITH ENCODING 句をコーディングしなければ、生成される XML 出力は下表のとおりエンコードされます。

表 47. ENCODING 句を省略した場合に生成される XML のエンコード

受信 XML ID の定義	生成される XML 出力のエンコード
ネイティブ英数字 (CHAR(EBCDIC) が有効になっていないか、またはデータ記述に NATIVE 句が含まれる)	有効なランタイム・ロケールによって指定された ASCII または UTF-8 コード・ページ
ホスト英数字 (CHAR(EBCDIC) が有効になっていて、データ記述に NATIVE 句が含まれない)	有効な EBCDIC コード・ページ ¹
国別	ビッグ・エンディアン形式の UTF-16

1. EBCDIC_CODEPAGE 環境変数を使用することによって、EBCDIC コード・ページを設定できます。環境変数が設定されていない場合には、エンコードは、現在のランタイム・ロケールに関連付けられたデフォルトの EBCDIC コード・ページになります。

バイト・オーダー・マークは生成されません。

データ項目が XML へ変換される方法および XML エlement 名および属性名が COBOL データ名から形成される方法に関する詳細については、XML GENERATE ステートメントの操作に関する以下の関連参照を参照してください。

関連タスク

209 ページの『第 11 章 ロケールの設定』

225 ページの『環境変数の設定』

関連参照

271 ページの『CHAR』

397 ページの『XML 文書のエンコード方式』

XML GENERATE ステートメント (COBOL for AIX 言語解説書)

XML GENERATE の操作 (COBOL for AIX 言語解説書)

International Components for Unicode: Converter Explorer

XML GENERATE 例外の処理

XML 出力の生成時にエラーが検出された場合、例外条件が存在します。エラー・タイプを示す数値の例外コードが格納される、XML-CODE 特殊レジスターを検査するコードを記述することができます。

エラーを処理するには、XML GENERATE ステートメントの以下の句の一方または両方を使用してください。

- ON EXCEPTION
- COUNT IN

XML GENERATE ステートメントに ON EXCEPTION 句をコーディングした場合、指定された命令ステートメントに制御が転送されます。命令ステートメントをコーディングして、例えば、XML-CODE 値を表示できます。ON EXCEPTION 句がコーディングされていない場合、制御は XML GENERATE ステートメントの終わりに移動します。

プログラム XGFX

```
Identification division.
  Program-id. XGFX.
Data division.
  Working-storage section.
    01 numItems pic 99 global.
    01 purchaseOrder global.
      05 orderDate pic x(10).
      05 shipTo.
        10 country pic xx value 'US'.
        10 name pic x(30).
        10 street pic x(30).
        10 city pic x(30).
        10 state pic xx.
        10 zip pic x(10).
      05 billTo.
        10 country pic xx value 'US'.
        10 name pic x(30).
        10 street pic x(30).
        10 city pic x(30).
        10 state pic xx.
        10 zip pic x(10).
      05 orderComment pic x(80).
    05 items occurs 0 to 20 times depending on numItems.
      10 item.
        15 partNum pic x(6).
        15 productName pic x(50).
        15 quantity pic 99.
        15 USPrice pic 999v99.
        15 shipDate pic x(10).
        15 itemComment pic x(40).
    01 numChars comp pic 999.
    01 xmlPO pic x(999).
Procedure division.
  m.
    Move 20 to numItems
    Move spaces to purchaseOrder

    Move '1999-10-20' to orderDate

    Move 'US' to country of shipTo
    Move 'Alice Smith' to name of shipTo
    Move '123 Maple Street' to street of shipTo
    Move 'Mill Valley' to city of shipTo
    Move 'CA' to state of shipTo
    Move '90952' to zip of shipTo

    Move 'US' to country of billTo
    Move 'Robert Smith' to name of billTo
    Move '8 Oak Avenue' to street of billTo
    Move 'Old Town' to city of billTo
    Move 'PA' to state of billTo
    Move '95819' to zip of billTo
    Move 'Hurry, my lawn is going wild!' to orderComment

    Move 0 to numItems
    Call 'addFirstItem'
    Call 'addSecondItem'
    Move space to xmlPO
    Xml generate xmlPO from purchaseOrder count in numChars
      with xml-declaration with attributes
      namespace 'http://www.example.com' namespace-prefix 'po'
    Call 'pretty' using xmlPO value numChars
    Goback
  .
```

```

Identification division.
  Program-id. 'addFirstItem'.
Procedure division.
  Add 1 to numItems
  Move '872-AA' to partNum(numItems)
  Move 'Lawnmower' to productName(numItems)
  Move 1 to quantity(numItems)
  Move 148.95 to USPrice(numItems)
  Move 'Confirm this is electric' to itemComment(numItems)
  Goback.
End program 'addFirstItem'.

Identification division.
  Program-id. 'addSecondItem'.
Procedure division.
  Add 1 to numItems
  Move '926-AA' to partNum(numItems)
  Move 'Baby Monitor' to productName(numItems)
  Move 1 to quantity(numItems)
  Move 39.98 to USPrice(numItems)
  Move '1999-05-21' to shipDate(numItems)
  Goback.
End program 'addSecondItem'.

End program XGFX.

```

プログラム Pretty

```

|
| Identification division.
|   Program-id. Pretty.
| Data division.
|   Working-storage section.
|     01 prettyPrint.
|       05 pose pic 999.
|       05 posd pic 999.
|       05 depth pic 99.
|       05 inx pic 999.
|       05 elementName pic x(30).
|       05 indent pic x(40).
|       05 buffer pic x(998).
|       05 lastitem pic 9.
|         88 unknown value 0.
|         88 xml-declaration value 1.
|         88 element value 2.
|         88 attribute value 3.
|         88 charcontent value 4.
|   Linkage section.
|     1 doc.
|       2 pic x occurs 16384 times depending on len.
|       1 len comp-5 pic 9(9).
|   Procedure division using doc value len.
|     m.
|       Move space to prettyPrint
|       Move 0 to depth
|       Move 1 to posd pose
|       Xml parse doc processing procedure p
|       Goback
|     .
|     p.
|       Evaluate xml-event
|         When 'VERSION-INFORMATION'
|           String '<?xml version="' xml-text '"' delimited by size
|             into buffer with pointer posd
|           Set xml-declaration to true
|         When 'ENCODING-DECLARATION'
|           String ' encoding="' xml-text '"' delimited by size

```

```

        into buffer with pointer posd
When 'STANDALONE-DECLARATION'
    String ' standalone="" xml-text ""' delimited by size
        into buffer with pointer posd
When 'START-OF-ELEMENT'
    Evaluate true
        When xml-declaration
            String '?>' delimited by size into buffer
                with pointer posd
            Set unknown to true
            Perform printline
            Move 1 to posd
        When element
            String '>' delimited by size into buffer
                with pointer posd
        When attribute
            String '>' delimited by size into buffer
                with pointer posd
    End-evaluate
    If elementName not = space
        Perform printline
    End-if
    Move xml-text to elementName
    Add 1 to depth
    Move 1 to pose
    Set element to true
    String '<' xml-text delimited by size
        into buffer with pointer pose
    Move pose to posd
When 'ATTRIBUTE-NAME'
    If element
        String ' ' delimited by size into buffer
            with pointer posd
    Else
        String '" ' delimited by size into buffer
            with pointer posd
    End-if
    String xml-text '=' delimited by size into buffer
        with pointer posd
    Set attribute to true
When 'ATTRIBUTE-CHARACTERS'
    String xml-text delimited by size into buffer
        with pointer posd
When 'ATTRIBUTE-CHARACTER'
    String xml-text delimited by size into buffer
        with pointer posd
When 'CONTENT-CHARACTERS'
    Evaluate true
        When element
            String '>' delimited by size into buffer
                with pointer posd
        When attribute
            String '>' delimited by size into buffer
                with pointer posd
    End-evaluate
    String xml-text delimited by size into buffer
        with pointer posd
    Set charcontent to true
When 'CONTENT-CHARACTER'
    Evaluate true
        When element
            String '>' delimited by size into buffer
                with pointer posd
        When attribute
            String '>' delimited by size into buffer
                with pointer posd
    End-evaluate

```



```

String xml-text delimited by size into buffer
  with pointer posd
Set charcontent to true
When 'END-OF-ELEMENT'
  Move space to elementName
  Evaluate true
  When element
    String '/>' delimited by size into buffer
    with pointer posd
  When attribute
    String '">' delimited by size into buffer
    with pointer posd
  When other
    String '</' xml-text '>' delimited by size
    into buffer with pointer posd
  End-evaluate
Set unknown to true
Perform printline
Subtract 1 from depth
Move 1 to posd
When other
  Continue
End-evaluate
.
printline.
Compute inx = function max(0 2 * depth - 2) + posd - 1
If inx > 120
  compute inx = 117 - function max(0 2 * depth - 2)
  If depth > 1
    Display indent(1:2 * depth - 2) buffer(1:inx) '...'
  Else
    Display buffer(1:inx) '...'
  End-if
Else
  If depth > 1
    Display indent(1:2 * depth - 2) buffer(1:posd - 1)
  Else
    Display buffer(1:posd - 1)
  End-if
End-if
.
End program Pretty.

```

プログラム XGFX からの出力

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<po:purchaseOrder xmlns:po="http://www.example.com" orderDate="1999-10-20" orderComment="Hurry, my lawn is going wild!">
  <po:shipTo country="US" name="Alice Smith" street="123 Maple Street" city="Mill Valley" state="CA" zip="90952"/>
  <po:billTo country="US" name="Robert Smith" street="8 Oak Avenue" city="Old Town" state="PA" zip="95819"/>
  <po:items>
    <po:item partNum="872-AA" productName="Lawnmower" quantity="1" USPrice="148.95" shipDate=" " itemComment="Confirm...
  </po:items>
  <po:items>
    <po:item partNum="926-AA" productName="Baby Monitor" quantity="1" USPrice="39.98" shipDate="1999-05-21" itemComme...
  </po:items>
</po:purchaseOrder>

```

関連タスク

387 ページの『第 21 章 XML 入力の処理』

関連参照

XML GENERATE の操作 (COBOL for AIX 言語解説書)

XML 出力の拡張

XML フォーマットで表したい情報が既に DATA DIVISION のグループ項目に存在しているが、1 つ以上の要因のためその項目を使用して XML 文書を直接生成できないおそれがあります。

以下に例を示します。

- 必要データのほかに、項目には、XML 出力文書とは無関係な値を含んでいる従属データ項目が含まれています。
- 必要データ項目の名前が、外部表示には不適當なものであり、プログラマーにしか意味のないものである可能性があります。
- データ定義が必要なデータ型ではありません。再定義 (XML GENERATE ステートメントでは無視されます) のみが適切なフォーマットになっていると思われます。
- 無関係な従属グループ内で必要データ項目があまりに深くネストされています。XML 出力は、階層ではなく、デフォルトで行われるように「平ら」にする必要があります。
- 必要データ項目があまりに多くのコンポーネントに分割されており、収容グループの内容として出力する必要があります。
- グループ項目は必要情報を含んでいますが、順序が正しくありません。

こうした状態を取り扱うことのできるさまざまな方法があります。1 つの手法として考えられるのは、適切な特性を持つデータ項目を新しく定義し、作成した新しいデータ項目の適切なフィールドに必要なデータを移動することです。しかし、この手法は多少面倒な作業で、元のデータ項目と新しいデータ項目の同期を維持するため注意深い保守作業が必要となります。

いくつかの利点のある代替方法として、元のグループ・データ項目の再定義を準備し、その再定義から XML 出力を生成するという方法があります。このためには、元のデータ記述セットを出発点にして、以下の変更を行ってください。

- 基本データ項目の名前を FILLER に変更するか、またはそれらの名前を削除することにより、生成された XML から基本データ項目を除外します。
- 選択された基本項目およびそれらの基本項目を含んでいるグループ項目に、もっと意味のある適切な名前を付けます。
- 不要な中間グループ項目を除去して、階層を平らにします。
- 種々のデータ型を指定して、必要なトリミング動作が行われるようにします。
- 一連の XML GENERATE ステートメントを使用して、異なった出力順序を選択します。

上記の変更を最も安全に行うには、1 つ以上の REPLACE コンパイラ指示ステートメントを伴う元の宣言の別のコピーを使用します。以下に示す例は、その方法を示しています。

419 ページの『例: XML 出力の拡張』

XML 文書を生成する際に、エレメントまたは属性名および値の中にハイフンを含んでいるものがあります。エレメントまたは属性値に含まれるハイフンは

そのままにして、エレメントまたは属性名のハイフンを下線に変換することがあります。以下に示す例は、その方法を示しています。

421 ページの『例: エレメントまたは属性名のハイフンを下線に変換する』

関連参照

XML GENERATE の操作 (COBOL for AIX 言語解説書)

例: XML 出力の拡張

次の例は、どうすれば XML 出力を変更させることができるかを示しています。

以下のデータ構造について考慮してください。構造から生成される XML には、訂正可能ないくつかの問題が含まれています。

```
01 CDR-LIFE-BASE-VALUES-BOX.
  15 CDR-LIFE-BASE-VAL-DATE PIC X(08).
  15 CDR-LIFE-BASE-VALUE-LINE OCCURS 2 TIMES.
    20 CDR-LIFE-BASE-DESC.
      25 CDR-LIFE-BASE-DESC1 PIC X(15).
      25 FILLER PIC X(01).
      25 CDR-LIFE-BASE-LIT PIC X(08).
      25 CDR-LIFE-BASE-DTE PIC X(08).
    20 CDR-LIFE-BASE-PRICE.
      25 CDR-LIFE-BP-SPACE PIC X(02).
      25 CDR-LIFE-BP-DASH PIC X(02).
      25 CDR-LIFE-BP-SPACE1 PIC X(02).
    20 CDR-LIFE-BASE-PRICE-ED REDEFINES
      CDR-LIFE-BASE-PRICE PIC $$$.$$.
    20 CDR-LIFE-BASE-QTY.
      25 CDR-LIFE-QTY-SPACE PIC X(08).
      25 CDR-LIFE-QTY-DASH PIC X(02).
      25 CDR-LIFE-QTY-SPACE1 PIC X(02).
      25 FILLER PIC X(02).
    20 CDR-LIFE-BASE-QTY-ED REDEFINES
      CDR-LIFE-BASE-QTY PIC ZZ,ZZZ,ZZZ.ZZZ.
    20 CDR-LIFE-BASE-VALUE PIC X(15).
    20 CDR-LIFE-BASE-VALUE-ED REDEFINES
      CDR-LIFE-BASE-VALUE
      PIC $(4),$$,$$9.99.
  15 CDR-LIFE-BASE-TOT-VALUE-LINE.
    20 CDR-LIFE-BASE-TOT-VALUE PIC X(15).
```

このデータ構造にいくつかのサンプル値を取り込み、XML をそれから直接生成し、その後プログラム Pretty (413 ページの『例: XML の生成』に示されています) を使用してフォーマット設定すると、結果は次のようになります。

```
<CDR-LIFE-BASE-VALUES-BOX>
  <CDR-LIFE-BASE-VAL-DATE>01/02/03</CDR-LIFE-BASE-VAL-DATE>
  <CDR-LIFE-BASE-VALUE-LINE>
    <CDR-LIFE-BASE-DESC>
      <CDR-LIFE-BASE-DESC1>First</CDR-LIFE-BASE-DESC1>
      <CDR-LIFE-BASE-LIT> </CDR-LIFE-BASE-LIT>
      <CDR-LIFE-BASE-DTE>01/01/01</CDR-LIFE-BASE-DTE>
    </CDR-LIFE-BASE-DESC>
    <CDR-LIFE-BASE-PRICE>
      <CDR-LIFE-BP-SPACE>$2</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>3.</CDR-LIFE-BP-DASH>
      <CDR-LIFE-BP-SPACE1>00</CDR-LIFE-BP-SPACE1>
    </CDR-LIFE-BASE-PRICE>
    <CDR-LIFE-BASE-QTY>
      <CDR-LIFE-QTY-SPACE> 1</CDR-LIFE-QTY-SPACE>
      <CDR-LIFE-QTY-DASH>23</CDR-LIFE-QTY-DASH>
```

```

      <CDR-LIFE-QTY-SPACE1>.0</CDR-LIFE-QTY-SPACE1>
    </CDR-LIFE-BASE-QTY>
    <CDR-LIFE-BASE-VALUE>          $765.00</CDR-LIFE-BASE-VALUE>
  </CDR-LIFE-BASE-VALUE-LINE>
  <CDR-LIFE-BASE-VALUE-LINE>
    <CDR-LIFE-BASE-DESC>
      <CDR-LIFE-BASE-DESC1>Second</CDR-LIFE-BASE-DESC1>
      <CDR-LIFE-BASE-LIT> </CDR-LIFE-BASE-LIT>
      <CDR-LIFE-BASE-DTE>02/02/02</CDR-LIFE-BASE-DTE>
    </CDR-LIFE-BASE-DESC>
    <CDR-LIFE-BASE-PRICE>
      <CDR-LIFE-BP-SPACE>$3</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>4.</CDR-LIFE-BP-DASH>
      <CDR-LIFE-BP-SPACE1>00</CDR-LIFE-BP-SPACE1>
    </CDR-LIFE-BASE-PRICE>
    <CDR-LIFE-BASE-QTY>
      <CDR-LIFE-QTY-SPACE>          2</CDR-LIFE-QTY-SPACE>
      <CDR-LIFE-QTY-DASH>34</CDR-LIFE-QTY-DASH>
      <CDR-LIFE-QTY-SPACE1>.0</CDR-LIFE-QTY-SPACE1>
    </CDR-LIFE-BASE-QTY>
    <CDR-LIFE-BASE-VALUE>          $654.00</CDR-LIFE-BASE-VALUE>
  </CDR-LIFE-BASE-VALUE-LINE>
  <CDR-LIFE-BASE-TOT-VALUE-LINE>
    <CDR-LIFE-BASE-TOT-VALUE>Very high!</CDR-LIFE-BASE-TOT-VALUE>
  </CDR-LIFE-BASE-TOT-VALUE-LINE>
</CDR-LIFE-BASE-VALUES-BOX>

```

生成されたこの XML にはいくつかの問題があります。

- エレメント名が長く、あまり意味のあるものではありません。
- 不要なデータ、例えば、CDR-LIFE-BASE-LIT や CDR-LIFE-BASE-DTE があります。
- 必要データに、不必要な親があります。例えば、CDR-LIFE-BASE-DESC1 には親の CDR-LIFE-BASE-DESC があります。
- 他の必須フィールドが、あまりに多くのサブコンポーネントに分割されています。例えば、CDR-LIFE-BASE-PRICE には、1 つの金額に対して 3 つのサブコンポーネントがあります。

XML 出力のこのような特性は、ストレージを次のように再定義することにより修正できます。

```

1 BaseValues redefines CDR-LIFE-BASE-VALUES-BOX.
2 BaseValueDate pic x(8).
2 BaseValueLine occurs 2 times.
3 Description pic x(15).
3 pic x(9).
3 BaseDate pic x(8).
3 BasePrice pic x(6) justified.
3 BaseQuantity pic x(14) justified.
3 BaseValue pic x(15) justified.
2 TotalValue pic x(15).

```

上記のデータ値の定義のセットから XML を生成してフォーマット設定した結果は、一層便利なものになっています。

```

<BaseValues>
  <BaseValueDate>01/02/03</BaseValueDate>
  <BaseValueLine>
    <Description>First</Description>
    <BaseDate>01/01/01</BaseDate>
    <BasePrice>$23.00</BasePrice>
    <BaseQuantity>123.000</BaseQuantity>
    <BaseValue>$765.00</BaseValue>
  </BaseValueLine>

```

```

<BaseValueLine>
  <Description>Second</Description>
  <BaseDate>02/02/02</BaseDate>
  <BasePrice>$34.00</BasePrice>
  <BaseQuantity>234.000</BaseQuantity>
  <BaseValue>$654.00</BaseValue>
</BaseValueLine>
<TotalValue>Very high!</TotalValue>
</BaseValues>

```

上に示すように、元のデータ定義を直接再定義できます。通常、元の定義を使用するほうがより安全ですが、コンパイラーのテキスト操作機能を使用して、元の定義を適宜に変更することができます。その例を、以下の REPLACE コンパイラー指示ステートメントで示します。この REPLACE ステートメントは複雑に見えますが、元のデータ定義が変更されると、自己保持するという利点があります。

```

replace ==CDR-LIFE-BASE-VALUES-BOX== by
  ==BaseValues redefines CDR-LIFE-BASE-VALUES-BOX==
  ==CDR-LIFE-BASE-VAL-DATE== by ==BaseValueDate==
  ==CDR-LIFE-BASE-VALUE-LINE== by ==BaseValueLine==
  ==20 CDR-LIFE-BASE-DESC.== by ====
  ==CDR-LIFE-BASE-DESC1== by ==Description==
  ==CDR-LIFE-BASE-LIT== by ====
  ==CDR-LIFE-BASE-DTE== by ==BaseDate==
  ==20 CDR-LIFE-BASE-PRICE.== by ====
  ==25 CDR-LIFE-BP-SPACE PIC X(02).== by ====
  ==25 CDR-LIFE-BP-DASH PIC X(02).== by ====
  ==25 CDR-LIFE-BP-SPACE1 PIC X(02).== by ====
  ==CDR-LIFE-BASE-PRICE-ED== by ==BasePrice==
  ==REDEFINES CDR-LIFE-BASE-PRICE PIC $$$.$$.== by
    ==pic x(6) justified.==
  ==20 CDR-LIFE-BASE-QTY.
    25 CDR-LIFE-QTY-SPACE PIC X(08).
    25 CDR-LIFE-QTY-DASH PIC X(02).
    25 CDR-LIFE-QTY-SPACE1 PIC X(02).
    25 FILLER PIC X(02).== by ====
  ==CDR-LIFE-BASE-QTY-ED== by ==BaseQuantity==
  ==REDEFINES CDR-LIFE-BASE-QTY PIC ZZ,ZZZ,ZZZ.ZZZ.== by
    ==pic x(14) justified.==
  ==CDR-LIFE-BASE-VALUE-ED== by ==BaseValue==
  ==20 CDR-LIFE-BASE-VALUE PIC X(15).== by ====
  ==REDEFINES CDR-LIFE-BASE-VALUE PIC $(4),$$$,$$9.99.==
    by ==pic x(15) justified.==
  ==CDR-LIFE-BASE-TOT-VALUE-LINE. 20== by ====
  ==CDR-LIFE-BASE-TOT-VALUE== by ==TotalValue==.

```

元の定義セットの 2 番目のインスタンスが後に続いているこの REPLACE ステートメントの結果は、上に示されているグループ項目 BaseValues の推奨再定義と似ています。この REPLACE ステートメントは、不要な定義を除去し、保持すべき定義の変更を行うさまざまな手法の例を示しています。それぞれの状況に適したいずれかの手法をご利用ください。

関連参照

XML GENERATE の操作 (COBOL for AIX 言語解説書)

REPLACE ステートメント (COBOL for AIX 言語解説書)

例: エレメントまたは属性名のハイフンを下線に変換する

| ハイフンを含んでいるデータ名を持つ項目のあるデータ構造から XML 文書を生
| 成すると、生成された XML にはハイフンを含んでいるエレメントまたは属性名が含

まれることとなります。この例は、エレメントまたは属性の値に含まれるハイフンを変更せずに、エレメントまたは属性の名前に含まれるハイフンを下線に変換する方法を示しています。

```
1 Customer-Record.  
2 Customer-Number pic 9(9).  
2 First-Name      pic x(10).  
2 Last-Name       pic x(20).
```

上記のデータ構造にいくつかのサンプル値を取り込み、XML をそれから生成し、その後プログラム Pretty (413 ページの『例: XML の生成』に示されています) を使用してフォーマット設定すると、結果は次のようになります。

```
<Customer-Record>  
  <Customer-Number>12345</Customer-Number>  
  <First-Name>John</First-Name>  
  <Last-Name>Smith-Jones</Last-Name>  
</Customer-Record>
```

エレメント名はハイフンを含みますが、エレメント Last-Name の内容もハイフンを含んでいます。

この XML 文書がデータ項目 xmldoc の内容であり、charcnt が XML 文書の長さ に設定されていると想定した場合、以下のコードを使用することにより、エレメント名の中のすべてのハイフンを下線に変更し、エレメント値は未変更のままにすることができます。

```
1 xmldoc          pic x(16384).  
1 charcnt        comp-5 pic 9(5).  
1 pos            comp-5 pic 9(5).  
1 tagstate       comp-5 pic 9 value zero.  
1 quotestate     comp-5 pic 9 value zero.  
  
...  
dash-to-underscore.  
perform varying pos from 1 by 1  
  until pos > charcnt  
  if xmldoc(pos:1) = '<'  
    move 1 to tagstate  
  end-if  
  if tagstate = 1  
    if xmldoc(pos:1) = '''  
      if quotestate = 0  
        move 1 to quotestate  
      else  
        move 0 to quotestate  
      end-if  
    end-if  
  end-if  
  if tagstate = 1 and quotestate = 0 and xmldoc(pos:1) = '-'  
    move '_' to xmldoc(pos:1)  
  else  
    if xmldoc(pos:1) = '>'  
      move 0 to tagstate  
    end-if  
  end-if  
end-perform.
```

データ項目 xmldoc 内の改訂された XML 文書には、エレメント名の中にハイフンの代わりに下線があります。ただし、次に示すように、エレメント値に含まれるハイフンは保持されています。

```
| <Customer_Record>  
|   <Customer_Number>12345</Customer_Number>  
|   <First_Name>John</First_Name>  
|   <Last_Name>Smith-Jones</Last_Name>  
| </Customer_Record>
```


第 6 部 オブジェクト指向プログラムの開発

第 23 章 オブジェクト指向プログラムの作成	427
例: 口座	428
サブクラス	429
クラスの定義	430
クラス定義用の CLASS-ID 段落	432
クラス定義用の REPOSITORY 段落	433
例: 外部クラス名および Java パッケージ	434
クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION	434
例: クラスの定義	435
クラス・インスタンス・メソッドの定義	436
クラス・インスタンス・メソッド定義用の METHOD-ID 段落	437
クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION	438
クラス・インスタンス・メソッド定義用の DATA DIVISION	438
クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION	439
インスタンス・メソッドのオーバーライド	440
インスタンス・メソッドの多重定義	441
属性 (get および set) メソッドのコーディング	442
例: ゲット・メソッドのコーディング	443
例: メソッドの定義	443
Account クラス	443
Check クラス	445
クライアントの定義	445
クライアント定義用の REPOSITORY 段落	446
クライアント定義用の DATA DIVISION	447
LOCAL-STORAGE または WORKING-STORAGE の選択	448
オブジェクト参照の比較および設定	449
メソッドの呼び出し (INVOKE)	450
引数の引き渡し用の USING 句	451
例: COBOL クライアントからの規格合致オブ ジェクト参照の引数の引き渡し	452
戻り値の取得用の RETURNING 句	454
オーバーライドされたスーパークラス・メソ ッドの呼び出し	454
クラスのインスタンスの作成および初期化	455
Java クラスのインスタンス化	455
COBOL クラスのインスタンス化	456
クラスのインスタンスの解放	457
例: クライアントの定義	457
サブクラスの定義	458
サブクラス定義用の CLASS-ID 段落	459
サブクラス定義用の REPOSITORY 段落	460
サブクラス・インスタンス・データ定義用の WORKING-STORAGE SECTION	460
サブクラス・インスタンス・メソッドの定義	461
例: サブクラスの定義 (メソッドに関して)	461
CheckingAccount クラス (Account のサブクラ ス)	461
ファクトリー・セクションの定義	462
ファクトリー・データ定義用の WORKING-STORAGE SECTION	463
ファクトリー・メソッドの定義	464
ファクトリー・メソッドまたは静的メソッド の隠蔽	466
ファクトリー・メソッドまたは静的メソッド の呼び出し	466
例: ファクトリーの定義 (メソッドに関して)	467
Account クラス	468
CheckingAccount クラス (Account のサブクラ ス)	469
Check クラス	471
TestAccounts クライアント・プログラム	471
TestAccounts クライアント・プログラムが生 成する出力	472
プロシーチャー指向 COBOL プログラムのラッピ ング	473
OO アプリケーションの構造化	473
例: java コマンドを使用して実行される COBOL アプリケーション	474
メッセージの表示	474
入カストリングのエコー	474
第 24 章 Java メソッドとの通信	477
JNI サービスへのアクセス	477
Java 例外の処理	479
例: Java 例外の処理	479
ローカル参照とグローバル参照の管理	480
ローカル参照の削除、保管、および解放	481
Java アクセス制御	482
Java とのデータ共有	482
COBOL および Java での相互運用可能なデータ 型のコーディング	483
Java 用の配列およびストリングの宣言	484
Java 配列の取り扱い	485
例: Java int 配列の処理	487
Java ストリングの取り扱い	488

第 23 章 オブジェクト指向プログラムの作成

オブジェクト指向 (OO) プログラムを書く際には、必要とするクラス、およびクラスが作業を行うのに必要なメソッドとデータを決定する必要があります。

OO プログラムは、オブジェクト (状態と動作をカプセル化するエンティティー) ならびにオブジェクトのクラス、メソッド、およびデータに基づいています。クラスは、オブジェクトの状態および機能を定義するテンプレートです。通常、プログラムは、あるクラスの複数のオブジェクト・インスタンス (または単にインスタンス)、つまりそのクラスのメンバーである複数のオブジェクトを作成し、それを扱う仕事をします。それぞれのインスタンスの状態はインスタンス・データと呼ばれるデータに保管されます。各インスタンスの機能は、インスタンス・メソッドと呼ばれます。クラスでは、そのクラスのすべてのインスタンスが共用するデータ (ファクトリーまたは静的データと呼ばれる)、およびいずれのオブジェクト・インスタンスとも無関係にサポートされるメソッド (ファクトリーまたは静的メソッドと呼ばれる) を定義できます。

COBOL for AIX を使用して、以下のことを行うことができます。

- メソッドとデータを COBOL でインプリメントした状態で、クラスを定義する。
- Java および COBOL クラスのインスタンスを作成する。
- Java および COBOL オブジェクトにメソッドを呼び出す。
- Java クラスまたはほかの COBOL クラスから継承するクラスを書き込む。
- 多重定義メソッドを定義して呼び出す。

COBOL for AIX プログラムで、Java Native Interface (JNI) が提供するサービスを呼び出して、COBOL 言語で直接使用可能な基本 OO 機能に加えて、Java 指向機能を取得できます。

COBOL for AIX クラスでは、CALL ステートメントをコーディングして、プロシージャ型 COBOL プログラムとインターフェースを取ることができます。したがって、COBOL クラス定義構文は、プロシージャ型 COBOL ロジックのラッパー・クラスを書き込むために特に役立ち、Java から既存の COBOL コードにアクセスすることを可能にします。

Java コードは、COBOL クラスのインスタンスを作成したり、これらのクラスのメソッドを呼び出したり、COBOL クラスを拡張したりすることができます。

制約事項:

- COBOL クラス定義およびメソッドは、EXEC SQL ステートメントを含むことができず、SQL コンパイラー・オプションを使用してコンパイルすることもできません。
- COBOL クラス定義およびメソッドは、EXEC CICS ステートメントを含むことができず、CICS 環境で実行することもできません。CICS コンパイラー・オプションを使用してコンパイルすることができません。

428 ページの『例: 口座』

関連概念

261 ページの『Java SDK サポート』

関連タスク

430 ページの『クラスの定義』

436 ページの『クラス・インスタンス・メソッドの定義』

445 ページの『クライアントの定義』

458 ページの『サブクラスの定義』

462 ページの『ファクトリー・セクションの定義』

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

関連参照

The Java Language Specification

例: 口座

銀行にカスタマーが口座を開設し、その口座で預金や引き出しを行うときの例を見てみましょう。口座は、`Account` という名前の汎用クラスで表します。多数のカスタマーが存在します。したがって、`Account` クラスの複数インスタンスが同時に存在すると考えることができます。

必要とするクラスを判別したら、次のステップでは、それらのクラスがそれぞれの作業を実行するために必要なメソッドを判別します。`Account` クラスは以下のサービスを提供する必要があります。

- 口座を開設する。
- 現在の収支を取る。
- 口座に預金する。
- 口座から預金を引き出す。
- 口座状況を報告する。

`Account` クラスの以下のメソッドは、上記の要件を満たします。

init 口座を開設し、それに口座番号を割り当てます。

getBalance

口座の現在の収支を戻します。

credit 指定の金額を口座に預金します。

debit 指定の金額を口座から引き出します。

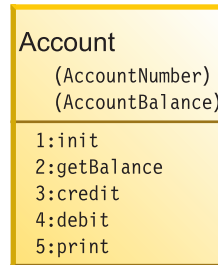
print 口座番号と勘定残高を表示します。

`Account` クラスとそのメソッドを設計すると、クラスはいくつかのインスタンス・データを保持する必要があることがわかります。一般に、`Account` オブジェクトには次のようなインスタンス・データが必要です。

- 口座番号
- 勘定残高
- カスタマー情報: 名前、住所、自宅の電話番号、勤務先電話番号、社会保障番号など

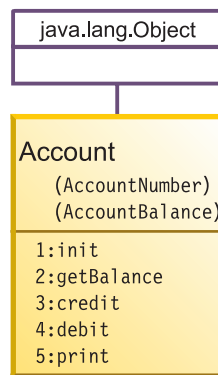
ただし、上記の例を単純化するため、口座番号と勘定残高は、Account クラスが必要とする唯一のインスタンス・データであると想定します。

クラスやメソッドを設計するとき、ダイアグラムは役に立ちます。次のダイアグラムで、Account クラスの設計における最初の試みを示します。



ダイアグラムにおいて、括弧内のワードはインスタンス・データの名前です。番号とコロンに続くワードは、インスタンス・メソッドの名前です。

以下の構造は、クラスの相互関係を示しており、継承の階層と呼ばれています。Account クラスはクラス `java.lang.Object` から直接継承します。



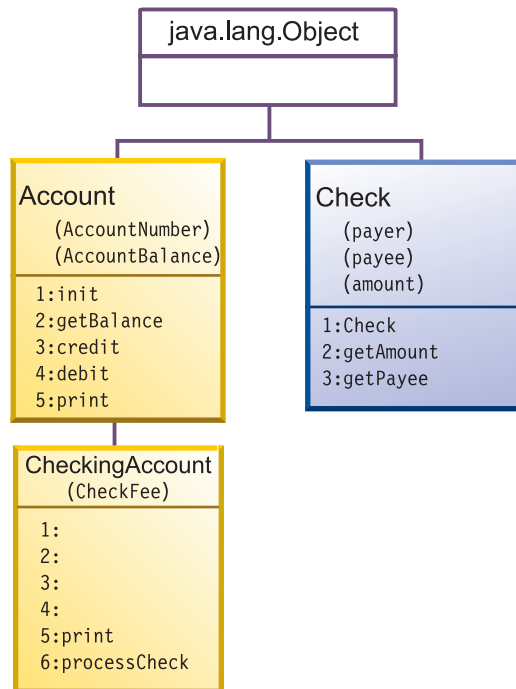
サブクラス

上記の口座の例において、Account は汎用クラスです。ただし、銀行は、当座預金、普通預金、住宅ローンなど、多くの種類の口座を用意することができます。これらの口座のすべてには、口座の一般的特性がある一方で、すべての種類の口座に必ずしも共有されない追加特性が含まれている場合があります。

例えば、CheckingAccount クラスには、すべての口座が持つ口座番号や勘定残高に加えて、口座に書き込まれる各当座に適用される当座手数料がある場合があります。また、CheckingAccount クラスには、当座を処理する (すなわち、金額を読み取る、支払人の借方に記入する、受取人の貸方に記入するなど) メソッドも必要です。したがって、CheckingAccount を Account のサブクラスとして定義し、そのサブクラスが必要とする追加のインスタンス・データやインスタンス・メソッドをそのサブクラスで定義することは意味があります。

CheckingAccount クラスを設計すると、当座をモデル化するクラスの必要性があることに気が付きます。クラス Check のインスタンスは、少なくとも、支払人、受取人、および当座の金額のインスタンス・データを必要とします。

実際のオブジェクト指向のアカウント・システムでは多くの追加クラス (ならびにデータベースおよびトランザクション処理ロジック) を設計する必要があるでしょうが、ここでは例を単純化するために省略されています。継承更新ダイアグラムを以下に示します。



番号とコロンの後にメソッド名が記されていないものは、その番号のメソッドがスーパークラスから継承されていることを示します。

多重継承: OO COBOL アプリケーションでは多重継承を使用できません。定義するすべてのクラスは、厳密に 1 つの親を持つ必要があります。java.lang.Object は、すべての継承の階層のルートになければなりません。したがって、OO COBOL アプリケーション内で定義されるオブジェクト指向システムのクラス構造はツリー状です。

443 ページの『例: メソッドの定義』

関連タスク

『クラスの定義』

436 ページの『クラス・インスタンス・メソッドの定義』

458 ページの『サブクラスの定義』

クラスの定義

COBOL クラス定義は、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION、その後にオプションのファクトリー定義とオプションのオブジェクト定義、さらにその後に END CLASS マーカーが続く構成となっています。

表 48. クラス定義の構成

セクション	目的	構文
IDENTIFICATION DIVISION (必須)	クラスの名前。継承情報を提供する。	432 ページの『クラス定義用の CLASS-ID 段落』 (必須) AUTHOR 段落 (オプション) INSTALLATION 段落 (オプション) DATE-WRITTEN 段落 (オプション) DATE-COMPILED 段落 (オプション)
ENVIRONMENT DIVISION (必須)	コンピューター環境を記述する。クラス定義内で使用されるクラス名を、コンパイル単位の外側で判明している、対応する外部クラス名に関連付ける。	CONFIGURATION SECTION (必須) 433 ページの『クラス定義用の REPOSITORY 段落』 (必須) SOURCE-COMPUTER 段落 (オプション) OBJECT-COMPUTER 段落 (オプション) SPECIAL-NAMES 段落 (オプション)
ファクトリー定義 (オプション)	クラスのすべてのインスタンスが共有するデータとオブジェクト・インスタンスとは別々にサポートされるメソッドを定義する。	IDENTIFICATION DIVISION. FACTORY. DATA DIVISION. WORKING-STORAGE SECTION. * (Factory data here) PROCEDURE DIVISION. * (Factory methods here) END FACTORY.
オブジェクト定義 (オプション)	インスタンス・データとインスタンス・メソッドを定義する。	IDENTIFICATION DIVISION. OBJECT. DATA DIVISION. WORKING-STORAGE SECTION. * (Instance data here) PROCEDURE DIVISION. * (Instance methods here) END OBJECT.

SOURCE-COMPUTER、OBJECT-COMPUTER、または SPECIAL-NAMES 段落をクラス CONFIGURATION SECTION に指定すると、それらの段落は、そのクラスが導入するすべてのメソッドを含む、クラス定義全体に適用されます。

クラス CONFIGURATION SECTION は、プログラム CONFIGURATION SECTION と同じ記入項目で構成されています。ただし、クラス CONFIGURATION SECTION には INPUT-OUTPUT SECTION を含めることはできません。INPUT-OUTPUT SECTION の定義は、クラス・レベルでその定義を行うのではなく、それを必要とする個々のメソッドにおいてのみ行います。

上記で説明したように、インスタンス・データとメソッドの定義は、そのクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION において、行います。個別のオブジェクト・インスタンスにではなく、クラス自体に関連付けるデータとメソッドを必要とするクラスに、クラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

各 COBOL クラス定義は、別々のソース・ファイルになければなりません。

435 ページの『例: クラスの定義』

関連タスク

434 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE

SECTION』

436 ページの『クラス・インスタンス・メソッドの定義』

458 ページの『サブクラスの定義』

462 ページの『ファクトリー・セクションの定義』

7 ページの『コンピューター環境の記述』

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

関連参照

COBOL クラス定義構成 (COBOL for AIX 言語解説書)

クラス定義用の CLASS-ID 段落

クラスを指定し、それに継承情報を提供するには、IDENTIFICATION DIVISION の CLASS-ID 段落を使用してください。

Identification Division. 必須
Class-id. Account inherits Base. 必須

以下のクラスを識別するには、CLASS-ID 段落を使用してください。

- 定義中のクラス (上記の例の Account)。
- 定義中のクラスがその特性を継承する元の即時スーパークラス。スーパークラスは、Java または COBOL でインプリメントされます。

上記の例において、inherits Base では、Account クラスは、クラス定義内で Base として認識されているクラスからメソッドとデータを継承することを示します。オブジェクト指向 COBOL プログラムにおいて、名前 Base は java.lang.Object を参照するために使用することをお勧めします。

クラス名では 1 バイト文字を使用する必要があり、クラス名は COBOL ユーザー定義語の通常の形成規則に準拠している必要があります。

ENVIRONMENT DIVISION の CONFIGURATION SECTION で REPOSITORY 段落を使用して、スーパークラス名 (例の Base) を外部に判明しているスーパークラス名 (Base の場合の java.lang.Object) に関連付けます。また、オプションとして、定義中のクラスの名前 (例の Account) を REPOSITORY 段落に指定し、その対応する外部クラス名にそれを関連付けることもできます。

すべてのクラスを java.lang.Object クラスから直接的または間接的に引き出さなければなりません。

関連タスク

433 ページの『クラス定義用の REPOSITORY 段落』

関連参照

CLASS-ID 段落 (COBOL for AIX 言語解説書)

ユーザー定義語 (COBOL for AIX 言語解説書)

クラス定義用の REPOSITORY 段落

指定された語をクラス定義内で使用するときその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名(コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

外部クラス名は大/小文字が区別されます。したがって、Java 形成規則に準拠しなければなりません。例えば、Account クラス定義において、以下のようにコーディングします。

```
Environment Division.           必須
Configuration Section.         必須
Repository.                     必須
    Class Base is "java.lang.Object" 必須
    Class Account is "Account".     オプション
```

REPOSITORY 段落記入項目では、クラス定義内で Base および Account として参照されるクラスの外部クラス名は、それぞれ、java.lang.Object および Account であることが示されます。

REPOSITORY 段落では、クラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。以下に例を示します。

- ベース
- 定義中のクラスが継承する元のスーパークラス
- クラス定義内のメソッドで参照するクラス

REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定しなければなりません。Java パッケージの一部である参照クラスごとに外部クラス名も指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド(.)が付き、続いてその後に Java クラスの単純名が付きます。例えば、Object クラスは java.lang パッケージの一部です。したがって、上記に示したように、その外部名を java.lang.Object として指定します。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形成規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名は以下の方法でクラス名から作成されます。

- クラス名は大文字に変換されます。
- 各ハイフンはゼロに変更されます。
- 数字の場合、最初の文字は以下のように変更されます。
 - 1 から 9 は A から I に変更されます。
 - 0 は J に変換されます。

外部名は英大/小文字混合で名前付けされます。したがって、上記の例で、クラス Account は外部的には Account (英大/小文字混合) として認識されます。

オプションとして、定義中のクラス(上記の例の Account) の記入項目を REPOSITORY 段落に組み込むことができます。外部クラス名が非 COBOL 文字を含

む場合は定義中のクラスの記入項目を組み込む必要があり、クラスが Java パッケージの一部となる場合には、完全パッケージ修飾クラス名を指定する必要があります。

『例: 外部クラス名および Java パッケージ』

関連タスク

484 ページの『Java 用の配列およびストリングの宣言』

関連参照

REPOSITORY 段落 (COBOL for AIX 言語解説書)

The Java Language Specification (Identifiers)

The Java Language Specification (Packages)

例: 外部クラス名および Java パッケージ

次の例では、REPOSITORY 段落内の記入項目から外部クラス名を決定する方法を説明します。

```
Environment division.  
Configuration section.  
Repository.  
    Class Employee is "com.acme.Employee"  
    Class JavaException is "java.lang.Exception"  
    Class Orders.
```

次の表には、ローカル・クラス名 (クラス定義内で使用されるクラス名)、そのクラスを含む Java パッケージ、および関連付けられた外部クラス名が記述されています。

ローカル・クラス名	Java パッケージ	外部クラス名
Employee	com.acme	com.acme.Employee
JavaException	java.lang	java.lang.Exception
Orders	(名前付けなし)	ORDERS

外部クラス名 (REPOSITORY 段落記入項目内のクラス名の後の名前とオプションの IS) は、パッケージ (ある場合) の完全修飾名から構成され、後にピリオドが付き、続いてその後にクラスの単純名が付きます。

関連タスク

433 ページの『クラス定義用の REPOSITORY 段落』

関連参照

REPOSITORY 段落 (COBOL for AIX 言語解説書)

クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION

COBOL クラスが必要とする インスタンス・データ、つまりクラスの各インスタンスに割り当てられるデータを記述するには、OBJECT 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。

IDENTIFICATION DIVISION 宣言の直前に入れる必要がある OBJECT キーワードは、クラスのインスタンス・データおよびインスタンス・メソッドの定義の開始を示します。例えば、Account クラスのインスタンス・データの定義は、以下のようになります。

```
Identification division.
Object.
  Data division.
  Working-storage section.
  01 AccountNumber pic 9(6).
  01 AccountBalance pic S9(9) value zero.
  .
  .
  .
End Object.
```

インスタンス・データは、オブジェクト・インスタンスが作成されるときに割り振られ、Java ランタイムによるインスタンスのガーベッジ・コレクションが行われるまで存在します。

上記に示すように、単純インスタンス・データの初期化は、VALUE 節を使用して行うことができます。より複雑なインスタンス・データの初期化は、カスタマイズしたメソッドをコーディングし、クラスのインスタンスを作成して初期化して行うことができます。

COBOL インスタンス・データは、Java private 非静的メンバー・データと同等です。他のクラスまたはサブクラス (同じクラス内のファクトリー・メソッドがあればそのメソッドも) は、COBOL インスタンス・データを直接参照することはできません。インスタンス・データは、OBJECT 段落で定義するすべてのインスタンス・メソッドにグローバルです。OBJECT 段落の外側からインスタンス・データにアクセス可能にしたい場合には、アクセスを可能にするために属性 (get または set) インスタンス・メソッドを定義します。

インスタンス・データ宣言のための WORKING-STORAGE SECTION の構文は、プログラムにおける場合と一般的に同じですが、次のような例外があります。

- EXTERNAL 属性を使用することはできません。
- GLOBAL 属性を使用できますが、効力はありません。

関連タスク

455 ページの『クラスのインスタンスの作成および初期化』

457 ページの『クラスのインスタンスの解放』

464 ページの『ファクトリー・メソッドの定義』

442 ページの『属性 (get および set) メソッドのコーディング』

例: クラスの定義

次の例では、Account クラスの定義での最初の試みを示します。ただし、メソッド定義は除きます。

```
cb1 thread,pgmname(longmixed)
Identification Division.
Class-id. Account inherits Base.
Environment Division.
Configuration section.
Repository.
  Class Base is "java.lang.Object"
  Class Account is "Account".
```

```

*
Identification division.
Object.
Data division.
Working-storage section.
01 AccountNumber pic 9(6).
01 AccountBalance pic S9(9) value zero.
*
Procedure Division.
*
* (Instance method definitions here)
*
End Object.
*
End class Account.

```

関連タスク

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

445 ページの『クライアントの定義』

クラス・インスタンス・メソッドの定義

クラス定義の OBJECT 段落の PROCEDURE DIVISION に COBOL インスタンス・メソッドを定義します。インスタンス・メソッドは、クラスのそれぞれのオブジェクト・インスタンスごとにサポートされる操作を定義します。

COBOL インスタンス・メソッド定義は、4 つの部 (COBOL プログラムに類似) とその後の END METHOD マーカーから構成されます。

表 49. インスタンス・メソッド定義の構成

除算	目的	構文
IDENTIFICATION (必須)	メソッドの名前を指定する。	437 ページの『クラス・インスタンス・メソッド定義用の METHOD-ID 段落』 (必須) AUTHOR 段落 (オプション) INSTALLATION 段落 (オプション) DATE-WRITTEN 段落 (オプション) DATE-COMPILED 段落 (オプション)
ENVIRONMENT (オプション)	メソッド内で使用されるファイル名を、オペレーティング・システムに認識された、対応するファイル名に関連付ける。	438 ページの『クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION』 (オプション)
DATA (オプション)	外部ファイルを定義する。データのコピーを割り振る。	438 ページの『クラス・インスタンス・メソッド定義用の DATA DIVISION』 (オプション)
PROCEDURE (オプション)	メソッドによって提供されるサービスを完了するために実行可能ステートメントをコーディングする。	439 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』 (オプション)

定義: メソッドのシグニチャーは、メソッドの名前およびその仮パラメーターの数と型から構成されています。(COBOL メソッドの仮パラメーターの定義は、そのメソッドの PROCEDURE DIVISION ヘッダーの USING 句で行います。)

クラス定義内では、各メソッド名を固有にする必要はありませんが、各メソッドに固有のシグニチャーを与える必要があります。(メソッドに同じ名前を与えても、別々のシグニチャーを与えると、メソッドを多重定義 することになります。)

COBOL インスタンス・メソッドは Java public 非静的メソッドと同じです。

443 ページの『例: メソッドの定義』

関連タスク

439 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

441 ページの『インスタンス・メソッドの多重定義』

440 ページの『インスタンス・メソッドのオーバーライド』

450 ページの『メソッドの呼び出し (INVOKE)』

461 ページの『サブクラス・インスタンス・メソッドの定義』

464 ページの『ファクトリー・メソッドの定義』

クラス・インスタンス・メソッド定義用の METHOD-ID 段落

インスタンス・メソッドを指定するには、METHOD-ID 段落を使用してください。IDENTIFICATION DIVISION 宣言とともに、METHOD-ID 段落の直前に置いて、メソッド定義の開始を表します。

例えば、Account クラス内の credit メソッドの定義は、以下のように開始します。

```
Identification Division.  
Method-id. "credit".
```

メソッド名を英数字または各国語リテラルとしてコーディングします。メソッド名は、大/小文字を区別して処理されるため、Java メソッド名の形成規則に準拠する必要があります。

他の Java または COBOL メソッドもしくはプログラム (すなわち、クライアント) は、メソッド名を使用してメソッドを呼び出します。

関連タスク

450 ページの『メソッドの呼び出し (INVOKE)』

189 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

The Java Language Specification (Meaning of method names)

The Java Language Specification (Identifiers)

METHOD-ID 段落 (*COBOL for AIX* 言語解説書)

クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION

インスタンス・メソッドの ENVIRONMENT DIVISION は、1 つのセクション INPUT-OUTPUT SECTION のみを持つことができます。このセクションは、メソッド定義で使用したファイル名をオペレーティング・システムに認識された、対応するファイル名に関連付けます。

例えば、情報をファイルから読み取ったメソッドを Account クラスが定義した場合には、その Account クラスは、以下のようにコーディングされる INPUT-OUTPUT SECTION を持つと考えられます。

```
Environment Division.  
Input-Output Section.  
File-Control.  
    Select account-file Assign AcctFile.
```

メソッドの INPUT-OUTPUT SECTION の構文は、プログラムの INPUT-OUTPUT SECTION の構文と同じです。

関連タスク

7 ページの『コンピューター環境の記述』

関連参照

INPUT-OUTPUT SECTION (COBOL for AIX 言語解説書)

クラス・インスタンス・メソッド定義用の DATA DIVISION

インスタンス・メソッドの DATA DIVISION は、4 つのセクション (FILE SECTION、LOCAL-STORAGE SECTION、WORKING-STORAGE SECTION、および LINKAGE SECTION) の任意のもので構成されます。

FILE SECTION

メソッド FILE SECTION では EXTERNAL ファイルしか定義できないことを除けば、プログラム FILE SECTION と同じです。

LOCAL-STORAGE SECTION

メソッドの呼び出しごとに、LOCAL-STORAGE データの別個のコピーを割り振り、そのメソッドからの戻り時に解放します。メソッド LOCAL-STORAGE SECTION は、プログラム LOCAL-STORAGE SECTION に類似しています。

データ項目上の VALUE 節を指定すると、メソッドの呼び出しのたびに、項目はその値に初期化されます。

WORKING-STORAGE SECTION

WORKING-STORAGE データの単一コピーが割り振られます。データは、実行単位が終了するまで、最後に使われた状態で持続します。メソッドの呼び出しのたびに、呼び出しているオブジェクトまたはスレッドに関係なく、データの同じ単一コピーを使用します。メソッド WORKING-STORAGE SECTION は、プログラム WORKING-STORAGE SECTION に類似しています。

データ項目上の VALUE 節を指定すると、メソッドの最初の呼び出しのときに、項目はその値に初期化されます。データ項目に対する EXTERNAL 節を指定することができます。

LINKAGE SECTION

プログラム LINKAGE SECTION と同じです。

インスタンス・メソッドの DATA DIVISION および OBJECT 段落の DATA DIVISION の両方において、データ項目を同じ名前前で定義した場合、そのデータ名に対するメソッド内の参照は、そのメソッド・データ項目だけを参照します。メソッド DATA DIVISION が優先します。

関連タスク

12 ページの『データの記述』

527 ページの『EXTERNAL 節によるデータの共用』

関連参照

DATA DIVISION 概要 (COBOL for AIX 言語解説書)

クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION

インスタンス・メソッドが提供するサービスをインプリメントするための実行可能ステートメントを、インスタンス・メソッドの PROCEDURE DIVISION にコーディングしてください。

プログラムの PROCEDURE DIVISION でコーディングできるメソッドの PROCEDURE DIVISION において、ほとんどの COBOL ステートメントをコーディングすることができます。ただし、メソッドで以下のステートメントをコーディングすることはできません。

- ENTRY
- EXIT PROGRAM
- 標準 COBOL 85 の以下の古くなったエレメント:
 - ALTER
 - プロシージャ名が指定されていない GOTO
 - SEGMENT-LIMIT
 - USE FOR DEBUGGING

インスタンス・メソッドで EXIT METHOD または GOBACK ステートメントをコーディングして、呼び出し側のクライアントに制御を戻すことができます。両方のステートメントに同じ効果があります。メソッドが呼び出されるときに RETURNING 句が指定されていると、EXIT METHOD または GOBACK ステートメントは、呼び出し側のクライアントにデータの値を戻します。

各メソッドの PROCEDURE DIVISION では、暗黙の EXIT METHOD が最後のステートメントとして生成されます。

メソッドで STOP RUN を指定することができます。この指定を行うと、実行単位内で実行しているすべてのスレッドを含め、実行単位全体が終了します。

メソッド定義の終了は、END METHOD マーカーを使用して行う必要があります。例えば、次のステートメントは credit メソッドの終わりを示します。

End method "credit".

渡された引数の取得のための USING 句: メソッドの PROCEDURE DIVISION ヘッダーの USING 句に、メソッドの仮パラメーター (ある場合) を指定してください。引数が BY VALUE で渡される指定をしなければなりません。各パラメーターは、メソッドの LINKAGE SECTION で、レベル 01 またはレベル 77 項目として定義します。各パラメーターのデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

値を返すための RETURNING 句: メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句に、メソッドの結果として戻されるデータ項目 (ある場合) を指定してください。データ項目は、メソッドの LINKAGE SECTION で、レベル 01 またはレベル 77 項目として定義します。戻り値のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
『インスタンス・メソッドのオーバーライド』

441 ページの『インスタンス・メソッドの多重定義』

449 ページの『オブジェクト参照の比較および設定』

450 ページの『メソッドの呼び出し (INVOKE)』

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

関連参照

311 ページの『THREAD』

手続き部ヘッダー (COBOL for AIX 言語解説書)

インスタンス・メソッドのオーバーライド

サブクラスで定義されたインスタンス・メソッドは、そのサブクラスで利用できるのであれば (これら 2 つのメソッドが同じシグニチャーを持っている場合)、継承されたインスタンス・メソッドをオーバーライドすると言います。

スーパークラス・インスタンス・メソッド `m1` を COBOL サブクラスでオーバーライドするには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでインスタンス・メソッド `m1` を定義します。(スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラスのインスタンスで `m1` を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

例えば、Account クラスは、その LINKAGE SECTION および PROCEDURE DIVISION ヘッダーが以下のような、メソッド `debit` を定義します。

```
Linkage section.  
01 inDebit    pic S9(9) binary.  
Procedure Division using by value inDebit.
```


CheckingAccount サブクラスを定義し、Account スーパークラスで定義された debit メソッドをオーバーライドする debit メソッドをそれに持たず場合には、pic S9(9) binary として指定された入力パラメーターを必ず 1 つ持つサブクラス・メソッドを定義します。クライアントが、CheckingAccount インスタンスへのオブジェクト参照を使用して、debit を呼び出すと、CheckingAccount debit メソッド (Account スーパークラス内の debit メソッドではなく) が呼び出されます。

メソッド戻り値の有無および PROCEDURE DIVISION RETURNING 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・インスタンス・メソッドとオーバーライドしたスーパークラス・インスタンス・メソッドにおいて同一でなければなりません。

インスタンス・メソッドは、COBOL スーパークラスのファクトリー・メソッドをオーバーライドしてはならないし、Java スーパークラスの静的メソッドをオーバーライドすることもできません。

443 ページの『例: メソッドの定義』

関連タスク

439 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

450 ページの『メソッドの呼び出し (INVOKE)』

454 ページの『オーバーライドされたスーパークラス・メソッドの呼び出し』

458 ページの『サブクラスの定義』

466 ページの『ファクトリー・メソッドまたは静的メソッドの隠蔽』

関連参照

The Java Language Specification (Inheritance, overriding, and hiding)

インスタンス・メソッドの多重定義

クラスでサポートされる 2 つのメソッド (クラスで定義されているか、スーパークラスから継承されたかにかかわらず) は、それらが同じ名前を持っており、シグニチャーが異なる場合には、多重定義されていると言います。

例えば、別々の一組のパラメーターを使用してデータを初期化するために、クライアントが別々の版のメソッドを呼び出せるようにするときは、メソッドを多重定義します。

メソッドを多重定義するには、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数や型が、同じクラスでサポートされる同一の名前のメソッドと異なるメソッドを定義します。例えば、Account クラスは、必ず 1 つの仮パラメーターを持つインスタンス・メソッド init を定義します。init メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下のようになります。

```
Linkage section.  
01 inAccountNumber pic S9(9) binary.  
Procedure Division using by value inAccountNumber.
```

クライアントはこのメソッドを呼び出し、inAccountNumber のデータ型と一致する引数を必ず 1 つ渡して、指定の口座番号 (およびデフォルトの勘定残高ゼロ) で Account インスタンスを初期化します。

ただし、Account クラスでは、例えば、開始の勘定残高も指定できる別の仮パラメーターを持つ、2 番目のインスタンス・メソッド `init` を定義することができます。この `init` メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下ようになります。

```
Linkage section.  
01 inAccountNumber pic S9(9) binary.  
01 inBalance       pic S9(9) binary.  
Procedure Division using by value inAccountNumber  
                             inBalance.
```

クライアントは、目的のメソッドのシグニチャーと一致する引数を渡して、いずれかの `init` メソッドを呼び出すことができます。

メソッド戻り値の有無は、多重定義したメソッドと共通している必要はありません。また、PROCEDURE DIVISION RETURNING 句 (ある場合) で指定する戻り値のデータ型は、多重定義したメソッドと同一である必要はありません。

ファクトリー・メソッドの多重定義は、インスタンス・メソッドを多重定義する場合とまったく同じ方法で行うことができます。

多重定義メソッドの定義および多重定義メソッドの呼び出しの解決に関する規則は、対応する Java 規則に基づきます。

関連タスク

450 ページの『メソッドの呼び出し (INVOKE)』

464 ページの『ファクトリー・メソッドの定義』

関連参照

The Java Language Specification (Overloading)

属性 (get および set) メソッドのコーディング

X のための accessor (`get`) メソッドおよび mutator (`set`) メソッドをコーディングして X を定義するクラスの外側から、インスタンス変数 X へのアクセスを提供することができます。

COBOL のインスタンス変数は、*private* です。インスタンス変数を定義するクラスは、インスタンス変数を完全にカプセル化します。したがって、直接アクセスできるのは、同じ OBJECT 段落で定義するインスタンス・メソッドだけです。通常、優れた設計のオブジェクト指向アプリケーションは、クラスの外側からインスタンス変数にアクセスする必要はありません。

`public` インスタンス変数の概念は、Java および他のオブジェクト指向言語で定義されており、クラス属性の概念は CORBA で定義されていますが、どちらの概念も COBOL には直接サポートされていません。(CORBA 属性は、変数が読み取り専用でない場合に、変数の値にアクセスするための自動生成 `get` メソッドと変数の値を変更するための自動生成 `set` メソッドを持つ、インスタンス変数です。)

443 ページの『例: ゲット・メソッドのコーディング』

関連タスク

434 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

17 ページの『データの処理』

例: ゲット・メソッドのコーディング

次の例は、インスタンス変数 `AccountBalance` の値をクライアントに戻すインスタンス・メソッド `getBalance` の、`Account` クラスにおける定義を示しています。`getBalance` および `AccountBalance` は、`Account` クラス定義の `OBJECT` 段落で定義されます。

```
Identification Division.
Class-id. Account inherits Base.
* (ENVIRONMENT DIVISION not shown)
* (FACTORY paragraph not shown)
*
Identification division.
Object.
Data division.
Working-storage section.
01 AccountBalance pic S9(9) value zero.
* (Other instance data not shown)
*
Procedure Division.
*
Identification Division.
Method-id. "getBalance".
Data division.
Linkage section.
01 outBalance pic S9(9) binary.
*
Procedure Division returning outBalance.
Move AccountBalance to outBalance.
End method "getBalance".
*
* (Other instance methods not shown)
End Object.
*
End class Account.
```

例: メソッドの定義

次の例は、直前の例に、`Account` クラスのインスタンス・メソッド定義を追加し、`Java Check` クラスの定義を示します。

(直前の例とは、435 ページの『例: クラスの定義』のことです。)

Account クラス

```
cb1 thread,pgmname(longmixed)
Identification Division.
Class-id. Account inherits Base.
Environment Division.
Configuration section.
Repository.
Class Base is "java.lang.Object"
Class Account is "Account".
*
* (FACTORY paragraph not shown)
*
Identification division.
```

```

Object.
  Data division.
  Working-storage section.
  01 AccountNumber pic 9(6).
  01 AccountBalance pic S9(9) value zero.
*
  Procedure Division.
*
*   init method to initialize the account:
  Identification Division.
  Method-id. "init".
  Data division.
  Linkage section.
  01 inAccountNumber pic S9(9) binary.
  Procedure Division using by value inAccountNumber.
    Move inAccountNumber to AccountNumber.
  End method "init".
*
*   getBalance method to return the account balance:
  Identification Division.
  Method-id. "getBalance".
  Data division.
  Linkage section.
  01 outBalance pic S9(9) binary.
  Procedure Division returning outBalance.
    Move AccountBalance to outBalance.
  End method "getBalance".
*
*   credit method to deposit to the account:
  Identification Division.
  Method-id. "credit".
  Data division.
  Linkage section.
  01 inCredit pic S9(9) binary.
  Procedure Division using by value inCredit.
    Add inCredit to AccountBalance.
  End method "credit".
*
*   debit method to withdraw from the account:
  Identification Division.
  Method-id. "debit".
  Data division.
  Linkage section.
  01 inDebit pic S9(9) binary.
  Procedure Division using by value inDebit.
    Subtract inDebit from AccountBalance.
  End method "debit".
*
*   print method to display formatted account number and balance:
  Identification Division.
  Method-id. "print".
  Data division.
  Local-storage section.
  01 PrintableAccountNumber pic ZZZZZ999999.
  01 PrintableAccountBalance pic $$$,$$$,$$9CR.
  Procedure Division.
    Move AccountNumber to PrintableAccountNumber
    Move AccountBalance to PrintableAccountBalance
    Display " Account: " PrintableAccountNumber
    Display " Balance: " PrintableAccountBalance.
  End method "print".
*
  End Object.
*
  End class Account.

```

Check クラス

```
/**
 * A Java class for check information
 */
public class Check {
    private CheckingAccount payer;
    private Account         payee;
    private int             amount;

    public Check(CheckingAccount inPayer, Account inPayee, int inAmount) {
        payer=inPayer;
        payee=inPayee;
        amount=inAmount;
    }

    public int getAmount() {
        return amount;
    }

    public Account getPayee() {
        return payee;
    }
}
```

関連タスク

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

クライアントの定義

クラス内で 1 つ以上のメソッドからサービスを要求するプログラムまたはメソッドは、そのクラスの **クライアント** と呼ばれます。

COBOL クライアントまたは Java クライアントで、以下のことを行うことができます。

- Java および COBOL クラスのオブジェクト・インスタンスを作成する。
- Java および COBOL オブジェクトにインスタンス・メソッドを呼び出す。
- COBOL ファクトリー・メソッドおよび Java 静的メソッドを呼び出す。

COBOL クライアントでは、Java Native Interface (JNI) が提供するサービスを呼び出すこともできます。

COBOL クライアント・プログラムは、次のような通常の 4 つの部分で成り立っています。

表 50. COBOL クライアントの構成

除算	目的	構文
IDENTIFICATION (必須)	クライアントの名前。	通常のようにコーディング。ただし、クライアント・プログラムは以下のとおりにする必要があります。 <ul style="list-style-type: none">• スレッド対応 (THREAD オプションでコンパイル、スレッド化アプリケーション用コーディングの指針に準拠)

表 50. COBOL クライアントの構成 (続き)

除算	目的	構文
ENVIRONMENT (必須)	コンピューター環境を記述する。クライアント内で使用されるクラス名を、コンパイル単位の外側で判明している、対応する外部クラス名に関連付ける。	CONFIGURATION SECTION (必須) 『クライアント定義用の REPOSITORY 段落』 (必須)
DATA (オプション)	クライアントが必要とするデータを記述する。	447 ページの『クライアント定義用の DATA DIVISION』 (オプション)
PROCEDURE (オプション)	クラスのインスタンスの作成、オブジェクト参照データ項目の操作、メソッドの呼び出し。	INVOKE、IF、および SET ステートメントを使用してコーディング

457 ページの『例: クライアントの定義』

関連タスク

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

545 ページの『第 29 章 マルチスレッド化のための COBOL プログラムの準備』

477 ページの『第 24 章 Java メソッドとの通信』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

455 ページの『クラスのインスタンスの作成および初期化』

449 ページの『オブジェクト参照の比較および設定』

450 ページの『メソッドの呼び出し (INVOKE)』

466 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』

関連参照

311 ページの『THREAD』

クライアント定義用の REPOSITORY 段落

指定された語を COBOL クライアント内で使用するときその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

外部クラス名は大/小文字が区別されます。したがって、Java 形成規則に準拠しなければなりません。例えば、Account および Check クラスを使用するクライアント・プログラムにおいて、以下のようにコーディングすることがあります。

```
Environment division.      必須
Configuration section.    必須
    Source-Computer.
    Object-Computer.
Repository.                必須
    Class Account is "Account"
    Class Check  is "Check".
```

REPOSITORY 段落記入項目では、クライアント内で Account および Check として参照されるクラスの外部クラス名は、それぞれ、Account および Check であることが示されます。

REPOSITORY 段落では、クライアントにおいて明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定しなければなりません。

Java パッケージの一部である参照クラスごとに外部クラス名を指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド (.) が付き、続いてその後に Java クラスの単純名が付きます。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形成規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名の作成は、クラス定義で外部クラス名が REPOSITORY 段落記入項目に組み込まれていない場合と同じ方法で、クラス名から行われます。外部名は英大/小文字混合で名前付けされます。したがって、上記の例で、クラス Account およびクラス Check は、それぞれ、外部的には Account および Check (英大/小文字混合) として認識されません。

CONFIGURATION SECTION の SOURCE-COMPUTER、OBJECT-COMPUTER、および SPECIAL-NAMES 段落はオプションです。

関連タスク

433 ページの『クラス定義用の REPOSITORY 段落』

関連参照

REPOSITORY 段落 (COBOL for AIX 言語解説書)

The Java Language Specification (Identifiers)

The Java Language Specification (Packages)

クライアント定義用の DATA DIVISION

クライアントが必要とするデータを記述するには、DATA DIVISION の任意のセクションを使用できます。

Data Division.

Local-storage section.

```
01 anAccount          usage object reference Account.
01 aCheckingAccount  usage object reference CheckingAccount.
01 aCheck             usage object reference Check.
01 payee              usage object reference Account.
. . .
```

クライアントはクラスを参照するので、オブジェクト参照 (つまり、クラスのインスタンスへの参照) と呼ばれる 1 つ以上の特別なデータ項目を必要とします。インスタンス・メソッドへのすべての要求は、メソッドがサポートされる (すなわち、継承によって定義されているか、または使用可能である) クラスのインスタンスへのオブジェクト参照が必要です。COBOL クラスのインスタンスを参照する場合と

同じ構文を使用して、オブジェクト参照をコーディングして Java クラスのインスタンスを参照します。上記の例では、`usage object reference` という句は、オブジェクト参照データ項目を表します。

上記コードの 4 つのオブジェクト参照はすべて、クラス名が `OBJECT REFERENCE` 句の後に現れるので、**型式化オブジェクト参照**と呼ばれます。型式化オブジェクト参照の参照先は、`OBJECT REFERENCE` 句で名前付けしたクラスのインスタンス、またはそのサブクラスのいずれか 1 つに限られます。したがって、`anAccount` は、`Account` クラスのインスタンス、またはそのサブクラスのいずれかを参照できますが、ほかのクラスのインスタンスを参照することはできません。同様に、`aCheck` の参照先は、`Check` クラスのインスタンス、またはそのサブクラスのインスタンスに限られます。

別の型のオブジェクト参照 (上に示されていない) には、`OBJECT REFERENCE` 句の後にクラス名がありません。そのような参照を**汎用オブジェクト参照**と呼びます。すべてのクラスのインスタンスを参照できるという意味です。汎用オブジェクト参照は非常に限られた環境 (`INVOKE class-name NEW . . .` ステートメントの `RETURNING` 句で使用するとき) でしか Java と相互運用できないため、汎用オブジェクト参照をコーディングすることは避けてください。

`OBJECT REFERENCE` 句で使用するクラス名は、`CONFIGURATION SECTION` の `REPOSITORY` 段落で定義しなければなりません。

関連タスク

『`LOCAL-STORAGE` または `WORKING-STORAGE` の選択』

483 ページの『`COBOL` および Java での相互運用可能なデータ型のコーディング』

450 ページの『メソッドの呼び出し (`INVOKE`)』

446 ページの『クライアント定義用の `REPOSITORY` 段落』

関連参照

`RETURNING` 句 (*COBOL for AIX* 言語解説書)

LOCAL-STORAGE または WORKING-STORAGE の選択

一般に、クライアント・プログラムが必要とする作業データを定義する場合に、`WORKING-STORAGE SECTION` を使用できます。しかし、プログラムがマルチスレッドで同時に実行できるような場合、代わりに `LOCAL-STORAGE SECTION` にデータを定義することができます。

各スレッドは、`LOCAL-STORAGE` データの別々のコピーへのアクセス権を持っていますが、`WORKING-STORAGE` データの単一のコピーへのアクセス権は共用します。`WORKING-STORAGE SECTION` でデータを定義する場合には、データへのアクセスを同期化するか、または、2 つのスレッドが同時にそのデータへアクセスしないようにする必要があります。

関連タスク

545 ページの『第 29 章 マルチスレッド化のための `COBOL` プログラムの準備』

オブジェクト参照の比較および設定

条件ステートメントまたは JNI サービス `IsSameObject` の呼び出しをコーディングすることによってオブジェクト参照を比較できます。また、SET ステートメントを使用して、オブジェクト参照を設定できます。

例えば、次の IF ステートメントのいずれかをコーディングして、オブジェクト参照 `anAccount` がオブジェクト・インスタンスをまったく参照していないことをチェックします。

```
If anAccount = Null . . .  
If anAccount = Nulls . . .
```

`IsSameObject` の呼び出しをコーディングして、2 つのオブジェクト参照 (`object1` と `object2`) が同じオブジェクト・インスタンスを指すかどうか、あるいはそれぞれがオブジェクト・インスタンスを指さないかどうかを検査することができます。引数および戻り値が Java と相互運用可能であることを確認し、呼び出し可能サービスへのアドレス可能性を確立するには、`IsSameObject` への呼び出しの前に、以下のデータ定義およびステートメントをコーディングします。

```
Local-storage Section.  
. . .  
01 is-same Pic X.  
   88 is-same-false Value X'00'.  
   88 is-same-true  Value X'01' Through X'FF'.  
Linkage Section.  
  Copy JNI.  
Procedure Division.  
  Set Address Of JNIEnv To JNIEnvPtr  
  Set Address Of JNINativeInterface To JNIEnv  
  Call IsSameObject Using By Value JNIEnvPtr object1 object2  
                                Returning is-same  
  If is-same-true . . .
```

メソッド内では、オブジェクト参照と `SELF` を比較する `IsSameObject` の呼び出しをコーディングすることにより、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト参照が指すかどうかを検査できます。

上記の代わりに、Java `equals` メソッド (`java.lang.Object` からの継承) を呼び出して、2 つのオブジェクト参照が同一のオブジェクト・インスタンスを参照するかを決定することができます。

SET ステートメントを使用することにより、オブジェクト参照がオブジェクト・インスタンスを指さないようにすることができます。以下に例を示します。

```
Set anAccount To Null.
```

また、SET ステートメントを使用して、1 つのオブジェクト参照が別のオブジェクト参照と同じインスタンスを参照するように設定することもできます。以下に例を示します。

```
Set anotherAccount To anAccount.
```

この SET ステートメントによって、`anotherAccount` は `anAccount` と同じオブジェクト・インスタンスを参照します。受け取り側 (`anotherAccount`) が汎用オブジェクト参照である場合、送り出し側 (`anAccount`) は、汎用オブジェクト参照か、または型式化オブジェクト参照のどちらかになります。受け取り側が型式化オブジェクト参

照である場合は、送り出し側も、受け取り側と同じクラス、または、そのサブクラスのいずれか 1 つにバインドされた、型式化オブジェクト参照でなければなりません。

メソッド内では、オブジェクト参照を SELF に設定して、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト参照が参照するように設定することができます。以下に例を示します。

```
Set anAccount To Self.
```

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
477 ページの『JNI サービスへのアクセス』

関連参照

The Java Native Interface (IsSameObject)

メソッドの呼び出し (INVOKE)

Java クライアントでは、COBOL でインプリメントされたクラスのオブジェクト・インスタンスを作成し、標準 Java 構文を使用して、それらのオブジェクトでメソッドを呼び出すことができます。COBOL クライアントでは、INVOKE ステートメントをコーディングすることにより、Java または COBOL クラスで定義されたメソッドを呼び出すことができます。

```
Invoke Account "createAccount"  
    using by value 123456  
    returning anAccount  
Invoke anAccount "credit" using by value 500.
```

上の最初の例の INVOKE ステートメントは、クラス名 Account を使用して、createAccount という名前のメソッドを呼び出します。このメソッドは、Account クラスで定義または継承されている必要があります。また、次のいずれかの型である必要があります。

- Java 静的メソッド
- COBOL ファクトリー・メソッド

using by value 123456 という句は、123456 が、メソッドの入力引数であり、値により受け渡されることを表します。入力引数 123456 と戻されるデータ項目 anAccount は、(多重定義されているかもしれない) createAccount メソッドの仮パラメーターおよび戻りの型の定義にそれぞれ準拠している必要があります。

2 番目の INVOKE ステートメントは、戻されるオブジェクト参照 anAccount を使用して、インスタンス・メソッド credit (Account クラスに定義されている) を呼び出します。入力引数 500 は、(おそらく多重定義された) credit メソッドの仮パラメーターの定義に準拠しなければなりません。

実行時におけるその値がターゲット・メソッドのシグニチャー内のメソッド名と一致するリテラルとして、または ID として呼び出すメソッドの名前をコーディングします。メソッド名は、英数字または国別リテラルであるか、あるいはカテゴリ・英字、英数字、または国別のデータ項目でなければならず、解釈されるときには大小文字が区別されます。

INVOKE ステートメントを（上記の 2 番目の例のステートメントのように）オブジェクト参照を使用してコーディングする場合、そのステートメントは次の 2 つの形式のうちのいずれかで始まります。

```
Invoke objRef "literal-name" . . .
Invoke objRef identifier-name . . .
```

メソッド名が ID である場合には、オブジェクト参照 (objRef) を、指定する型のない USAGE OBJECT REFERENCE として、すなわち、汎用オブジェクト参照として、定義しなければなりません。

呼び出されたメソッドが、オブジェクト参照の参照先のクラスでサポートされない場合、重大エラー条件が実行時に発生します。ただし、INVOKE ステートメントで ON EXCEPTION 句をコーディングした場合は別です。

オプションの範囲終了符号 END-INVOKE を INVOKE ステートメントで使用することができます。

INVOKE ステートメントは RETURN-CODE 特殊レジスターを設定しません。

関連タスク

『引数の引き渡し用の USING 句』

454 ページの『戻り値の取得用の RETURNING 句』

439 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

454 ページの『オーバーライドされたスーパークラス・メソッドの呼び出し』

466 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』

関連参照

INVOKE ステートメント (COBOL for AIX 言語解説書)

引数の引き渡し用の USING 句

引数をメソッドに渡す場合、INVOKE ステートメントの USING 句に引数を指定してください。それぞれの引数のデータ型が、意図されたターゲット・メソッドの対応する仮パラメーターの型と一致するように、それぞれの引数のデータ型をコーディングしてください。

表 51. COBOL クライアントでの引数の合致

ターゲット・メソッドのプログラミング言語	引数はオブジェクト参照ですか	引数の DATA DIVISION 定義を次のようにコーディングします	制約事項
COBOL	いいえ	対応する仮パラメーターの定義と同じ	
Java	いいえ	対応する Java パラメーターと相互運用可能	

表 51. COBOL クライアントでの引数の合致 (続き)

ターゲット・メソッドのプログラミング言語	引数はオブジェクト参照ですか	引数の DATA DIVISION 定義を次のようにコーディングします	制約事項
COBOL または Java	はい	ターゲット・メソッドの対応するパラメーターと同じクラスに型式化されるオブジェクト参照	COBOL クライアントでは (Java クライアントとは異なり)、引数のクラスを、対応するパラメーターのクラスのサブクラスにすることができません。

SET ステートメントまたは REDEFINES 節を使用して、オブジェクト参照引数を対応する仮パラメーターの型と一致させる方法については、以下に参照されている例を参照してください。

『例: COBOL クライアントからの規格合致オブジェクト参照の引数の引き渡し』

ターゲット・メソッドが多重定義されている場合、引数のデータ型は、同じ名前を持つメソッドの中から選択するために使用されます。

引数が BY VALUE で渡される指定をしなければなりません。言い換えれば、引数は、呼び出されるメソッドの対応する仮パラメーターが変更されても影響を受けません。

各引数のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

関連タスク

439 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

441 ページの『インスタンス・メソッドの多重定義』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

517 ページの『データの受け渡し』

関連参照

INVOKE ステートメント (COBOL for AIX 言語解説書)

SET ステートメント (COBOL for AIX 言語解説書)

REDEFINES 節 (COBOL for AIX 言語解説書)

例: COBOL クライアントからの規格合致オブジェクト参照の引数の引き渡し

以下の例は、COBOL クライアントのオブジェクト参照引数を、呼び出されるメソッドに対応する仮パラメーターの予想クラスに合致させる方法を示しています。

クラス C は、1 つのパラメーター (クラス java.lang.Object のオブジェクトへの参照) を持つメソッド M を定義します。

```

. . .
Class-id. C inherits Base.
. . .
Repository.
    Class Base          is "java.lang.Object"

```

```

        Class JavaObject is "java.lang.Object".
Identification division.
Factory.
. . .
Procedure Division.
Identification Division.
Method-id. "M".
Data division.
Linkage section.
01 obj object reference JavaObject.
Procedure Division using by value obj.
. . .

```

メソッド M を呼び出すには、COBOL クライアントは、クラス java.lang.Object のオブジェクトへの参照である引数を渡す必要があります。以下のクライアントは、データ項目 aString を定義していますが、これを M に引数として渡すことができません。aString は、クラス java.lang.String のオブジェクトへの参照だからです。クライアントはまず SET ステートメントを使用して、aString をデータ項目 anObj (クラス java.lang.Object のオブジェクトへの参照) に割り当てます。(java.lang.String は java.lang.Object のサブクラスなので、この SET ステートメントは正しいものです。) その後クライアントは anObj を引数として M に渡します。

```

. . .
Repository.
Class jstring is "java.lang.String"
Class JavaObject is "java.lang.Object".
Data division.
Local-storage section.
01 aString object reference jstring.
01 anObj object reference JavaObject.
*
Procedure division.
. . . (statements here assign a value to aString)
Set anObj to aString
Invoke C "M"
using by value anObj

```

SET ステートメントを使用して、クラス java.lang.Object のオブジェクトへの参照として anObj を取得する代わりに、クライアントは、以下のように REDEFINES 節で aString および anObj を定義することができます。

```

. . .
01 aString object reference jstring.
01 anObj redefines aString object reference JavaObject.

```

クライアントが値をデータ項目 aString (つまり、クラス java.lang.String のオブジェクトへの有効な参照) を割り当てた後、anObj を引数として M に渡すことができます。

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
439 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

関連参照

INVOKE ステートメント (COBOL for AIX 言語解説書)
SET ステートメント (COBOL for AIX 言語解説書)
REDEFINES 節 (COBOL for AIX 言語解説書)

戻り値の取得用の RETURNING 句

データ項目がメソッドの結果として戻される場合、その項目を、INVOKE ステートメントの RETURNING 句に指定してください。戻される項目は、クライアントの DATA DIVISION で定義します。

INVOKE ステートメントの RETURNING 句に指定する項目は、以下の表に示すように、ターゲット・メソッドが戻す型と合致している必要があります。

表 52. COBOL クライアントでの戻されるデータ項目の合致

ターゲット・メソッドのプログラミング言語	戻される項目はオブジェクト参照ですか	戻される項目の DATA DIVISION 定義を次のようにコーディングします
COBOL	いいえ	ターゲット・メソッドの RETURNING 項目の定義と同じ
Java	いいえ	戻された Java データ項目と相互運用可能
COBOL または Java	はい	ターゲット・メソッドが戻すオブジェクト参照と同じクラスに型式化されるオブジェクト参照

すべての場合において、戻り値のデータ型は、Java と相互運用可能になる型のうちのいずれかでなければなりません。

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

INVOKE ステートメント (*COBOL for AIX 言語解説書*)

オーバーライドされたスーパークラス・メソッドの呼び出し

ときおりクラス内で、現行クラスで定義された同じシグニチャーを持つメソッドを呼び出す代わりに、オーバーライドされたスーパークラス・メソッドを呼び出さなければならないことがあります。

例えば、CheckingAccount クラスが、その即時スーパークラス Account に定義されている debit インスタンス・メソッドをオーバーライドすると想定します。次のステートメントをコーディングして、CheckingAccount クラスのメソッド内で Account の debit メソッドを呼び出すことができます。

```
Invoke Super "debit" Using By Value amount.
```

debit メソッドのシグニチャーに一致するように、amount を PIC S9(9) BINARY として定義します。

CheckingAccount クラスは、Account クラスに定義されている print メソッドをオーバーライドします。print メソッドには仮パラメーターがありません。したがって、CheckingAccount クラス内のメソッドは、次のステートメントでスーパークラス print メソッドを呼び出すことができます。

```
Invoke Super "print".
```

キーワード SUPER は、現行クラス内のメソッドではなく、スーパークラス・メソッドを呼び出すことを指示します。(SUPER は、現在実行中のメソッドの起動に使用されているオブジェクトへの暗黙の参照です。)

428 ページの『例: 口座』

関連タスク

440 ページの『インスタンス・メソッドのオーバーライド』

関連参照

INVOKE ステートメント (COBOL for AIX 言語解説書)

クラスのインスタンスの作成および初期化

Java または COBOL クラスで定義されたインスタンス・メソッドを使用するには、まずクラスのインスタンスを作成する必要があります。

クラス *class-name* の新しいインスタンスを生成するには、また、作成したオブジェクトへの参照 *object-reference* を取得するには、次の形式のステートメントをコーディングします (*object-reference* は、クライアントの DATA DIVISION で定義されます)。

```
INVOKE class-name NEW . . . RETURNING object-reference
```

メソッド内に INVOKE . . . NEW ステートメントをコーディングしてあり、返されたオブジェクト参照の使用がメソッド呼び出しの期間に制限されていない場合は、JNI サービス NewGlobalRef を呼び出すことによって、返されたオブジェクト参照をグローバル参照に変換する必要があります。

```
Call NewGlobalRef using by value JNIEnvPtr object-reference  
returning object-reference
```

NewGlobalRef を呼び出さない場合には、戻されたオブジェクト参照はあくまでもローカル参照にすぎないため、メソッドが戻った後で自動的に解放されます。

関連タスク

『Java クラスのインスタンス化』

456 ページの『COBOL クラスのインスタンス化』

477 ページの『JNI サービスへのアクセス』

480 ページの『ローカル参照とグローバル参照の管理』

447 ページの『クライアント定義用の DATA DIVISION』

450 ページの『メソッドの呼び出し (INVOKE)』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

INVOKE ステートメント (COBOL for AIX 言語解説書)

Java クラスのインスタンス化

Java クラスをインスタンス化するには、そのクラスがサポートするパラメーター化コンストラクターを呼び出します。そのためには、INVOKE . . . NEW ステートメントで USING 句を RETURNING 句の直前にコーディングし、そのコンストラクターのシグニチャーと一致する引数の数および型を BY VALUE で渡します。

各引数のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。デフォルトの (パラメーターなし) コンストラクターを呼び出すには、USING 句を省略します。

例えば、Check クラスのインスタンスを生成し、そのインスタンス・データを初期化し、生成した Check インスタンスへの参照 aCheck を取得するには、次のステートメントを COBOL クライアントにコーディングすることができます。

```
Invoke Check New
  using by value aCheckingAccount, payee, 125
  returning aCheck
```

関連タスク

450 ページの『メソッドの呼び出し (INVOKE)』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

VALUE 節 (COBOL for AIX 言語解説書)

INVOKE ステートメント (COBOL for AIX 言語解説書)

COBOL クラスのインスタンス化

COBOL クラスをインスタンス化するには、型式化オブジェクト参照または汎用オブジェクト参照を、INVOKE . . . NEW ステートメントの RETURNING 句に指定できます。ただし、USING 句をコーディングすることはできません。インスタンス・データは、クラス定義の VALUE 節に指定されたとおりに初期化されます。

したがって、INVOKE . . . NEW ステートメントは、単一のインスタンス・データのみを有する COBOL クラスのインスタンスを生成するときに役立ちます。例えば、次のステートメントは、Account クラスのインスタンスを作成し、Account クラス定義の OBJECT 段落の WORKING-STORAGE SECTION の VALUE 節に指定されたとおりに、インスタンス・データを初期化し、新しいインスタンスへの参照 outAccount を提供します。

```
Invoke Account New returning outAccount
```

VALUE 節だけを使用して初期化することができない COBOL クラス・データの初期化を可能にするには、COBOL クラスを設計する際に、FACTORY 段落にパラメーター化生成メソッドを定義し、OBJECT 段落にパラメーター化初期化メソッドを定義する必要があります。

1. パラメーター化ファクトリー生成メソッドで、以下の手順を実行します。
 - a. INVOKE *class-name* NEW RETURNING *objectRef* をコーディングして、*class-name* のインスタンスを作成し、VALUE 節を有するインスタンス・データ項目に初期値を与えます。
 - b. パラメーター化した初期化メソッドをインスタンス (*objectRef*) 上で呼び出し、指定された引数 BY VALUE をファクトリー・メソッドに渡します。
2. 初期化メソッドで、ロジックをコーディングし、仮パラメーターを介して指定された値を使用して、インスタンス・データ初期化を完了します。

COBOL クラスのインスタンスを作成して適切に初期化するために、クライアントはパラメーター化ファクトリー・メソッドを呼び出し、BY VALUE で目的の引数を渡します。クライアントに戻されるオブジェクト参照は、ローカル参照です。メソッ

ド内にクライアント・コードがあり、戻されるオブジェクト参照を使用するのがそのメソッドの存続期間に限定されない場合、クライアント・コードは、JNI サービス `NewGlobalRef` を呼び出すことによって、戻されるオブジェクト参照をグローバル参照に変換しなければなりません。

467 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

477 ページの『JNI サービスへのアクセス』

480 ページの『ローカル参照とグローバル参照の管理』

450 ページの『メソッドの呼び出し (INVOKE)』

462 ページの『ファクトリー・セクションの定義』

関連参照

VALUE 節 (*COBOL for AIX 言語解説書*)

INVOKE ステートメント (*COBOL for AIX 言語解説書*)

クラスのインスタンスの解放

任意のクラスの個々のオブジェクト・インスタンスを解放するために、アクションを実行する必要はありません。したがって、オブジェクト・インスタンスを解放するために有効な構文はありません。Java ランタイム・システムは自動的にガーベッジ・コレクションを実行します。すなわち、使用されなくなったオブジェクトのメモリーを再利用します。

ただし、参照済みオブジェクトのガーベッジ・コレクションを許可するために、ネイティブ COBOL クライアント内のオブジェクトへのローカル参照またはグローバル参照を明示的に解放する必要がある場合があります。

関連タスク

480 ページの『ローカル参照とグローバル参照の管理』

例: クライアントの定義

次の例では、Account クラスの小さいクライアント・プログラムを示します。

プログラムは以下を実行します。

- ファクトリー・メソッド `createAccount` を呼び出して、デフォルト収支ゼロの Account インスタンスを作成します。
- インスタンス・メソッド `credit` を呼び出して、\$500 をこの新規の口座に預金します。
- インスタンス・メソッド `print` を呼び出して、口座の状況を表示します。

(Account クラスは、443 ページの『例: メソッドの定義』に示されています。)

```
cb1 thread,pgmname(longmixed)
Identification division.
Program-id. "TestAccounts" recursive.
Environment division.
Configuration section.
Repository.
    Class Account is "Account".
```

```

Data Division.
* Working data is declared in LOCAL-STORAGE instead of
* WORKING-STORAGE so that each thread has its own copy:
Local-storage section.
01 anAccount usage object reference Account.
*
Procedure division.
Test-Account-section.
    Display "Test Account class"
* Create account 123456 with 0 balance:
    Invoke Account "createAccount"
        using by value 123456
        returning anAccount
* Deposit 500 to the account:
    Invoke anAccount "credit" using by value 500
    Invoke anAccount "print"
    Display space
*
    Stop Run.
End program "TestAccounts".

```

467 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

464 ページの『ファクトリー・メソッドの定義』

466 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

サブクラスの定義

クラス (サブクラス、派生クラス、または子クラスと呼ばれる) を別のクラス (スーパークラス、基本クラス、または親クラスと呼ばれる) の特殊化クラスにすることができます。

サブクラスは、そのスーパークラスのメソッドおよびインスタンス・データを継承し、*is-a* 関係によって、そのスーパークラスに関連付けられています。例えば、サブクラス P がスーパークラス Q から継承し、サブクラス Q がスーパークラス S から継承した場合、P のインスタンスは Q のインスタンスであり、また (推移性によって) S のインスタンスでもあります。したがって、P のインスタンスは、Q と S のメソッドおよびデータを継承します。

サブクラスを使用することの利点:

- コードの再利用: 継承を通じて、サブクラスは、スーパークラスに既に存在するメソッドを再利用することができます。
- 特化: サブクラスでは、スーパークラスが処理しないケースを処理するために新規のメソッドを追加することができます。また、スーパークラスが必要としない新規のデータ項目を追加することもできます。
- アクションの変更: サブクラスは、スーパークラスにあるシグニチャーと同じシグニチャーのメソッドを定義して、スーパークラスから継承するメソッドをオーバーライドすることができます。メソッドをオーバーライドするときは、メソッドの実行内容について、いくつかの小さい変更を行うだけの場合、または全面的な変更を行う場合があります。

制約事項: COBOL プログラムでは多重継承を使用することはできません。定義する各 COBOL クラスには、Java または COBOL でインプリメントされた即時スーパークラスは必ず 1 つだけでなければなりません。また、それぞれのクラスは、直接的または間接的に `java.lang.Object` から派生したものでなければなりません。継承のセマンティクスは、Java によって定義されます。

サブクラスの構造および構文は、クラス定義の構造および構文と同一です。サブクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION に、インスタンス・データとメソッドを定義します。個別のオブジェクト・インスタンスにではなく、サブクラス自体に関連付けるデータとメソッドを必要とするサブクラスに、サブクラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

COBOL インスタンス・データは `private` です。サブクラスが COBOL スーパークラスのインスタンス・データにアクセスすることができるのは、スーパークラスがそのアクセスを可能にするために属性 (`get` または `set`) インスタンス・メソッドを定義する場合に限られます。

428 ページの『例: 口座』

461 ページの『例: サブクラスの定義 (メソッドに関して)』

関連タスク

430 ページの『クラスの定義』

440 ページの『インスタンス・メソッドのオーバーライド』

442 ページの『属性 (`get` および `set`) メソッドのコーディング』

461 ページの『サブクラス・インスタンス・メソッドの定義』

462 ページの『ファクトリー・セクションの定義』

関連参照

The Java Language Specification (Inheritance, overriding, and hiding)

COBOL クラス定義構成 (*COBOL for AIX 言語解説書*)

サブクラス定義用の CLASS-ID 段落

サブクラスを指定し、それが特性を継承する直接の Java または COBOL スーパークラスを示すには、CLASS-ID 段落を使用してください。

```
Identification Division.           必須  
Class-id. CheckingAccount inherits Account. 必須
```

上記の例で、定義されるサブクラスは `CheckingAccount` です。`CheckingAccount` は、サブクラス定義において `Account` として認識されているクラスのすべてのメソッドを継承します。`CheckingAccount` メソッドが `Account` インスタンス・データにアクセスできるのは、`Account` クラスが、そのアクセスを可能にするために、属性 (`get` または `set`) メソッドを指定する場合に限られます。

ENVIRONMENT DIVISION の CONFIGURATION SECTION の REPOSITORY 段落に、即時スーパークラスの名前を指定しなければなりません。オプションとして、外部で認識されているクラスの名前にスーパークラス名を関連付けることができます。また、定義中のサブクラスの名前 (上記の例の `CheckingAccount`) を REPOSITORY 段落に指定し、その対応する外部クラス名にそれを関連付けることもできます。

関連タスク

432 ページの『クラス定義用の CLASS-ID 段落』
442 ページの『属性 (get および set) メソッドのコーディング』
『サブクラス定義用の REPOSITORY 段落』

サブクラス定義用の REPOSITORY 段落

指定された語をサブクラス定義内で使用するときその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

例えば、CheckingAccount サブクラス定義では、これらの REPOSITORY 段落記入項目は、サブクラス定義内で CheckingAccount、Check、および Account として参照されるクラスの外部クラス名が、それぞれ、CheckingAccount、Check、および Account であることを示します。

Environment Division.		必須
Configuration Section.		必須
Repository.		必須
Class CheckingAccount	is "CheckingAccount"	オプション
Class Check	is "Check"	必須
Class Account	is "Account".	必須

REPOSITORY 段落では、サブクラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。以下に例を示します。

- 定義中のサブクラスが継承する元のユーザー定義スーパークラス
- サブクラス定義内のメソッドで参照するクラス

サブクラス内の REPOSITORY 段落記入項目をコーディングする場合の規則は、クラス内の REPOSITORY 段落記入項目をコーディングする場合の規則と同一です。

関連タスク

433 ページの『クラス定義用の REPOSITORY 段落』

関連参照

REPOSITORY 段落 (COBOL for AIX 言語解説書)

サブクラス・インスタンス・データ定義用の WORKING-STORAGE SECTION

スーパークラスに定義したインスタンスに加えてサブクラスが必要とするインスタンス・データを記述するには、サブクラス OBJECT 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。クラスにインスタンス・データを定義するとき使用する構文と同じ構文を使用します。

例えば、Account クラスの CheckingAccount サブクラスのインスタンス・データの定義は、以下のようになります。

```
Identification division.  
Object.  
Data division.
```

```
Working-storage section.  
01 CheckFee pic S9(9) value 1.  
.....  
End Object.
```

関連タスク

434 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

サブクラス・インスタンス・メソッドの定義

サブクラスは、そのスーパークラスのメソッドを継承します。サブクラス定義において、継承したメソッドと同じシグニチャーのインスタンス・メソッドを定義して、サブクラスが継承するインスタンス・メソッドをオーバーライドすることができます。また、サブクラスが必要とする新しいメソッドを定義することもできます。

サブクラス・インスタンス・メソッドの構造と構文は、クラス・インスタンス・メソッドの構造と構文と同一です。サブクラス定義の OBJECT 段落の PROCEDURE DIVISION にサブクラス・インスタンス・メソッドを定義します。

『例: サブクラスの定義 (メソッドに関して)』

関連タスク

436 ページの『クラス・インスタンス・メソッドの定義』
440 ページの『インスタンス・メソッドのオーバーライド』
441 ページの『インスタンス・メソッドの多重定義』

例: サブクラスの定義 (メソッドに関して)

次の例は、Account クラスの CheckingAccount サブクラスのインスタンス・メソッド定義を示しています。

processCheck メソッドは、Check クラスの Java インスタンス・メソッド getAmount および getPayee を呼び出して、チェック・データを取得します。Account クラスから継承した credit および debit インスタンス・メソッドを呼び出して、当座の受取人を貸し方に記入し、支払人を借方に記入します。

print メソッドは、Account クラスに定義されている print インスタンス・メソッドをオーバーライドします。オーバーライドした print メソッドを呼び出して、口座状況を表示し、また当座手数料も表示します。CheckFee は、サブクラスに定義するインスタンス・データ項目です。

(Account クラスは、443 ページの『例: メソッドの定義』に示されています。)

CheckingAccount クラス (Account のサブクラス)

```
cb1 thread,pgmname(1ongmixed)  
Identification Division.  
Class-id. CheckingAccount inherits Account.  
Environment Division.  
Configuration section.  
Repository.  
Class CheckingAccount is "CheckingAccount"
```

```

        Class Check          is "Check"
        Class Account       is "Account".
*
* (FACTORY paragraph not shown)
*
Identification division.
Object.
Data division.
Working-storage section.
01 CheckFee pic S9(9) value 1.
Procedure Division.
*
*   processCheck method to get the check amount and payee,
*   add the check fee, and invoke inherited methods debit
*   to debit the payer and credit to credit the payee:
Identification Division.
Method-id. "processCheck".
Data division.
Local-storage section.
01 amount pic S9(9) binary.
01 payee usage object reference Account.
Linkage section.
01 aCheck usage object reference Check.
*
Procedure Division using by value aCheck.
    Invoke aCheck "getAmount" returning amount
    Invoke aCheck "getPayee" returning payee
    Invoke payee "credit" using by value amount
    Add checkFee to amount
    Invoke self "debit" using by value amount.
End method "processCheck".
*
*   print method override to display account status:
Identification Division.
Method-id. "print".
Data division.
Local-storage section.
01 printableFee pic $$,$$$,$$9.
Procedure Division.
    Invoke super "print"
    Move CheckFee to printableFee
    Display " Check fee: " printableFee.
End method "print".
*
End Object.
*
End class CheckingAccount.

```

関連タスク

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

450 ページの『メソッドの呼び出し (INVOKE)』

440 ページの『インスタンス・メソッドのオーバーライド』

454 ページの『オーバーライドされたスーパークラス・メソッドの呼び出し』

ファクトリー・セクションの定義

個別のオブジェクト・インスタンスではなく、クラス自身に関連付けるデータおよびメソッドを定義するためには、クラス定義の FACTORY 段落を使用します。

COBOL ファクトリー・データ は、Java private 静的データと同じです。データの単一コピーは、そのクラス用にインスタンス生成され、クラスのすべてのオブジェクト・インスタンスに共有されます。クラスのすべてのインスタンスからデータを収集するときは、ごく一般的にファクトリー・データを使用します。例えば、ファクトリー・データ項目を定義して、作成するクラスのインスタンス数の現在高を集計できます。

COBOL ファクトリー・メソッド は Java public 静的メソッドと同じです。これらのメソッドは、どのオブジェクト・インスタンスとも無関係に、クラスによってサポートされます。VALUE 節を使用しただけではインスタンス・データを初期化できないときは、ごく一般的に、ファクトリー・メソッドを使用して、オブジェクトの生成をカスタマイズします。

対照的に、クラスのそれぞれのオブジェクト・インスタンスごとに作成されるデータを定義したり、クラスのそれぞれのオブジェクト・インスタンスごとにサポートされるメソッドを定義したりする場合には、クラス定義の OBJECT 段落を使用します。

ファクトリー定義は、以下の 3 つの部から構成され、その後に END FACTORY ステートメントが続きます。

表 53. ファクトリー定義の構成

除算	目的	構文
IDENTIFICATION (必須)	ファクトリー定義の開始を示す。	IDENTIFICATION DIVISION. FACTORY.
DATA (オプション)	このクラス用に一度割り振られたデータを記述する (クラスのそれぞれのインスタンスごとに割り振られたデータとは正反対)。	『ファクトリー・データ定義用の WORKING-STORAGE SECTION』 (オプション)
PROCEDURE (オプション)	ファクトリー・メソッドを定義する。	ファクトリー・メソッドの定義: 464 ページの『ファクトリー・メソッドの定義』

467 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

430 ページの『クラスの定義』

456 ページの『COBOL クラスのインスタンス化』

473 ページの『プロシージャ指向 COBOL プログラムのラッピング』

473 ページの『OO アプリケーションの構造化』

ファクトリー・データ定義用の WORKING-STORAGE SECTION

COBOL クラスが必要とする ファクトリー・データ、つまりクラスのすべてのオブジェクト・インスタンスに共有される、静的に割り当てられたデータを記述するには、FACTORY 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。

IDENTIFICATION DIVISION 宣言の直前に入れる必要がある FACTORY キーワードは、クラスのファクトリー・データおよびファクトリー・メソッドの定義の開始を示します。例えば、Account クラスのファクトリー・データの定義は、以下のようになります。

```
Identification division.  
Factory.  
  Data division.  
  Working-storage section.  
  01 NumberOfAccounts pic 9(6) value zero.  
  .  
  .  
  .  
End Factory.
```

上記に示すように、単純ファクトリー・データの初期化は、VALUE 節を使用して行うことができます。

COBOL ファクトリー・データ は、Java private 静的データと同じです。他のクラスまたはサブクラス (必要に応じて、同じクラス内のインスタンス・メソッドも) は、COBOL ファクトリー・データを直接参照することはできません。ファクトリー・データは、FACTORY 段落で定義するすべてのファクトリー・メソッドにグローバルです。FACTORY 段落の外側からファクトリー・データにアクセス可能にする場合には、アクセスを可能にするためにファクトリー属性 (get または set) メソッドを定義します。

関連タスク

442 ページの『属性 (get および set) メソッドのコーディング』

456 ページの『COBOL クラスのインスタンス化』

ファクトリー・メソッドの定義

クラス定義の FACTORY 段落の PROCEDURE DIVISION に COBOL ファクトリー・メソッド を定義します。ファクトリー・メソッドは、クラスのどのオブジェクト・インスタンスとも無関係に、クラスによってサポートされる操作を定義します。COBOL ファクトリー・メソッドは Java public 静的メソッドと同じです。

一般的には、そのインスタンスが複雑な初期化を必要とするクラスについて、すなわち、VALUE 節だけの使用では割り当てることができない値に対して、ファクトリー・メソッドを定義します。ファクトリー・メソッド内でインスタンス・メソッドを呼び出して、インスタンス・データを初期化することができます。ファクトリー・メソッドは、インスタンス・データに直接アクセスすることはできません。

ファクトリー属性 (get および set) メソッドをコーディングして、FACTORY 段落の外側からファクトリー・データにアクセス可能にすることができます。例えば、同じクラスのインスタンス・メソッドから、またはクライアント・プログラムからファクトリー・データにアクセス可能にすることができます。例えば、Account クラスはファクトリー・メソッド getNumberOfAccounts を定義して、口座数の現在の集計を戻すことができます。

ファクトリー・メソッドを使用して、Java プログラムからアクセス可能になるようにプロシージャ指向の COBOL プログラムをラップすることもできます。main という名前のファクトリー・メソッドをコーディングすることで、java コマンドを

使用してオブジェクト指向アプリケーションを実行したり、Java の標準的な方法に従ってアプリケーションを構成したりすることができます。詳細については、関連タスクを参照してください。

ファクトリー・メソッドの定義では、インスタンス・メソッドを定義するときを用いる構文と同じ構文を使用します。COBOL ファクトリー・メソッド定義は、4 つの部 (COBOL プログラムに類似) とその後の END METHOD マーカーから構成されません。

表 54. ファクトリー・メソッド定義の構成

除算	目的	構文
IDENTIFICATION (必須)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ (必須)
ENVIRONMENT (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
DATA (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
PROCEDURE (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ

クラス定義内では、各ファクトリー・メソッド名を固有にする必要はありませんが、各ファクトリー・メソッドに固有のシグニチャーを与える必要があります。ファクトリー・メソッドの多重定義は、インスタンス・メソッドを多重定義する場合とまったく同じ方法で行うことができます。例えば、CheckingAccount サブクラスは、2 つの版のファクトリー・メソッド createCheckingAccount、すなわち、口座を初期化してデフォルトの収支ゼロを設定する版と、開始残高を渡せるようにする版を提供します。クライアントは、意図したメソッドのシグニチャーと一致する引数を渡して、createCheckingAccount メソッドのいずれかを呼び出すことができます。

ファクトリー・メソッドの DATA DIVISION および FACTORY 段落の DATA DIVISION の両方において、データ項目を同じ名前前で定義した場合、そのデータ名に対するメソッド内の参照は、そのメソッド・データ項目だけを参照します。メソッド DATA DIVISION が優先します。

467 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

- 473 ページの『OO アプリケーションの構造化』
- 473 ページの『プロシージャ指向 COBOL プログラムのラッピング』
- 456 ページの『COBOL クラスのインスタンス化』
- 436 ページの『クラス・インスタンス・メソッドの定義』
- 442 ページの『属性 (get および set) メソッドのコーディング』
- 441 ページの『インスタンス・メソッドの多重定義』
- 466 ページの『ファクトリー・メソッドまたは静的メソッドの隠蔽』
- 466 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』

ファクトリー・メソッドまたは静的メソッドの隠蔽

サブクラスで定義されたファクトリー・メソッドは、そのサブクラスで利用できるのであれば、継承された COBOL または Java メソッドを (これら 2 つのメソッドが同じシグニチャーを持っている場合) 隠蔽すると言います。

スーパークラス・ファクトリー・メソッド f1 を COBOL サブクラスで隠すには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでファクトリー・メソッド f1 を定義します。(スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラス名を使用して f1 を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

メソッド戻り値の有無および PROCEDURE DIVISION RETURNING 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・ファクトリー・メソッドと隠されたスーパークラス・メソッドにおいて同一でなければなりません。

ファクトリー・メソッドは、Java または COBOL スーパークラスに、インスタンス・メソッドを隠してはいけません。

467 ページの『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

440 ページの『インスタンス・メソッドのオーバーライド』

450 ページの『メソッドの呼び出し (INVOKE)』

関連参照

The Java Language Specification (Inheritance, overriding, and hiding)

手続き部ヘッダー (COBOL for AIX 言語解説書)

ファクトリー・メソッドまたは静的メソッドの呼び出し

COBOL ファクトリー・メソッドまたは Java 静的メソッドを COBOL メソッドまたはクライアント・プログラムで呼び出すには、クラス名を INVOKE ステートメントの第 1 オペランドとしてコーディングしてください。

例えば、クライアント・プログラムは次のステートメントをコーディングして、createCheckingAccount という名前の多重定義 CheckingAccount ファクトリー・メソッドの 1 つを呼び出して、口座番号 777777 および開始残高 \$300 の当座預金を作成することができます。

```
Invoke CheckingAccount "createCheckingAccount"  
  using by value 777777 300  
  returning aCheckingAccount
```

ファクトリー・メソッドを定義する同じクラス内からファクトリー・メソッドを呼び出す場合にも、クラス名を INVOKE ステートメントの第 1 オペランドとして使用します。

実行時におけるその値がメソッド名であるリテラルとして、または ID として呼び出すメソッドの名前をコーディングします。メソッド名は、英数字または国別リテラルであるか、あるいはカテゴリ英字、英数字、または国別のデータ項目でなければならない、解釈されるときには大/小文字が区別されます。

呼び出されたメソッドが、INVOKE ステートメントで指定されたクラスでサポートされない場合、重大エラー条件が実行時に発生します。ただし、INVOKE ステートメントで ON EXCEPTION 句をコーディングした場合は別です。

USING 句で COBOL ファクトリー・メソッドまたは Java 静的メソッドに引数を渡すときの適合要件と、RETURNING 句で戻り値を受けるときの適合要件は、インスタンス・メソッドを呼び出す場合と同じです。

『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

450 ページの『メソッドの呼び出し (INVOKE)』

189 ページの『COBOL での国別データ (Unicode) の使用』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

INVOKE ステートメント (COBOL for AIX 言語解説書)

例: ファクトリーの定義 (メソッドに関して)

次の例では、以前の例が、ファクトリー・データおよびメソッドの定義を示すように更新します。

以下の更新が示されます。

- Account クラスは、ファクトリー・データとパラメーター化ファクトリー・メソッド createAccount を追加します。こうすることで、渡される口座番号を使用して Account インスタンスの作成が可能になります。
- CheckingAccount サブクラスは、ファクトリー・データおよび多重定義のパラメーター化ファクトリー・メソッド createCheckingAccount を追加します。
createCheckingAccount の 1 つのインプリメンテーションでデフォルトの収支ゼロの口座を初期化し、もう 1 つのインプリメンテーションで開始残高を渡せるようにします。クライアントは、目的のメソッドのシグニチャーと一致する引数を渡して、メソッドを呼び出すこともできます。
- TestAccounts クライアントは、Account および CheckingAccount クラスのファクトリー・メソッドによって提供されるサービスを呼び出して、Java Check クラスのインスタンスを生成します。
- TestAccounts クライアント・プログラムからの出力を表示します。

(以前の例とは、443 ページの『例: メソッドの定義』、457 ページの『例: クライアントの定義』、および 461 ページの『例: サブクラスの定義 (メソッドに関して)』のことです。)

また、上記の例の完全なソース・コードが COBOL インストール・ディレクトリーの samples サブディレクトリーに入っています。通常は /usr/lpp/cobol/samples/oo です。そこで MAKE ファイルを作成して、コードのコンパイルとリンクを行うことができます。

Account クラス

```
cb1 thread,pgmname(longmixed),lib
Identification Division.
Class-id. Account inherits Base.
Environment Division.
Configuration section.
Repository.
    Class Base    is "java.lang.Object"
    Class Account is "Account".
*
Identification division.
Factory.
Data division.
Working-storage section.
01 NumberOfAccounts pic 9(6) value zero.
*
Procedure Division.
*
*   createAccount method to create a new Account
*   instance, then invoke the OBJECT paragraph's init
*   method on the instance to initialize its instance data:
Identification Division.
Method-id. "createAccount".
Data division.
Linkage section.
01 inAccountNumber pic S9(6) binary.
01 outAccount object reference Account.
*   Facilitate access to JNI services:
    Copy JNI.
    Procedure Division using by value inAccountNumber
        returning outAccount.
*   Establish addressability to JNI environment structure:
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
    Invoke Account New returning outAccount
    Invoke outAccount "init" using by value inAccountNumber
    Add 1 to NumberOfAccounts.
    End method "createAccount".
*
End Factory.
*
Identification division.
Object.
Data division.
Working-storage section.
01 AccountNumber pic 9(6).
01 AccountBalance pic S9(9) value zero.
*
Procedure Division.
*
*   init method to initialize the account:
Identification Division.
Method-id. "init".
Data division.
Linkage section.
01 inAccountNumber pic S9(9) binary.
Procedure Division using by value inAccountNumber.
    Move inAccountNumber to AccountNumber.
    End method "init".
*
```

```

*   getBalance method to return the account balance:
    Identification Division.
    Method-id. "getBalance".
    Data division.
    Linkage section.
    01 outBalance pic S9(9) binary.
    Procedure Division returning outBalance.
      Move AccountBalance to outBalance.
    End method "getBalance".

*
*   credit method to deposit to the account:
    Identification Division.
    Method-id. "credit".
    Data division.
    Linkage section.
    01 inCredit pic S9(9) binary.
    Procedure Division using by value inCredit.
      Add inCredit to AccountBalance.
    End method "credit".

*
*   debit method to withdraw from the account:
    Identification Division.
    Method-id. "debit".
    Data division.
    Linkage section.
    01 inDebit pic S9(9) binary.
    Procedure Division using by value inDebit.
      Subtract inDebit from AccountBalance.
    End method "debit".

*
*   print method to display formatted account number and balance:
    Identification Division.
    Method-id. "print".
    Data division.
    Local-storage section.
    01 PrintableAccountNumber pic ZZZZZ999999.
    01 PrintableAccountBalance pic $$$,$$$,$$9CR.
    Procedure Division.
      Move AccountNumber to PrintableAccountNumber
      Move AccountBalance to PrintableAccountBalance
      Display " Account: " PrintableAccountNumber
      Display " Balance: " PrintableAccountBalance.
    End method "print".

*
    End Object.

*
    End class Account.

```

CheckingAccount クラス (Account のサブクラス)

```

cbl thread,pgmname(longmixed),lib
Identification Division.
Class-id. CheckingAccount inherits Account.
Environment Division.
Configuration section.
Repository.
  Class CheckingAccount is "CheckingAccount"
  Class Check          is "Check"
  Class Account        is "Account".

*
Identification division.
Factory.
Data division.
Working-storage section.
01 NumberOfCheckingAccounts pic 9(6) value zero.

*
Procedure Division.

```

```

*
*   createCheckingAccount overloaded method to create a new
*   CheckingAccount instance with a default balance, invoke
*   inherited instance method init to initialize the account
*   number, and increment factory data tally of checking accounts:
Identification Division.
Method-id. "createCheckingAccount".
Data division.
Linkage section.
01 inAccountNumber pic S9(6) binary.
01 outCheckingAccount object reference CheckingAccount.
*   Facilitate access to JNI services:
    Copy JNI.
Procedure Division using by value inAccountNumber
    returning outCheckingAccount.
*   Establish addressability to JNI environment structure:
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
    Invoke CheckingAccount New returning outCheckingAccount
    Invoke outCheckingAccount "init"
        using by value inAccountNumber
    Add 1 to NumberOfCheckingAccounts.
End method "createCheckingAccount".

*
*   createCheckingAccount overloaded method to create a new
*   CheckingAccount instance, invoke inherited instance methods
*   init to initialize the account number and credit to set the
*   balance, and increment factory data tally of checking accounts:
Identification Division.
Method-id. "createCheckingAccount".
Data division.
Linkage section.
01 inAccountNumber pic S9(6) binary.
01 inInitialBalance pic S9(9) binary.
01 outCheckingAccount object reference CheckingAccount.
    Copy JNI.
Procedure Division using by value inAccountNumber
                                inInitialBalance
    returning outCheckingAccount.
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
    Invoke CheckingAccount New returning outCheckingAccount
    Invoke outCheckingAccount "init"
        using by value inAccountNumber
    Invoke outCheckingAccount "credit"
        using by value inInitialBalance
    Add 1 to NumberOfCheckingAccounts.
End method "createCheckingAccount".

*
End Factory.
*
Identification division.
Object.
Data division.
Working-storage section.
01 CheckFee pic S9(9) value 1.
Procedure Division.

*
*   processCheck method to get the check amount and payee,
*   add the check fee, and invoke inherited methods debit
*   to debit the payer and credit to credit the payee:
Identification Division.
Method-id. "processCheck".
Data division.
Local-storage section.
01 amount pic S9(9) binary.
01 payee usage object reference Account.

```

```

Linkage section.
01 aCheck usage object reference Check.
Procedure Division using by value aCheck.
  Invoke aCheck "getAmount" returning amount
  Invoke aCheck "getPayee" returning payee
  Invoke payee "credit" using by value amount
  Add checkFee to amount
  Invoke self "debit" using by value amount.
End method "processCheck".
*
*   print method override to display account status:
  Identification Division.
  Method-id. "print".
  Data division.
  Local-storage section.
  01 printableFee pic $,$$,,$9.
  Procedure Division.
    Invoke super "print"
    Move CheckFee to printableFee
    Display " Check fee: " printableFee.
  End method "print".
*
  End Object.
*
  End class CheckingAccount.

```

Check クラス

```

/**
 * A Java class for check information
 */
public class Check {
  private CheckingAccount payer;
  private Account        payee;
  private int             amount;

  public Check(CheckingAccount inPayer, Account inPayee, int inAmount) {
    payer=inPayer;
    payee=inPayee;
    amount=inAmount;
  }

  public int getAmount() {
    return amount;
  }

  public Account getPayee() {
    return payee;
  }
}

```

TestAccounts クライアント・プログラム

```

cbl thread,pgmname(longmixed)
  Identification division.
  Program-id. "TestAccounts" recursive.
  Environment division.
  Configuration section.
  Repository.
    Class Account          is "Account"
    Class CheckingAccount is "CheckingAccount"
    Class Check           is "Check".
  Data Division.
  * Working data is declared in Local-storage
  * so that each thread has its own copy:
  Local-storage section.
  01 anAccount          usage object reference Account.

```

```

01 aCheckingAccount usage object reference CheckingAccount.
01 aCheck           usage object reference Check.
01 payee            usage object reference Account.
*
Procedure division.
Test-Account-section.
  Display "Test Account class"
* Create account 123456 with 0 balance:
  Invoke Account "createAccount"
    using by value 123456
    returning anAccount
* Deposit 500 to the account:
  Invoke anAccount "credit" using by value 500
  Invoke anAccount "print"
  Display space
*
  Display "Test CheckingAccount class"
* Create checking account 777777 with balance of 300:
  Invoke CheckingAccount "createCheckingAccount"
    using by value 777777 300
    returning aCheckingAccount
* Set account 123456 as the payee:
  Set payee to anAccount
* Initialize check for 125 to be paid by account 777777 to payee:
  Invoke Check New
    using by value aCheckingAccount, payee, 125
    returning aCheck
* Debit the payer, and credit the payee:
  Invoke aCheckingAccount "processCheck"
    using by value aCheck
  Invoke aCheckingAccount "print"
  Invoke anAccount "print"
*
  Stop Run.
End program "TestAccounts".

```

TestAccounts クライアント・プログラムが生成する出力

```

Test Account class
Account:      123456
Balance:      $500

Test CheckingAccount class
Account:      777777
Balance:      $174
Check fee:    $1
Account:      123456
Balance:      $625

```

関連タスク

- 455 ページの『クラスのインスタンスの作成および初期化』
- 464 ページの『ファクトリー・メソッドの定義』
- 466 ページの『ファクトリー・メソッドまたは静的メソッドの呼び出し』
- 257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

プロシージャ指向 COBOL プログラムのラッピング

ラッパー は、オブジェクト指向コードとプロシージャ指向コードの間のインターフェースを提供するクラスです。ファクトリー・メソッドは、既存のプロシージャ型 COBOL コード用にラッパーを書き込み、Java プログラムからアクセス可能にするときの便利な方法を提供します。

COBOL コードをラップするためには、以下の手順を実行します。

1. FACTORY 段落を含む単純 COBOL クラスを作成する。
2. FACTORY 段落で、CALL ステートメントを使用してプロシージャ型プログラムを呼び出すファクトリー・メソッドをコーディングする。

Java プログラムは、静的メソッドの呼び出しの式を使用して、すなわち、COBOL プロシージャ型プログラムを呼び出して、ファクトリー・メソッドを呼び出すことができます。

関連タスク

430 ページの『クラスの定義』

462 ページの『ファクトリー・セクションの定義』

464 ページの『ファクトリー・メソッドの定義』

00 アプリケーションの構造化

次の 3 つの方法のいずれかで、オブジェクト指向 COBOL 構文を使用するアプリケーションを構造化することができます。

オブジェクト指向アプリケーションは、次のいずれかで始めることができます。

- COBOL プログラム。名前は何でも構いません。

コマンド・プロンプトで実行可能モジュールの名前を指定することによって、プログラムを実行することができます。

- main という名前のメソッドを含む Java クラス定義。main は、単一の String[] 型パラメーターを持つ public、static、および void として宣言します。

main を含むクラスの名前を指定し、0 以上のストリングをコマンド行引数として渡すことで、java コマンドでアプリケーションを実行できます。

- main という名前のファクトリー・メソッドを含む COBOL クラス定義。main は、RETURNING 句を指定せず、java.lang.String 型のエレメントの配列であるクラスへのオブジェクト参照である単一の USING パラメーターを指定して宣言します。つまり、main は、事実上、String[] 型のパラメーターを 1 つ持つ、public、static、および void です。

main を含むクラスの名前を指定し、0 以上のストリングをコマンド行引数として渡すことで、java コマンドでアプリケーションを実行できます。

以下の場合には、この方法でオブジェクト指向アプリケーションを構成します。

- java コマンドを使用してアプリケーションを実行する。

- アプリケーションが Java クラス・ファイルの main メソッドで開始しなければならぬ環境でアプリケーションを実行する。
- 標準的な Java プログラミング方式に従う。

『例: java コマンドを使用して実行される COBOL アプリケーション』

関連タスク

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

254 ページの『プログラムの実行』

464 ページの『ファクトリー・メソッドの定義』

484 ページの『Java 用の配列およびストリングの宣言』

例: java コマンドを使用して実行される COBOL アプリケーション

以下の例では、main という名前のファクトリー・メソッドを含む COBOL クラス定義を示します。

いずれの場合も、main には RETURNING 句がなく、java.lang.String 型のエレメントの配列であるクラスへのオブジェクト参照である単一の USING パラメーターがあります。これらのアプリケーションは、java コマンドを使用して実行できます。

メッセージの表示

```

cbl thread
Identification Division.
Class-id. CBLmain inherits Base.
Environment Division.
Configuration section.
Repository.
    Class Base is "java.lang.Object"
    Class stringArray is "objectArray:java.lang.String"
    Class CBLmain is "CBLmain".
*
Identification Division.
Factory.
    Procedure division.
*
    Identification Division.
    Method-id. "main".
    Data division.
    Linkage section.
    01 SA usage object reference stringArray.
    Procedure division using by value SA.
        Display " >> COBOL main method entered"
        .
    End method "main".
End factory.
End class CBLmain.

```

入カストリングのエコー

```

cbl thread,lib,ssrange
Identification Division.
Class-id. Echo inherits Base.
Environment Division.

```

```

Configuration section.
Repository.
    Class Base is "java.lang.Object"
    Class stringArray is "jobjectArray:java.lang.String"
    Class jstring is "java.lang.String"
    Class Echo is "Echo".
*
Identification Division.
Factory.
    Procedure division.
*
    Identification Division.
    Method-id. "main".
    Data division.
    Local-storage section.
    01 SALen          pic s9(9) binary.
    01 I              pic s9(9) binary.
    01 SAelement     object reference jstring.
    01 SAelementlen  pic s9(9) binary.
    01 P              pointer.
    Linkage section.
    01 SA             object reference stringArray.
    01 Sbuffer        pic N(65535).
    Copy "JNI.cpy" suppress.
    Procedure division using by value SA.
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
    Call GetArrayLength using by value JNIEnvPtr SA
    returning SALen
    Display "Input string array length: " SALen
    Display "Input strings:"
    Perform varying I from 0 by 1 until I = SALen
    Call GetObjectArrayElement
    using by value JNIEnvPtr SA I
    returning SAelement
    Call GetStringLength
    using by value JNIEnvPtr SAelement
    returning SAelementlen
    Call GetStringChars
    using by value JNIEnvPtr SAelement 0
    returning P
    Set address of Sbuffer to P
    Display function display-of(Sbuffer(1:SAelementlen))
    Call ReleaseStringChars
    using by value JNIEnvPtr SAelement P
    End-perform
    .
    End method "main".
End factory.
End class Echo.

```

関連タスク

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

464 ページの『ファクトリー・メソッドの定義』

477 ページの『第 24 章 Java メソッドとの通信』

第 24 章 Java メソッドとの通信

Java との言語間インターオペラビリティを達成するには、Java Native Interface (JNI) でのサービスの使用、データ型のコーディング、および COBOL プログラムのコンパイルに関する特定の規則および指針に従う必要があります。

Java で書き込まれたメソッドを COBOL プログラムから呼び出したり、COBOL で書き込まれたメソッドを Java プログラムから呼び出したりすることができます。Java の基本オブジェクト機能に対応するには、COBOL オブジェクト指向言語をコーディングする必要があります。追加の Java 機能に対応するには、JNI サービスを呼び出すことができます。

Java プログラムはマルチスレッド化され、非同期シグナルを用いる場合があります。したがって、THREAD オプションを使用して COBOL プログラムをコンパイルしてください。

関連概念

261 ページの『Java SDK サポート』

関連タスク

257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

『JNI サービスへのアクセス』

482 ページの『Java とのデータ共有』

427 ページの『第 23 章 オブジェクト指向プログラムの作成』

545 ページの『第 29 章 マルチスレッド化のための COBOL プログラムの準備』

関連参照

Java 2 SDK, Standard Edition Documentation

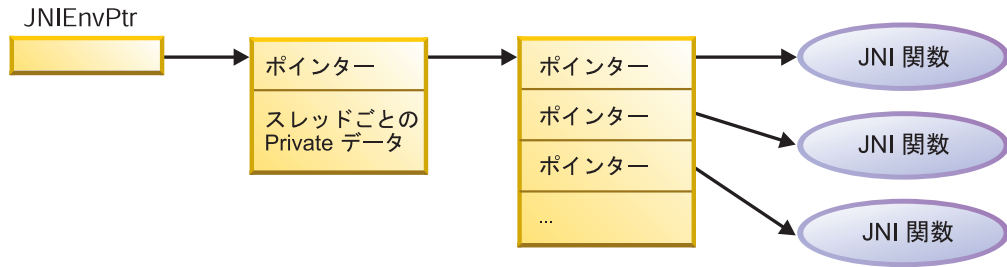
JNI サービスへのアクセス

Java Native Interface (JNI) は、COBOL と Java を併用するアプリケーションを開発する際に使用できる多くの呼び出し可能サービスを提供します。このサービスへのアクセスを円滑に行うには、COBOL プログラムの LINKAGE SECTION に JNI.cpy をコピーします。

この JNI.cpy コピーブックには、以下の定義が含まれています。

- Java JNI タイプに対応する COBOL データ定義
- JNINativeInterface、呼び出し可能サービス関数にアクセスするための関数ポインターが含まれている JNI 環境構造

次の図に示すように、JNI 環境ポインターからの 2 つのレベルの間接化技法によって JNI 環境構造を取得します。



特殊レジスタ `JNIEnvPtr` を使用して、JNI 環境ポインタを参照し、JNI 環境構造のアドレスを取得します。`JNIEnvPtr` は、`USAGE POINTER` として暗黙的に定義されます。それを受取データ項目として使用することはできません。JNI 環境構造の内容を参照する前に、以下のステートメントをコーディングして、そのアドレス可能性を確立する必要があります。

```
Linkage section.
COPY JNI
. . .
Procedure division.
    Set address of JNIEnv to JNIEnvPtr
    Set address of JNINativeInterface to JNIEnv
. . .
```

上記コードは、以下の項目のアドレスを設定します。

- `JNIEnv`。`JNI.cpy` が提供するポインタ・データ項目です。`JNIEnvPtr` は、環境ポインタが含まれている `COBOL` 特殊レジスタです。
- `JNINativeInterface`。`JNI.cpy` に含まれている `COBOL` グループ構造です。この構造には、JNI 呼び出し可能サービスの関数ポインタの配列が含まれている JNI 環境構造がマップされています。

上記のステートメントをコーディングした後に、関数ポインタを参照する `CALL` ステートメントを使用して、JNI 呼び出し可能サービスにアクセスすることができます。次の例に示すように、環境ポインタを必要とするサービスに、最初の引数として `JNIEnvPtr` 特殊レジスタを渡すことができます。

```
01 InputArrayObj usage object reference jlongArray.
01 ArrayLen pic S9(9) comp-5.
. . .
    Call GetArrayLength using by value JNIEnvPtr InputArrayObj
    returning ArrayLen
```

重要: すべての引数を値によって JNI 呼び出し可能サービスに渡します。相互互換性を保持するためには、引数はネイティブ・フォーマットである必要があります (データ記述項目の `NATIVE` 節で宣言されているなど)。

一部の JNI 呼び出し可能サービスは、Java クラス・オブジェクト参照を引数として必要とします。クラスに関連付けられたクラス・オブジェクトへの参照を取得するには、以下の JNI 呼び出し可能サービスのどちらかを使用します。

- `GetObjectClass`
- `FindClass`

制限: JNI 環境ポインタはスレッド固有のものです。スレッドから別のスレッドに渡すことはできません。

関連タスク

480 ページの『ローカル参照とグローバル参照の管理』

『Java 例外の処理』

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

445 ページの『クライアントの定義』

関連参照

266 ページの『ADDR』

705 ページの『付録 H. JNI.cpy』

The Java Native Interface

Java 例外の処理

JNI サービスを使用して、Java 例外を throw したり、catch したりします。

例外のスロー: COBOL メソッドから Java 例外をスローするには、次のいずれかのサービスを使用します。

- Throw
- ThrowNew

throw したオブジェクトは、`java.lang.Throwable` のサブクラスのインスタンスにする必要があります。

Java 仮想マシン (JVM) は、呼び出しを含んでいるメソッドが完了して JVM に戻るまで、throw された例外を認識して処理することはいけません。

例外のキャッチ: Java 例外をスローした可能性があるメソッドを呼び出してから、次の手順を行うことができます。

1. 例外が発生したかどうかをテストします。
2. 例外が発生した場合は、その例外を処理します。
3. 例外をクリアします (クリアが適切な場合)。

次の JNI サービスを使用します。

- ExceptionOccurred
- ExceptionCheck
- ExceptionDescribe
- ExceptionClear

エラー分析を行うには、戻された例外オブジェクトによってサポートされるメソッドを使用します。このオブジェクトは、`java.lang.Throwable` クラスのインスタンスです。

『例: Java 例外の処理』

例: Java 例外の処理

次の例は、Java からの例外を catch するための JNI サービスの使用と、エラー分析を行うための `java.lang.Throwable` の `printStackTrace` メソッドの使用を示しています。

```

Repository.
  Class JavaException is "java.lang.Exception".
. . .
Local-storage section.
01 ex usage object reference JavaException.
Linkage section.
COPY "JNI.cpy".
. . .
Procedure division.
  Set address of JNIEnv to JNIEnvPtr
  Set address of JNINativeInterface to JNIEnv
. . .
  Invoke anObj "someMethod"
  Perform ErrorCheck
. . .
ErrorCheck.
  Call ExceptionOccurred
    using by value JNIEnvPtr
    returning ex
  If ex not = null then
    Call ExceptionClear using by value JNIEnvPtr
    Display "Caught an unexpected exception"
    Invoke ex "printStackTrace"
  Stop run
End-if

```

ローカル参照とグローバル参照の管理

Java 仮想マシンは、ネイティブ・メソッド (COBOL メソッドなど) で使用されるオブジェクト参照を追跡します。この追跡によって、ガーベッジ・コレクションの際、まだ使用中のオブジェクトが解放されないようにします。

オブジェクト参照には、以下の 2 つのクラスがあります。

ローカル参照

ローカル参照は、呼び出したメソッドが稼働している間のみ有効です。ネイティブ・メソッドが戻ると、ローカル参照の自動解放が実行されます。

グローバル参照

グローバル参照は、明示的に削除するまで有効です。グローバル参照は、JNI サービス `NewGlobalRef` を使用して、ローカル参照から作成することができます。

以下のオブジェクト参照は常にローカルです。

- メソッド・パラメーターとして受け取られるオブジェクト参照
- メソッドの RETURNING 値としてメソッドの起動から戻されるオブジェクト参照
- JNI 関数への呼び出しによって戻されるオブジェクト参照
- INVOKE . . . NEW ステートメントを使用して作成するオブジェクト参照

ローカル参照またはグローバル参照のいずれかをオブジェクト参照引数として JNI サービスに渡すことができます。

RETURNING 値としてローカル参照またはグローバル参照のいずれかを戻すメソッドをコーディングできます。ただし、いずれの場合も、呼び出すプログラムが受け取る参照はローカル参照です。

メソッドの起動で USING 引数としてローカル参照またはグローバル参照のいずれかを渡すことができます。ただし、いずれの場合も、呼び出されたメソッドが受け取る参照はローカル参照です。

ローカル参照は、それが作成されたスレッド内でのみ有効です。ローカル参照をスレッドから別のスレッドに渡すことはできません。

関連タスク

477 ページの『JNI サービスへのアクセス』
『ローカル参照の削除、保管、および解放』

ローカル参照の削除、保管、および解放

ローカル参照は、メソッド内で、随時に手動で削除できます。ローカル参照は、メソッドの LOCAL-STORAGE SECTION に定義したオブジェクト参照内にものみ保管します。

次のいずれかのデータ項目で参照を保管する場合にローカル参照をグローバル参照に変換するには、SET ステートメントを使用します。

- オブジェクト・インスタンス変数
- ファクトリー変数
- メソッドの WORKING-STORAGE SECTION 内のデータ項目

そうしないと、エラーが発生します。メソッドが戻ったときにこれらのストレージ域は保持されるので、ローカル参照は無効になります。

ほとんどのケースにおいて、メソッドが戻るときに発生するローカル参照の自動解放に依存することができます。ただし、一部のケースにおいては、JNI サービス DeleteLocalRef を使用して、メソッド内のローカル参照を明示的に解放する必要があります。以下に、明示的解放が適切な 2 つの状態を示します。

- メソッドにおいて、ラージ・オブジェクトにアクセスすることで、オブジェクトへのローカル参照を作成します。膨大な計算を行った後で、メソッドが戻ります。このラージ・オブジェクトを別の計算に必要としない場合には、このオブジェクトを解放してください。このローカル参照は、ガーベッジ・コレクションの間にオブジェクトを解放する妨げになるからです。
- メソッドに多数のローカル参照を作成しますが、それらのすべてのローカル参照を同時には使用しません。Java 仮想マシンは、各ローカル参照を追跡するためのスペースが必要です。不要になったローカル参照を解放すると、システムがメモリー不足にならないようにすることができます。

例えば、COBOL メソッドにおいて、大規模な配列のオブジェクトをループし、エレメントをローカル参照として検索し、それぞれの反復ごとに 1 つのエレメントを操作します。それぞれの反復後に、配列エレメントへのローカル参照を解放することができます。

ローカル参照およびグローバル参照を管理するには、以下の呼び出し可能サービスを使用してください。

表 55. ローカルおよびグローバル参照の JNI サービス

サービス	入力引数	戻り値	目的
NewGlobalRef	<ul style="list-style-type: none"> JNI 環境ポインター ローカルまたはグローバル・オブジェクト参照 	グローバル参照、またはシステムがメモリー不足のときは NULL	入力オブジェクト参照が参照するオブジェクトに新規グローバル参照を作成する
DeleteGlobalRef	<ul style="list-style-type: none"> JNI 環境ポインター グローバル・オブジェクト参照 	なし	入力オブジェクト参照が参照するオブジェクトへのグローバル参照を削除する
DeleteLocalRef	<ul style="list-style-type: none"> JNI 環境ポインター ローカル・オブジェクト参照 	なし	入力オブジェクト参照が参照するオブジェクトへのローカル参照を削除する

関連タスク

477 ページの『JNI サービスへのアクセス』

Java アクセス制御

Java アクセス修飾子 `protected` および `private` は、Java Native Interface を使用すると、強制されません。したがって、COBOL プログラムは、Java クライアントからは呼び出し不可能な `protected` または `private` Java メソッドを呼び出すことができます。この使用法はお勧めできません。

Java とのデータ共用

Java データ型と同じものを持つ COBOL データ型を共有することができます。(COBOL データ型には、Java データ型と同じものがありますが、同じでないものもあります。)

次の方法で Java とデータ項目を共有します。

- INVOKE ステートメントの USING 句に引数として渡します。
- Java メソッドから、USING 句のパラメーターとして受け取ります。
- INVOKE ステートメントの RETURNING 値として受け取ります。
- COBOL メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句の値として戻します。

配列およびストリングを渡したり受け取ったりするには、以下のように、それらをオブジェクト参照として宣言します。

- 特殊配列クラスのうちの 1 つのインスタンスが含まれているオブジェクト参照として、配列を宣言します。
- `jstring` クラスのインスタンスが含まれているオブジェクト参照として、ストリングを宣言します。

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
 484 ページの『Java 用の配列およびストリングの宣言』

- 485 ページの『Java 配列の取り扱い』
- 488 ページの『Java スtringの取り扱い』
- 450 ページの『メソッドの呼び出し (INVOKE)』
- 517 ページの『第 27 章 データの共用』

COBOL および Java での相互運用可能なデータ型のコーディング

Java との通信時には、COBOL プログラムは、特定のデータ型しか使用できません。

表 56. COBOL および Java で相互運用可能なデータ型

Java の基本データ型	対応する COBOL データ型
boolean ¹	PIC X の後に以下の形式とまったく同じ 2 つの条件名を記述する。 <code>level-number data-name PIC X.</code> <code>88 data-name-false value X'00'.</code> <code>88 data-name-true value X'01' through X'FF'.</code>
byte ¹	1 バイト英数字: PIC X または PIC A
short	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(n) の PICTURE 節付き。ここで、1<=n<=4
int	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(n) の PICTURE 節付き。ここで、5<=n<=9
long	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9(n) の PICTURE 節。ここで、10<=n<=18
float ²	USAGE COMP-1
double ²	USAGE COMP-2
char	1 文字基本国別: PIC N USAGE NATIONAL. (国別グループは不可です。)
クラス型 (オブジェクト参照)	USAGE OBJECT REFERENCE <i>class-name</i>

1. boolean 型と byte 型はいずれも PIC X に対応しているため、この 2 つは区別する必要があります。PIC X は、前述の 2 つの条件名を指定して引数またはパラメーターを定義した場合にのみ、boolean 型として解釈されます。それ以外の場合、PIC X データ項目は、Java の byte 型として解釈されます。1 バイト英数字項目には、EBCDIC またはネイティブ・コンテンツを含めることができます。

2. Java 浮動小数点データは、IEEE 浮動小数点として表現されます。INVOKE ステートメントで引数として渡す浮動小数点データ項目、または Java メソッドからパラメーターとして受け取る浮動小数点データ項目は、ネイティブ・フォーマットである必要があります。そのため、cob2_j コマンドの -host オプション、または FLOAT(S390) オプションを使用してコンパイルした場合には、Java メソッドとやり取りする各浮動小数点データ項目は、データ記述で NATIVE 句を使用して宣言する必要があります。

関連タスク

189 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

271 ページの『CHAR』

289 ページの『FLOAT』

Java 用の配列およびストリングの宣言

Java と通信する場合には、特別な配列クラスを使用して配列を宣言し、jstring を使用してストリングを宣言してください。下のテーブルで示された COBOL データ型をコーディングしてください。

表 57. COBOL および Java で相互運用可能な配列およびストリング

Java データ型	対応する COBOL データ型
boolean[]	オブジェクト参照 jbooleanArray
byte[]	オブジェクト参照 jbyteArray
short[]	オブジェクト参照 jshortArray
int[]	オブジェクト参照 jintArray
long[]	オブジェクト参照 jlongArray
char[]	オブジェクト参照 jcharArray
Object[]	オブジェクト参照 jobjectArray
String	オブジェクト参照 jstring

Java とのインターオペラビリティのためにこれらのクラスのいずれかを使用するには、REPOSITORY 段落で項目をコーディングする必要があります。以下に例を示します。

```
Configuration section.  
Repository.  
    Class jbooleanArray is "jbooleanArray".
```

オブジェクト配列型に対する REPOSITORY 段落記入項目では、以下のいずれかの形式の外部クラス名を指定しなければなりません。

```
"jobjectArray"  
"jobjectArray:external-classname-2"
```

最初のケースでは、REPOSITORY 記入項目は、配列エレメントが java.lang.Object 型のオブジェクトである配列クラスを指定しています。2 番目のケースでは、REPOSITORY 記入項目は、配列のエレメントが external-classname-2 型のオブジェクトである配列クラスを指定しています。jobjectArray 型の指定と、配列のエレメントの外部クラス名の間で分離文字として、コロンをコーディングします。

次の例は、両方のケースを示しています。この例では、oa は、java.lang.Object 型のオブジェクトであるエレメントの配列を定義しています。また、aDepartment は、com.acme.Employee 型のオブジェクトであるエレメントの配列を定義しています。

```
Environment Division.  
Configuration Section.  
Repository.  
    Class jobjectArray is "jobjectArray"  
    Class Employee    is "com.acme.Employee"  
    Class Department  is "jobjectArray:com.acme.Employee".  
    . . .
```

```

Linkage section.
01 oa          usage object reference jobjectArray.
01 aDepartment usage object reference Department.
. . .
Procedure division using by value aDepartment.
. . .

```

474 ページの『例: java コマンドを使用して実行される COBOL アプリケーション』

以下の Java 配列型は現在、COBOL プログラムとの相互協調処理にはサポートされていません。

表 58. COBOL および Java で相互運用可能でない配列型

Java データ型	対応する COBOL データ型
float[]	オブジェクト参照 jfloatArray
double[]	オブジェクト参照 jdoubleArray

関連タスク

433 ページの『クラス定義用の REPOSITORY 段落』

Java 配列の取り扱い

COBOL プログラムで配列を表すには、その配列の Java タイプに対応するデータ・タイプの単一基本項目が含まれているグループ項目をコーディングします。その配列に適した OCCURS または OCCURS DEPENDING ON 節を指定します。

例えば、次のコードは、jlongArray オブジェクトから 500 以下の整数値を受け取る構造を指定します。

```

01 jlongArray.
   02 X pic S9(10) comp-5 occurs 1 to 500 times depending on N.

```

特殊な Java 配列クラスのオブジェクトを操作するには、JNI が提供するサービスを呼び出します。サービスを使用して、配列の個々のエレメントにアクセスして設定し、呼び出したサービスを使用して、以下を行います。

表 59. JNI 配列サービス

サービス	入力引数	戻り値	目的
GetArrayLength	<ul style="list-style-type: none"> JNI 環境ポインター 配列オブジェクトの参照 	2 進数フルワード整数としての配列の長さ	Java 配列オブジェクト内のエレメント数を取得する
NewBooleanArray、 NewByteArray、 NewCharArray、 NewShortArray、 NewIntArray、 NewLongArray	<ul style="list-style-type: none"> JNI 環境ポインター 2 進数フルワード整数としての、配列内のエレメントの数 	配列オブジェクトの参照、または配列を構成できない場合は NULL	新しい Java 配列オブジェクトを作成する

表 59. JNI 配列サービス (続き)

サービス	入力引数	戻り値	目的
GetBooleanArrayElements、 GetByteArrayElements、 GetCharArrayElements、 GetShortArrayElements、 GetIntArrayElements、 GetLongArrayElements	<ul style="list-style-type: none"> • JNI 環境ポインター • 配列オブジェクトの参照 • ブール項目へのポインターポインターが NULL でない場合は、配列エレメントのコピーが作成されたときは、ブール項目は true に設定される。コピーが作成された場合、変更を配列オブジェクトに書き戻す必要がある場合は、対応する ReleasexxxArrayElements サービスを呼び出さなければならない。 	ストレージ・バッファへのポインター	配列エレメントを Java 配列からストレージ・バッファに抽出する。サービスにより、ポインターがストレージ・バッファに戻される。ポインターは、LINKAGE SECTION に定義される COBOL グループ・データ項目のアドレスとして使用することができる。
ReleaseBooleanArrayElements、 ReleaseByteArrayElements、 ReleaseCharArrayElements、 ReleaseShortArrayElements、 ReleaseIntArrayElements、 ReleaseLongArrayElements	<ul style="list-style-type: none"> • JNI 環境ポインター • 配列オブジェクトの参照 • ストレージ・バッファへのポインター • 2 進数フルワード整数としてのリリース・モード。詳細については、Java JNI の資料を参照。(推奨: 配列の内容をコピーして戻し、ストレージ・バッファを解放するには、0 を指定する。) 	なし。配列のストレージは解放される。	Java 配列から抽出したエレメントが含まれているストレージ・バッファを解放し、条件によっては、更新された配列値を配列オブジェクトにマップして戻す。
NewObjectArray	<ul style="list-style-type: none"> • JNI 環境ポインター • 2 進数フルワード整数としての、配列内のエレメントの数 • 配列エレメント・クラスに対するオブジェクト参照 • 最初のエレメント値に対するオブジェクト参照。すべての配列エレメントにはこの値が設定されます。 	配列オブジェクトの参照、または配列を構成できない場合は NULL ¹	新しい Java オブジェクト配列を作成する。
GetObjectArrayElement	<ul style="list-style-type: none"> • JNI 環境ポインター • 配列オブジェクトの参照 • 2 進数フルワード整数としての、起点 0 の配列エレメント索引 	オブジェクト参照 ²	オブジェクト配列内の特定の索引のエレメントを返す。
SetObjectArrayElement	<ul style="list-style-type: none"> • JNI 環境ポインター • 配列オブジェクトの参照 • 2 進数フルワード整数としての、起点 0 の配列エレメント索引 • 新しい値に対するオブジェクト参照 	なし ³	オブジェクト配列内のエレメントを設定する。

表 59. JNI 配列サービス (続き)

サービス	入力引数	戻り値	目的
1. システムがメモリー不足の場合、NewObjectArray は例外を throw します。 2. 索引が有効でない場合、GetObjectArrayElement は例外を throw します。 3. 索引が有効でない場合、または新しい値が配列の要素・クラスのサブクラスでない場合、SetObjectArrayElement は例外を throw します。			

474 ページの『例: java コマンドを使用して実行される COBOL アプリケーション』
 『例: Java int 配列の処理』

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
 484 ページの『Java 用の配列およびストリングの宣言』
 477 ページの『JNI サービスへのアクセス』

例: Java int 配列の処理

次の例は、Java 配列クラスと JNI サービスを使用した、COBOL での Java 配列の処理を示しています。

```

cb1 lib,thread
Identification division.
Class-id. OOARRAY inherits Base.
Environment division.
Configuration section.
Repository.
    Class Base is "java.lang.Object"
    Class jintArray is "jintArray".
Identification division.
Object.
Procedure division.
    Identification division.
    Method-id. "ProcessArray".
    Data Division.
    Local-storage section.
    01 intArrayPtr pointer.
    01 intArrayLen pic S9(9) comp-5.
    Linkage section.
        COPY JNI.
    01 inIntArrayObj usage object reference jintArray.
    01 intArrayGroup.
        02 X pic S9(9) comp-5
            occurs 1 to 1000 times depending on intArrayLen.
    Procedure division using by value inIntArrayObj.
        Set address of JNIEnv to JNIEnvPtr
        Set address of JNINativeInterface to JNIEnv

        Call GetArrayLength
            using by value JNIEnvPtr inIntArrayObj
            returning intArrayLen
        Call GetIntArrayElements
            using by value JNIEnvPtr inIntArrayObj 0
            returning IntArrayPtr
        Set address of intArrayGroup to intArrayPtr

* . . . process the array elements X(I) . . .

        Call ReleaseIntArrayElements
    
```

```

        using by value JNIEnvPtr inIntArrayObj intArrayPtr 0.
    End method "ProcessArray".
End Object.
End class OOARRAY.

```

Java スtringの取り扱い

COBOL は、Java スtring・データを Unicode で表します。Java スtringを COBOL プログラムで表すには、jstring クラスのオブジェクト参照として String を宣言してください。続いて、JNI サービスを使用して、COBOL 国別 (Unicode) または UTF-8 データを設定するか、オブジェクトから抽出します。

Unicode 用のサービス: jstring オブジェクト参照と COBOL USAGE NATIONAL データ項目との間の変換を行うには、以下の標準サービスを使用してください。これらのサービスへのアクセスは、JNINativeInterface 環境構造の関数ポインターを使用しています。

表 60. jstring 参照と国別データ間の変換サービス

サービス	入力引数	戻り値
NewString ¹	<ul style="list-style-type: none"> JNI 環境ポインター COBOL 国別データ項目などの、Unicode スtringへのポインター Stringの文字数。2 進数フルワード 	jstring オブジェクト参照。
GetStringLength	<ul style="list-style-type: none"> JNI 環境ポインター jstring オブジェクト参照 	jstring オブジェクト参照の Unicode 文字数。2 進数フルワード。
GetStringChars ¹	<ul style="list-style-type: none"> JNI 環境ポインター jstring オブジェクト参照 ブール・データ項目を指すポインター、または NULL 	<ul style="list-style-type: none"> jstring オブジェクトから抜き出された Unicode 文字の配列を指すポインター、または NULL (操作が失敗した場合)。ポインターは、ReleaseStringChars を使用して解放されるまで有効。 ブール・データ項目へのポインターが NULL でないとき、ブール値は、Stringのコピーが作成される場合には true に、コピーが作成されない場合には false に設定される。
ReleaseStringChars	<ul style="list-style-type: none"> JNI 環境ポインター jstring オブジェクト参照 GetStringChars から戻された Unicode 文字の配列へのポインター 	なし。配列のストレージは解放される。

1. システムがメモリー不足の場合、このサービスは例外を throw します。

UTF-8 サービス JNI の拡張機能である以下のサービスを使用して、jstring オブジェクト参照と UTF-8 String間の変換を行うことができます。メインフレームに移植可能である必要がないプログラムでこれらのサービスを使用します。これらのサービスへのアクセスは、JNI 環境構造 JNINativeInterface の関数ポインターを使用しています。

表 61. `jstring` 参照と UTF-8 データ間の変換サービス

サービス	入力引数	戻り値
<code>NewStringUTF</code> ¹	<ul style="list-style-type: none"> JNI 環境ポインタ ヌル終了 UTF-8 ストリングへのポインタ 	<code>jstring</code> オブジェクト参照、またはストリングを構成できない場合は <code>NULL</code>
<code>GetStringUTFLength</code>	<ul style="list-style-type: none"> JNI 環境ポインタ <code>jstring</code> オブジェクト参照 	UTF-8 フォーマットでストリングを表すのに必要なバイト数、バイナリー・フルワード
<code>GetStringUTFChars</code> ¹	<ul style="list-style-type: none"> JNI 環境ポインタ <code>jstring</code> オブジェクト参照 プール・データ項目を指すポインタ、または <code>NULL</code> 	<ul style="list-style-type: none"> <code>jstring</code> オブジェクトから抜き出された UTF-8 文字の配列を指すポインタ、または <code>NULL</code> (操作が失敗した場合)。ポインタは、<code>ReleaseStringUTFChars</code> を使用して解放されるまで有効。 プール・データ項目へのポインタが <code>NULL</code> でないとき、プール値は、ストリングのコピーが作成される場合には <code>true</code> に、コピーが作成されない場合には <code>false</code> に設定される。
<code>ReleaseStringUTFChars</code>	<ul style="list-style-type: none"> JNI 環境ポインタ <code>jstring</code> オブジェクト参照 <code>GetStringUTFChars</code> を使用して、<code>jstring</code> アーギュメントから取り出した UTF-8 ストリングへのポインタ 	なし、UTF-8 ストリングのストレージがリリースされます。

1. システムがメモリー不足の場合、このサービスは例外を `throw` します。

関連タスク

- 477 ページの『JNI サービスへのアクセス』
- 483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
- 484 ページの『Java 用の配列およびストリングの宣言』
- 189 ページの『COBOL での国別データ (Unicode) の使用』
- 257 ページの『第 13 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

第 7 部 複雑なアプリケーションを扱う作業

第 25 章 プラットフォーム間でのアプリケーションの移植	493
コンパイルするメインフレーム・アプリケーションの取得	493
実行するメインフレーム・アプリケーションの取得: 概要	494
データ表現による違いの修正	494
ASCII SBCS 文字と EBCDIC SBCS 文字の違いの処理	495
IEEE データと 16 進数データの違いの処理	495
ASCII マルチバイト と EBCDIC DBCS ストリングの違いの処理	496
移植性に影響する環境の違いの修正	497
言語エレメントによる違いの修正	497
メインフレーム上で実行するコードの作成	498
AIX と Windows 間で移植可能なアプリケーションの作成	499
第 26 章 サブプログラムの使用	501
メインプログラム、サブプログラム、および呼び出し	501
メインプログラムまたはサブプログラムの終了と再入	502
ネストされた COBOL プログラムの呼び出し	503
ネストされたプログラム	503
例: ネストされたプログラムの構造	505
名前の有効範囲	505
ローカル名	506
グローバル名	506
名前の宣言の探索	506
ネストなし COBOL プログラムの呼び出し	506
CALL identifier および CALL literal	507
COBOL および C/C++ プログラム間の呼び出し	508
環境の初期設定	509
COBOL と C/C++ 間でのデータの受け渡し	509
スタック・フレームの縮小と実行単位またはプロセスの終了	510
COBOL および C/C++ のデータ型	511
例: C 関数を呼び出す COBOL プログラム	511
例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プログラム	512
例: C プログラムによって呼び出される COBOL プログラム	513
例: コンパイルおよび実行の結果の例	514
例: C++ 関数を呼び出す COBOL プログラム	514
再帰呼び出しの実行	515
戻りコードの受け渡し	516
第 27 章 データの共用	517
データの受け渡し	517

呼び出し側プログラムの中での引数の記述	519
呼び出し先プログラムの中でのパラメーターの記述	519
OMITTED 引数に関するテスト	520
LINKAGE SECTION のコーディング	520
引数を受け渡すための PROCEDURE DIVISION のコーディング	521
受け渡されるデータのグループ化	522
ヌル終了ストリングの処理	522
ポインターによるチェーン・リストの処理	523
例: チェーン・リストを処理するためのポインターの使用	523
プロシージャ・ポインターと関数ポインターの使用	526
戻りコード情報の引き渡し	526
RETURN-CODE 特殊レジスターの理解	527
PROCEDURE DIVISION RETURNING の使用	527
CALL RETURNING の指定	527
EXTERNAL 節によるデータの共用	527
プログラム間でのファイルの共用 (外部ファイル)	528
例: 外部ファイルの使用	529
外部ファイルを使用する入出力	529
コマンド行引数の使用	532
例: -host オプションを使用しないコマンド行引数	532
例: -host オプションを使用したコマンド行引数	533
第 28 章 共用ライブラリーの構築	535
スタティック・リンクおよび共用ライブラリー	535
共用ライブラリーへの参照をリンカーが解決する方法	536
例: サンプルの共用ライブラリーの作成	537
例 1: alpha.cbl	537
例 2: beta.cbl	537
例 3: gamma.cbl	538
プロシージャ	538
例: サンプルの共用ライブラリー用の Make ファイルの作成	540
例: CALL identifier を使用した共用ライブラリーの呼び出し	540
dll.cbl	540
dlla.cbl	541
プロシージャ	541
例: COBOL および C の共用ライブラリーを呼び出す COBOL プログラム	542
例 1: d16.cbl	542
例 2: d16a.c	543
例 3: d16b.cbl	543
プロシージャ	543

第 29 章 マルチスレッド化のための COBOL プログラムの準備	545
マルチスレッド化	545
マルチスレッド化による言語エレメントの処理	547
実行単位の有効範囲を持つエレメントの処理	547
プログラム呼び出しインスタンスの有効範囲を持つエレメントの処理	548
マルチスレッド化を使用した COBOL 言語エレメントの有効範囲	548
マルチスレッド化サポートのための THREAD の選択	549
マルチスレッド化されたプログラムへの制御権移動	549
マルチスレッド化されたプログラムの終了	550
マルチスレッド化による COBOL 制限の処理	550
例: マルチスレッド環境での COBOL の使用	551
thrcob.c のソース・コード	551
subd.cbl のソース・コード	553
sube.cbl のソース・コード	553
第 30 章 COBOL ランタイム環境の事前初期設定	555
永続的な COBOL 環境の初期設定	555
事前初期設定された COBOL 環境の終了	556
例: COBOL 環境の事前初期設定	557
第 31 章 2 桁年の日付の処理	561
2000 年言語拡張 (MLE)	562
この拡張の原則と目標	563
日付に関連したロジック問題の解決	564
世紀ウィンドウの使用	565
例: 世紀ウィンドウ	566
内部ブリッジングの使用	566
例: 内部ブリッジング	567
完全フィールド拡張への移行	567
例: 拡張日付形式へのファイルの変換	568
年先行型、年単独型、および年末尾型の日付フィールドの使用	570
互換性のある日付	570
例: 年先行型日付フィールドの比較	571
その他の日付形式の使用	572
例: 年の分離	572
リテラルを日付として操作する	573
仮定による世紀ウィンドウ	574
非日付の処理	575
符号条件の使用	576
日付フィールドに対する算術の実行	576
ウィンドウ表示日付フィールドのオーバーフローの考慮	577
評価の順序の指定	578
日付処理の明示的制御	579
DATEVAL の使用	579
UNDATE の使用	580
例: DATEVAL	580
例: UNDATE	580
日付関連診断メッセージの分析および回避	581
日付処理上の問題の回避	583
バック 10 進数フィールドの問題の回避	583

拡張日付フィールドからウィンドウ表示日付フィールドへの移動	584
-------------------------------	-----

第 25 章 プラットフォーム間でのアプリケーションの移植

AIX ワークステーションには、IBM メインフレームや Windows ベースのワークステーションとは異なるハードウェアやオペレーティング・システムのアーキテクチャが採用されています。このようなアーキテクチャーの違いのために、COBOL プログラムをこれらのプラットフォーム環境間で移植する際には、いくつかの問題が発生する可能性があります。

以下の関連タスクと関連参照では、開発プラットフォーム間の違いと、移植性の問題を最小限に抑えるための方法について説明します。

関連タスク

- 『コンパイルするメインフレーム・アプリケーションの取得』
- 494 ページの『実行するメインフレーム・アプリケーションの取得: 概要』
- 498 ページの『メインフレーム上で実行するコードの作成』
- 499 ページの『AIX と Windows 間で移植可能なアプリケーションの作成』

関連参照

- 619 ページの『付録 B. ホスト COBOL との違いの要約』

コンパイルするメインフレーム・アプリケーションの取得

プログラムをメインフレームから AIX ワークステーションに移行して、この新しい環境でコンパイルする場合は、正しいコンパイラー・オプションを選択して、メインフレームの COBOL とは異なる言語機能を意識する必要があります。COPY ステートメントを使用して、プログラムの移植に役立てることもできます。

正しいコンパイラー・オプションの選択: NOADV COBOL コンパイラー・オプション (メインフレームで利用可能) は、COBOL for AIX では適用できず、コメントとして扱われます。これにより、予測不能な結果が生じる可能性があります。コンパイラーは、オブジェクトに W レベルのメッセージでフラグを立てます。

NOADV NOADV を使用する必要があるプログラムは装置制御文字に依存するため、一般には移植できません。プログラムが NOADV に依存する場合は、プリンター制御文字をファイル内のレベル 01 のレコードの先頭文字として想定しないように、プログラムを修正してください。

移植性に影響するホスト・コンパイラー・オプションについて詳しくは、コンパイラー・オプションに関する下記の関連参照を参照してください。

メインフレーム COBOL の言語機能の考慮: メインフレーム COBOL プログラムで複数の言語機能が有効になっている場合に、COBOL for AIX を使用してコンパイルを行うと、エラーが発生したり、予期しない結果が生じたりすることがあります。詳しくは、下記の言語エレメントに関する関連参照を参照してください。

移植の問題に役立つ COPY ステートメントの使用: 多くの場合、COPY ステートメントを使用してプラットフォーム固有のコードを分離すると、移植性の問題を回避することができます。例えば、あるプラットフォーム用のコンパイル時にプラットフ

オーム固有のコードを組み込み、別のプラットフォーム用のコンパイル時にはそのコードを除外することができます。また、COPY REPLACING 句を使用して、ファイル名などの移植不可能なソース・コード・エレメントをグローバル変更することも可能です。

関連参照

228 ページの『ランタイム環境変数』

619 ページの『コンパイラー・オプション』

623 ページの『言語エレメント』

COPY ステートメント (COBOL for AIX 言語解説書)

実行するメインフレーム・アプリケーションの取得: 概要

ソース・プログラムをダウンロードして AIXワークステーション上で正常にコンパイルしたら、次にそのプログラムを実行します。多くの場合、ソースを大幅に変更しなくても、メインフレーム上での結果と同じものを取得できます。

ソースに変更を加えるべきかどうかを判断するには、基本的なハードウェアまたはソフトウェア・アーキテクチャーによって異なる COBOL 言語のエレメントや動作の修正方法を理解する必要があります。

関連タスク

『データ表現による違いの修正』

497 ページの『移植性に影響する環境の違いの修正』

497 ページの『言語エレメントによる違いの修正』

データ表現による違いの修正

プログラムに同じ動作をさせるためには、特定のデータ表現方法における違いを理解して、適切な処置を取る必要があります。

COBOL では、符号付きパック 10 進数および 2 進数データをメインフレーム上でも Windows ワークステーション上でも同じ方法で格納します。しかし、外部 10 進数データ、浮動小数点データ、データ、および符号なしパック 10 進数データは、デフォルトでは異なる方法で表現されます。文字データの表現は、データ項目を記述する USAGE 文節と実行時に有効なロケールによって異なる可能性があります。

大半のプログラムは、データ表現に関係なく、ワークステーション上でもメインフレーム上と同じ動作をします。

関連タスク

495 ページの『ASCII SBCS 文字と EBCDIC SBCS 文字の違いの処理』

495 ページの『IEEE データと 16 進数データの違いの処理』

496 ページの『ASCII マルチバイト と EBCDIC DBCS スtringの違いの処理』

関連参照

620 ページの『データ表現』

ASCII SBCS 文字と EBCDIC SBCS 文字の違いの処理

ASCII 文字と EBCDIC 文字のデータ表現の違いによる問題を避けるには、CHAR(EBCDIC) コンパイラー・オプションを使用します。

AIX ワークステーションは ASCII 文字セットを使用しますが、メインフレームは EBCDIC 文字セットを使用します。したがって、大半の文字が、次の表に示すように異なる 16 進値を持ちます。

表 62. ASCII 文字と EBCDIC 文字との対比

文字	ASCII の場合の 16 進値	EBCDIC の場合の 16 進値
'0' から '9'	X'30' から X'39'	X'F0' から X'F9'
'a'	X'61'	X'81'
'A'	X'41'	X'C1'
ブランク	X'20'	X'40'

また、次の表に示すように、文字データの EBCDIC 16 進値に依存するコードは、文字データが ASCII 値を持つ場合、ほとんどが失敗してしまいます。

表 63. ASCII での比較と EBCDIC での比較の対比

比較	ASCII の場合の評価	EBCDIC の場合の評価
'a' < 'A'	偽	真
'A' < '1'	偽	真
$x \geq '0'$	真の場合、 x が数字であるかどうかは示されない	真である場合、 x はおそらく数字
$x = X'40'$	x がブランクかどうかはテストされない	x かどうかはテストされる

このような違いがあるため、文字ストリングのソート結果は EBCDIC と ASCII では異なります。多くのプログラムでは、これらの違いによる影響はありませんが、プログラムが一部の文字ストリングの正確なソート順序に依存する場合は、論理エラーの可能性にも注意する必要があります。EBCDIC 照合シーケンスに依存するプログラムをワークステーションに移植する場合は、PROGRAM COLLATING SEQUENCE IS EBCDIC または COLLSEQ(EBCDIC) コンパイラー・オプションを使用して、EBCDIC 照合シーケンスを取得することができます。

関連参照

271 ページの『CHAR』

274 ページの『COLLSEQ』

IEEE データと 16 進数データの違いの処理

IEEE と 16 進数の浮動小数点データ間の表現の違いによる一般的な問題を回避するには、FLOAT(S390) コンパイラー・オプションを使用します。

AIX ワークステーションは、IEEE 形式を使用して浮動小数点データを表現します。メインフレームは、zSeries の 16 進形式を使用します。次の表に、USAGE COMP-1 データと USAGE COMP-2 データに対する、正規化浮動小数点 IEEE と正規化 16 進数の違いをまとめます。

表 64. IEEE と 16 進数の対比

仕様	COMP-1 データの IEEE	COMP-1 データの 16 進数	COMP-2 データの IEEE	COMP-2 データの 16 進数
範囲	1.17E-38* から 3.37E+38*	5.4E-79* から 7.2E+75*	2.23E-308* から 1.67E+308*	5.4E-79* から 7.2E+75*
指数表現	8 ビット	7 ビット	11 ビット	7 ビット
小数部表現	23 ビット	24 ビット	53 ビット	56 ビット
正確性のある桁数	6 桁	6 桁	15 桁	16 桁

* 値は正数でも負数でも構いません。

大半のプログラムでは、これらの違いによる問題は発生しません。ただし、データの 16 進数表現に依存するプログラムの場合は、移植時に注意が必要です。

パフォーマンスについての考慮事項: 一般に、zSeries の浮動小数点表記を使用すると、ソフトウェアが zSeries のハードウェア命令のセマンティクスをシミュレートする必要があるため、プログラムの実行速度が遅くなります。これは、特に FLOAT(S390) コンパイラ・オプションが有効で、なおかつプログラムに多数の浮動小数点計算がある場合の考慮事項です。

49 ページの『例: 数値データおよび内部表現』

関連参照

289 ページの『FLOAT』

ASCII マルチバイト と EBCDIC DBCS スtringのの違いの処理

DBCS 文字を含む英数字データ項目に対するメインフレームの動作を取得するには、CHAR(EBCDIC) コンパイラ・オプションと SOSI コンパイラ・オプションを使用します。ASCII DBCS 文字と EBCDIC DBCS 文字のデータ表現の違いによる問題を避けるには、CHAR(EBCDIC) コンパイラ・オプションを使用します。

英数字データ項目では、メインフレームの 2 バイト文字ストリング (EBCDIC DBCS 文字を含む) はシフト・コードで囲まれますが、AIX ワークステーションのマルチバイト文字ストリング (ASCII DBCS、UTF-8、または EUC 文字を含む) はシフト・コードでは囲まれません。また、同じ文字の表現に使用される 16 進値も異なります。

DBCS データ項目で、メインフレームの 2 バイト文字ストリングはシフト・コードで囲まれません。文字を表すために使用される 16 進値は、ワークステーションのマルチバイト・ストリングで同じ文字を表すために使用される 16 進値とは異なります。

大半のプログラムでは、これらの違いがあっても移植に問題はありません。ただし、プログラムが、マルチバイト・ストリングの 16 進値に依存する場合や、英数

字ストリングに 1 バイト文字とマルチバイト文字が混在することを期待する場合は、コーディングの方法に注意してください。

関連参照

271 ページの『CHAR』

305 ページの『SOSI』

移植性に影響する環境の違いの修正

AIX ワークステーションとメインフレームのプラットフォーム間におけるファイル名や制御コードの違いは、プログラムの移植に影響する可能性があります。

AIX ワークステーション上のファイル命名規約は、メインフレーム上の命名規約とはかなり異なります。COBOL ソース・プログラム内でファイル名を使用する場合は、この違いが移植性に影響を与える可能性があります。例えば、次のファイル名は、AIX ワークステーション上では有効ですが、メインフレーム (場合によっては HFS 以外) 上では無効です。

```
/users/joesmith/programs/cobol/myfile.cb1
```

大/小文字の区別: メインフレームとは異なり、AIX は大/小文字を区別します。ソース・プログラムで使用される名前 (大文字のファイル名など) は、AIX ファイル・ディレクトリー内で適切に名前を付ける必要があります。

メインフレーム上で特定の意味を持たない一部の文字は、AIX では制御文字として解釈されます。この違いにより、ASCII テキスト・ファイルが誤って処理される可能性があります。ファイルには、次のどの文字も含めないようにしてください。

- X'0A' (LF: 改行)
- X'0D' (CR: 復帰)
- X'1A' (EOF: ファイルの終わり)

装置依存 (プラットフォーム固有) の制御コードをプログラムまたはファイル内に使用する場合は、これらの制御コードをサポートしていないプラットフォームにそのプログラムまたはファイルを移植しようとする、問題が生じる可能性があります。他のプラットフォーム固有コードについても同様ですが、このようなコードはできるだけ分離して、アプリケーションを別のプラットフォームに移行する際に、容易にコードを置き換えられるようにすることが得策です。

言語エレメントによる違いの修正

一般に、移植可能な COBOL プログラムは、AIX ワークステーション上でもメインフレーム上と同様に動作します。ただし、I/O 処理で使用されるファイル状況値の違いには注意してください。

プログラムが状況キー・データ項目に応答する場合は、最初の状況キーと 2 番目の状況キーのどちらに応答するかによって、次の 2 つの問題に注意してください。

- プログラムが最初のファイル状況データ項目 (*data-name-1*) に応答する場合は、 $9n$ の範囲内で返される値がプラットフォームによって異なることに注意してください。プログラムが特定の $9n$ 値 (例えば 97) の解釈に依存する場合は、その値

の持つ意味が ワークステーション上とメインフレーム上とは異なる可能性があります。この場合は、一般的な I/O エラーとして任意の 9n 値にプログラムが応答するように修正してください。

- プログラムが 2 番目のファイル状況データ項目 (*data-name-8*) に応答する場合は、返される値がプラットフォームとファイル・システムの両方によって異なることに注意してください。例えば、STL ファイル・システムは、ワークステーション上では、メインフレーム上の VSAM ファイル・システムとはレコード構造の異なる値を返します。2 番目のファイル状況データ項目の解釈に依存するプログラムは、たいていは移植できません。

関連タスク

174 ページの『ファイル状況キーの使用』

175 ページの『ファイル・システム状況コードの使用』

関連参照

FILE STATUS 節 (COBOL for AIX 言語解説書)

メインフレーム上で実行するコードの作成

IBM COBOL for AIX を使用して新しいアプリケーションを記述すると、AIX ワークステーションを使用した場合の生産性と柔軟性が向上します。ただし、IBM メインフレーム COBOL でサポートされていない言語機能は使用しないようにする必要があります。

IBM COBOL for AIX がサポートしている言語機能の中には、IBM メインフレーム COBOL コンパイラーでは対応していないものもあります。メインフレーム上で実行するコードを AIX ワークステーション上で作成する場合は、次の機能を使用しないでください。

- DISPLAY-OF または NATIONAL-OF 組み込み関数への引数としてのコード・ページ名
- PREVIOUS 句を使用した READ ステートメント
- KEY 句で <, <=, または NOT > を使用した START ステートメント
- >>CALLINTERFACE コンパイラー指示

下の各コンパイラー・オプションは、メインフレーム・コンパイラーでは使用できません。コードをメインフレームに移植する場合は、ソース・コードに次のどのオプションも使用しないでください。

- BINARY(NATIVE) (コメントとして扱われる)
- CALLINT (コメントとして扱われる)
- CHAR(NATIVE)
- ENTRYINT (コメントとして扱われる)
- FLOAT(NATIVE)
- PROFILE

AIX と他のファイル・システムではファイルの命名規則が異なることに注意してください。ソース・プログラム内でファイル名をハードコーディングすることは避けてください。この代わりに、各プラットフォーム上で定義する簡略名を使用して、

これをメインフレームの DD 名または環境変数にマップします。その後、プログラムをコンパイルして、ソース・コードに変更を加えずにファイル名の変更に対応することができます。

特に、次の言語エレメントでのファイル参照方法を検討してください。

- ACCEPT または DISPLAY ターゲット名
- ASSIGN 文節
- COPY ステートメント (*text-name* および *library-name*)

メインフレーム上のマルチスレッド・プログラムは再帰的でなければなりません。したがって、プログラムをメインフレームに移植して、それをマルチスレッド環境で実行できるようにしたい場合は、ネストされたプログラムをコーディングしないでください。

AIX と Windows 間で移植可能なアプリケーションの作成

Windows と AIX の言語サポートはほぼ同じです。ただし、これらのプラットフォーム間にはいくつかの違いがあることに注意してください。

Windows ベースのワークステーションと AIX のワークステーション間で移植可能なアプリケーションを開発する際には、以下の項目について考慮してください。

- ソース・プログラム内にハードコーディングされたファイル名は、問題を引き起こす可能性があります。名前をハード・コーディングする代わりに、簡略名を使用して、ソース・コードを変更せずにプログラムをコンパイルできるようにします。特に、次の言語エレメントでのファイル参照方法を検討してください。
 - ACCEPT および DISPLAY ステートメント
 - ASSIGN 文節
 - COPY (*text-name* および *library-name*) ステートメント
- Windows では、整数をリトル・エンディアン形式で表現します。AIX では、整数をビッグ・エンディアン形式で保守します。したがって、整数の内部表現に依存する Windows COBOL プログラムは、大抵は AIX に移植できません。このような内部表現に依存するプログラムを記述することは避けてください。
- Windows では、クラス国別データ項目をリトル・エンディアン形式で表現します。AIX では、クラス国別データ項目をビッグ・エンディアン形式で表現します。したがって、このようなデータ項目の内部表現に依存している COBOL プログラムは、そのほとんどが COBOL for Windows と COBOL for AIX との間で移植できません。このような内部表現に依存するプログラムを記述することは避けてください。

クラス国別データ項目の内部表現に依存しないプログラムは、たいていの場合、COBOL for Windows と COBOL for AIX との間で移植できますが、ファイル・データをターゲット・プラットフォームの表現に変換する必要があります。

第 26 章 サブプログラムの使用

多くのアプリケーションは、ともにリンクされた別々にコンパイルされたプログラムから構成されます。各プログラムが相互に呼び出す場合は、これらは通信できる必要があります。これらのプログラムは制御権を渡す必要があります、また通常は共通データに対するアクセス権を必要とします。

COBOL プログラムどうしが互いの内部にネストされている場合は、プログラム間でのやり取りも可能です。1つのアプリケーションに必要なサブプログラムをすべて1つのソース・ファイルに含めることができるため、必要なコンパイルは1回だけで済みます。

関連概念

『メインプログラム、サブプログラム、および呼び出し』

関連タスク

502 ページの『メインプログラムまたはサブプログラムの終了と再入』

503 ページの『ネストされた COBOL プログラムの呼び出し』

506 ページの『ネストなし COBOL プログラムの呼び出し』

508 ページの『COBOL および C/C++ プログラム間の呼び出し』

515 ページの『再帰呼び出しの実行』

メインプログラム、サブプログラム、および呼び出し

COBOL プログラムが実行単位の最初のプログラムであれば、その COBOL プログラムはメインプログラムです。それ以外の場合は、そのプログラムも実行単位内の他のすべての COBOL プログラムも、サブプログラムです。特定のソース・コードのステートメントまたはオプションが、COBOL プログラムをメインプログラムまたはサブプログラムとして識別することはありません。

COBOL プログラムがメインプログラムであるかサブプログラムであるかは、次の2つの理由により重要になることもあります。

- プログラム終了処理ステートメントの影響
- 戻った後に再入する際のそのプログラムの状態

PROCEDURE DIVISION で、あるプログラムから別のプログラム (通常サブプログラムと呼ばれる) を呼び出すことができ、この呼び出し先プログラム自体からも別のプログラムを呼び出すことができます。別のプログラムを呼び出すプログラムは呼び出し側プログラムと呼ばれ、そのプログラムが呼び出すプログラムは呼び出し先プログラムと呼ばれます。呼び出し先プログラムの処理が完了すると、そのプログラムは制御権を呼び出し側プログラムに戻すか、実行単位を終了することができます。

呼び出し先 COBOL プログラムは、PROCEDURE DIVISION の先頭で実行を開始します。

関連タスク

- 『メインプログラムまたはサブプログラムの終了と再入』
- 503 ページの『ネストされた COBOL プログラムの呼び出し』
- 506 ページの『ネストなし COBOL プログラムの呼び出し』
- 508 ページの『COBOL および C/C++ プログラム間の呼び出し』
- 515 ページの『再帰呼び出しの実行』

メインプログラムまたはサブプログラムの終了と再入

プログラムが最後に使用された状態になるのかまたは初期状態になるのか、および戻り先がどの呼び出し元になるのかは、使用するステートメントにより異なる可能性があります。

メインプログラムで実行を終了させるには、メインプログラムで、STOP RUN または GOBACK ステートメントをコーディングする必要があります。STOP RUN は実行単位を終了して、メインプログラムおよびメインプログラムに呼び出されたサブプログラムによって開かれたすべてのファイルを閉じます。制御は、メインプログラムの呼び出し元に返されます。この呼び出し元は多くの場合、オペレーティング・システムです。GOBACK は、メインプログラムで同じ効果を発揮します。メインプログラムで実行された EXIT PROGRAM には効果がありません。

EXIT PROGRAM、GOBACK、または STOP RUN ステートメントを使用して、サブプログラムを終了することができます。EXIT PROGRAM または GOBACK ステートメントを使用した場合には、制御は、実行単位を終了させることなく、サブプログラムの直接の呼び出し元に戻ります。呼び出されるプログラムの中に次に実行可能なステートメントがない場合、暗黙の EXIT PROGRAM ステートメントが生成されます。サブプログラムを STOP RUN ステートメントを使用して終了させた場合には、その効果は、メインプログラムの場合と同じです。実行単位のすべての COBOL プログラムが終了し、制御はメインプログラムの呼び出し元に戻ります。

サブプログラムは通常、EXIT PROGRAM または GOBACK で終了されると、最後に使用された状態になります。サブプログラムが実行単位の中で次に呼び出されると、PERFORM ステートメントの戻り値が初期値にリセットされることを除けば、その内部値は終了時のまま残されます(それに対して、メインプログラムは呼び出されるたびに初期化されます)。

プログラムが初期状態になるのは、次のような場合です。

- 動的に呼び出され、その後取り消されるサブプログラムは、次に呼び出されると初期状態になります。
- INITIAL 属性を持つプログラムは、呼び出されるたびに初期状態になります。
- LOCAL-STORAGE SECTION で定義されているデータ項目は、プログラムが呼び出されるたびに、VALUE 節で指定されている初期状態にリセットされます。

関連概念

- 14 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』

関連タスク

- 503 ページの『ネストされた COBOL プログラムの呼び出し』
- 515 ページの『再帰呼び出しの実行』

ネストされた COBOL プログラムの呼び出し

ネストされたプログラムを呼び出すことによって、構造化プログラミング技法を使用したアプリケーションを作成することができます。また、データ項目を不用意に変更しないようにするために、PERFORM プロシージャの代わりに、ネストされたプログラムを呼び出すことができます。ネストされたプログラムの呼び出しには、CALL *literal* ステートメントまたは CALL *identifier* ステートメントを使用します。

ネストされた プログラムは、それを直接収容するプログラムからのみ呼び出すことができます。ただし、ネストされた プログラムをその PROGRAM-ID 段落で COMMON として識別している場合は別です。その場合、共通プログラム は、共通プログラムと同じプログラム内の任意の (直接的または間接的) ネストされた プログラムから呼び出すことができます。COMMON として識別できるのは、ネストされた プログラムだけです。再帰呼び出しはできません。

ネストされたプログラム構造を使用する際には、以下の指針に従ってください。

- 各プログラムに IDENTIFICATION DIVISION をコーディングします。他の部はすべてオプションです。
- 状況に応じて、それぞれの ネストされた プログラムの名前を固有にしてください。ネストされた プログラムの名前は (名前の有効範囲に関する関連参照で記述されているように) 固有である必要はありませんが、名前を固有にしておくこと、アプリケーションの保守が一層容易になります。ネストされた プログラムの名前として、任意の有効なユーザー定義語または英数字リテラルを使用できます。
- 必要となる可能性のある CONFIGURATION SECTION 記入項目は、最外部のプログラムでコーディングします。ネストされた プログラムが CONFIGURATION SECTION を持つことはできません。
- それぞれの ネストされた プログラムを収容プログラム内に組み込む場合、収容プログラムの END PROGRAM マーカーの直前に組み込んでください。
- END PROGRAM マーカーを使用して、ネストされた プログラムと収容しているプログラムを終了させます。

関連概念

『ネストされたプログラム』

関連参照

505 ページの『名前の有効範囲』

ネストされたプログラム

COBOL プログラムには、他の COBOL プログラムをネスト すること、つまり、他の COBOL プログラムを含めることができます。ネストされたプログラム自体にも他のプログラムを含められます。ネストされたプログラムは、プログラムに直接的に含めることも、間接的に含めることもできます。

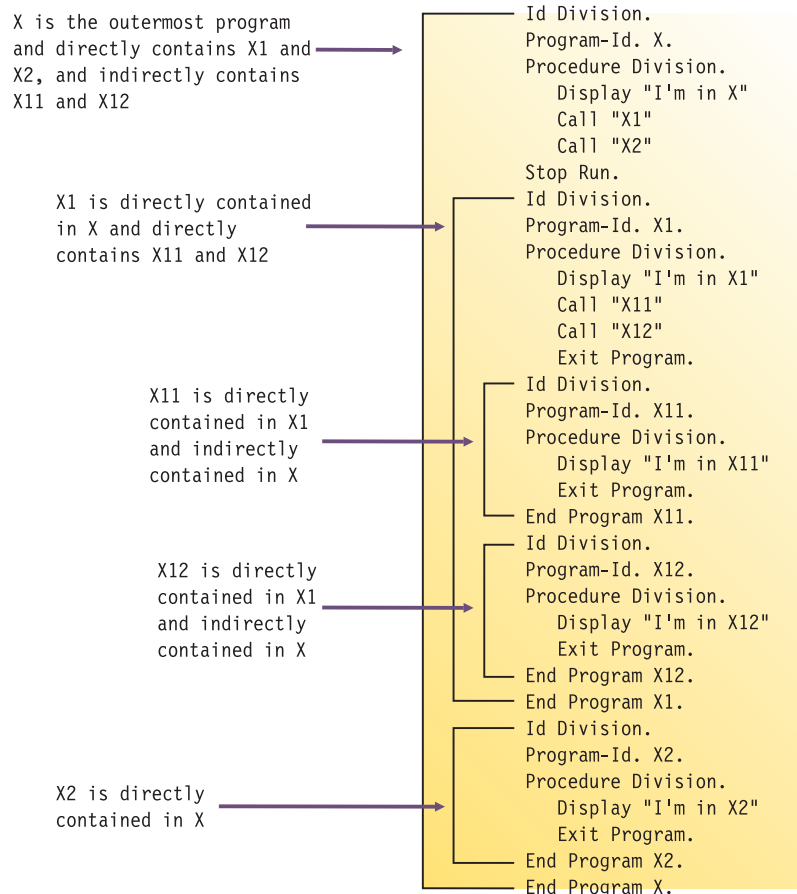
呼び出されるプログラムをネストすることには、主に次の 4 つの長所があります。

- ネストされたプログラムは、モジュラー機能の作成と構造化プログラミング手法の保守を行う方法を提供します。これらは、(PERFORM ステートメントを使用して)

プロシージャーを実行するために同じように使用することができます。ただし、制御フローはより構造化されたものになり、ローカル・データ項目を保護することができます。

- ネストされたプログラムにより、プログラムをアプリケーションに組み込む前にデバッグすることができます。
- ネストされたプログラムを使用すると、コンパイラーを一度起動するだけでアプリケーションをコンパイルできます。
- COBOL CALL ステートメントのさまざまな形式の中で、ネストされたプログラムへの呼び出しは最高のパフォーマンスを発揮します。

次の例は、直接的および間接的に含まれたプログラムのあるネストされた構造を説明したものです。



505 ページの『例: ネストされたプログラムの構造』

関連タスク

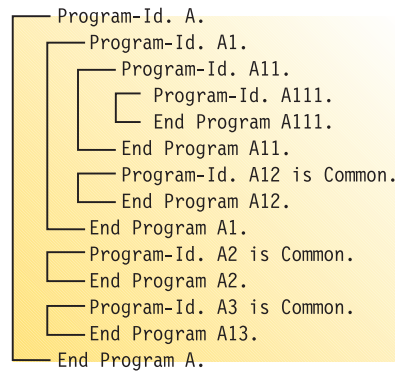
503 ページの『ネストされた COBOL プログラムの呼び出し』

関連参照

505 ページの『名前の有効範囲』

例: ネストされたプログラムの構造

次の例は、一部のネストされたプログラムが COMMON として識別された、ネストされた構造を示しています。



次の表に、上記の例で示した構造の呼び出し階層について説明しています。プログラム A12、A2、および A3 は COMMON として識別されており、それらに関連する呼び出しはそれぞれ異なります。

対象となるプログラム	対象プログラムが呼び出せるプログラム	対象プログラムを呼び出せるプログラム
A	A1、A2、A3	なし
A1	A11、A12、A2、A3	A
A11	A111、A12、A2、A3	A1
A111	A12、A2、A3	A11
A12	A2、A3	A1、A11、A111
A2	A3	A、A1、A11、A111、A12、A3
A3	A2	A、A1、A11、A111、A12、A2

この例では、次の点に注目してください。

- A2 は A1 を呼び出すことはできません。A1 は共通ではなく、A2 に含まれていないからです。
- A1 は A2 を呼び出すことができます。A2 は共通であるからです。

名前の有効範囲

ネストされた構造における名前は、ローカルとグローバルの 2 つのクラスに分けられます。名前を宣言しているプログラムの有効範囲を超えてその名前が知られているかどうか、クラスによって判別されます。プログラム内で名前が参照された後で、特定の検索シーケンスで名前の宣言を見つけます。

ローカル名

他に宣言されていない限り、名前はローカルです (プログラム名を除く)。ローカル名は、それが宣言されているプログラム内からのみ見る、またはアクセスすることができます。含まれているプログラムおよび収容プログラムが、ローカル名を見たり、アクセスすることはできません。

グローバル名

グローバルである (GLOBAL 節を使用して示された) 名前は、その名前が宣言されているプログラムと、そのプログラムに直接的および間接的に含まれているすべてのプログラムから可視でありアクセス可能です。したがって、含まれているプログラムは、単に項目の名前を参照するだけで、収容プログラムからの共通データおよびファイルを共用することができます。

グローバル項目に従属するすべての項目 (条件名と指標を含む) は、自動的にグローバルになります。

各宣言が異なるプログラムの中で現れるのであれば、GLOBAL 節を使用して同じ名前を複数回宣言することができます。同じ収容構造の異なるプログラムに同じ名前を持たせることによって、ネストされた構造における名前のマスキングや隠蔽が可能であることに注意してください。ただし、このようなマスキングによって、名前宣言の検索時に問題が生じることがあります。

名前の宣言の探索

プログラム内で名前が参照されると、その名前の宣言を見つける探索が行われます。探索は、参照が含まれるプログラムで始まり、一致する名前が見つかるまで、順番に収容プログラムへ移って外側へ続けられます。探索は次のプロセスに従います。

1. そのプログラム内の宣言が検索されます。
2. 一致する名前が見つからない場合は、連続する外側の収容プログラムの中で、グローバル宣言だけが検索されます。
3. 一致する最初の名前が検出されると、探索は終了します。一致が検出されない場合、エラーが存在します。

探索はグローバル名に関するものであり、データ項目やファイル結合子などの名前に関連した特定の型のオブジェクトに関するものではありません。オブジェクトの型に関係なく、何らかの一致する名前が見つかり探索は停止します。宣言されたオブジェクトが予期されたものと違う場合は、エラー状態が存在します。

ネストなし COBOL プログラムの呼び出し

COBOL プログラムは、呼び出し元と同じ実行可能モジュールにリンクされた (スタティック・リンク) サブプログラムか、または 共用ライブラリー で提供された (ダイナミック・リンク) サブプログラムを呼び出すことができます。COBOL for AIX では、共用ライブラリー からターゲット・サブプログラムを実行時に解決することができます。

ターゲット・プログラムを静的にリンクする場合は、プログラムが呼び出し元の実行可能モジュールの一部になり、呼び出し元とともにロードされます。ターゲット・プログラムを動的にリンクするか、実行時に呼び出しを解決する場合は、ターゲット・プログラムがライブラリー内に提供され、呼び出し元のロード時かターゲット・プログラムの呼び出し時にロードされます。

COBOL の *CALL literal* に対してサブプログラムの動的または静的リンクが実行されます。実行時の解決は、COBOL *CALL identifier* に対しては常に実行され、*CALL literal* に対しては DYNAM オプションが有効な場合に実行されます。

| **制限:** 1 つのアプリケーションで 32 ビット COBOL プログラムと 64 ビット
| COBOL プログラムを混用することはできません。アプリケーション内のプログラ
| ム・コンポーネントはすべて、同じ ADDR コンパイラー・オプション設定を使用し
| てコンパイルしなければなりません。

関連概念

『CALL identifier および CALL literal』

535 ページの『スタティック・リンクおよび共用ライブラリー』

関連参照

266 ページの『ADDR』

280 ページの『DYNAM』

CALL ステートメント (*COBOL for AIX* 言語解説書)

CALL identifier および CALL literal

CALL identifier を使用すると、呼び出し時に常にターゲット・プログラムがロードされます。この *identifier* には、実行時にネストなしサブプログラム名を含むデータ項目が入ります。*CALL literal* は、静的または動的に解決することができます。この *literal* には、ネストなしターゲット・サブプログラムの明示的な名前が入ります。

CALL identifier については、共用ライブラリー の名前が、ターゲット入り口点の名前と一致している必要があります。

CALL literal については、NODYNAM コンパイラー・オプションが有効な場合は、スタティック・リンクとダイナミック・リンクのどちらでも実行できます。DYNAM が有効な場合は、*CALL literal* が *CALL identifier* と同じ方法で解決されます。つまり、ターゲット・サブプログラムは呼び出し時にロードされ、共用ライブラリー の名前とターゲット入り口点の名前が一致していなければなりません。

これらの呼び出し定義は、COBOL プログラムがネストなしプログラムを呼び出す場合にのみ適用されます。COBOL プログラムがネストされたプログラムを呼び出す場合は、この呼び出しは、システムが介入せずに、コンパイラーによって解決されます。

| **制約事項:** アプリケーション内の 2 つ以上の別々にリンクされた実行可能モジュール
| は、同一のネストなしサブプログラムを静的に呼び出さないでください。

関連概念

535 ページの『スタティック・リンクおよび共用ライブラリー』

関連タスク

506 ページの『ネストなし COBOL プログラムの呼び出し』

関連参照

280 ページの『DYNAM』

CALL ステートメント (COBOL for AIX 言語解説書)

COBOL および C/C++ プログラム間の呼び出し

C/C++ で記述された関数は、COBOL プログラムから呼び出すことができ、また、COBOL プログラムを C/C++ 関数から呼び出すこともできます。

言語間アプリケーションでは、64 ビット COBOL プログラムを 64 ビット C/C++ 関数と結合したり、32 ビット COBOL プログラムを 32 ビット C/C++ 関数と結合したりできます。

制約事項: 1 つのアプリケーションで 32 ビット・コンポーネントと 64 ビット・コンポーネントを混用することはできません。

COBOL と C++ の間で行われる言語間通信: COBOL と C++ を混用する言語間アプリケーションでは、以下のガイドラインに従ってください。

- C++ から呼び出される COBOL プログラムの関数プロトタイプと、COBOL から呼び出される C++ 関数に、extern "C" を指定します。
- COBOL では、通常の C++ パラメーター規則に適合するように BY VALUE パラメーターを使用します。
- C++ では、COBOL BY REFERENCE 規則に適合するように参照パラメーターを使用します。

以下で参照される規則とガイドラインでは、これらの言語間呼び出しの実行方法について詳しく説明します。

参照される各セクションで「C/C++」を無条件で参照する場合、参照先は IBM XL C/C++ を指しています。

関連タスク

367 ページの『CICS での COBOL および C/C++ 間の呼び出し』

509 ページの『環境の初期設定』

509 ページの『COBOL と C/C++ 間でのデータの受け渡し』

510 ページの『スタック・フレームの縮小と実行単位またはプロセスの終了』

517 ページの『第 27 章 データの共用』

XL C/C++ for AIX 最適化およびプログラミング・ガイド

関連参照

266 ページの『ADDR』

511 ページの『COBOL および C/C++ のデータ型』

環境の初期設定

C/C++ と COBOL の間で呼び出しを行うには、ターゲット環境を適切に初期設定する必要があります。

C/C++ で記述されたメインプログラムが、COBOL プログラムに対して複数の呼び出しを行う場合は、次のいずれかの方法を使用します。

- COBOL プログラムを呼び出す前に、C/C++ プログラムに COBOL 環境を事前初期設定します。これにより最適なパフォーマンスが得られるため、この方法をお勧めします。
- COBOL プログラムを、COBOL を呼び出す C/C++ ルーチンの一部ではない実行可能ファイル内に入れます。その後で、COBOL メインプログラムを呼び出すたびに、C/C++ プログラムで以下のステップを実行します。
 1. プログラムをロードします。
 2. プログラムを呼び出します。
 3. プログラムをアンロードします。

関連概念

555 ページの『第 30 章 COBOL ランタイム環境の事前初期設定』

COBOL と C/C++ 間でのデータの受け渡し

COBOL データ型には、C/C++ データ型と同じものがありますが、同じでないものもあります。COBOL プログラムと C/C++ の関数間でデータを受け渡す場合は、データ交換を適切なデータ型のみで行うようにしてください。

デフォルトでは、COBOL は、引数を BY REFERENCE で受け渡します。引数 BY REFERENCE を受け渡す場合、C/C++ はその引数を指すポインターを取得します。引数を BY VALUE で受け渡す場合は、COBOL は実引数を渡します。BY VALUE は、次のデータ型にしか使用できません。

- 英数字
- USAGE NATIONAL 文字
- BINARY
- COMP
- COMP-1
- COMP-2
- COMP-4
- COMP-5
- FUNCTION-POINTER
- OBJECT REFERENCE
- POINTER
- PROCEDURE-POINTER

511 ページの『例: C 関数を呼び出す COBOL プログラム』

512 ページの『例: COBOL プログラムによって呼び出される C プログラムと

COBOL を呼び出す C/C++ プログラム』
514 ページの『例: C++ 関数を呼び出す COBOL プログラム』

関連タスク

517 ページの『第 27 章 データの共用』

関連参照

511 ページの『COBOL および C/C++ のデータ型』

スタック・フレームの縮小と実行単位またはプロセスの終了

ある言語で関数を呼び出すと、別の言語のプログラム・スタック・フレームが縮小してしまうような呼び出しは避けてください。

このガイドラインに該当する状況例を次に示します。

- ある言語で記述された一部のアクティブ・スタック・フレームを縮小する場合に、縮小されるスタック・フレーム内に別の言語で記述されたアクティブ・スタック・フレーム (C/C++ longjmp()) があるような場合。
- ある言語で記述されたスタック・フレームがアクティブなときに、COBOL の STOP RUN や C/C++ の exit() または _exit() などを発行すると、別の言語で記述された実行単位またはプロセスが終了します。このような場合は、プログラムの呼び出し側に戻ることで、呼び出されたプログラムが終了するような方法で、アプリケーションを構築してください。

C/C++ の longjmp()、COBOL の STOP RUN、C/C++ の exit() または _exit() 呼び出しが使用できるのは、操作を開始する言語以外の言語で記述されたアクティブ・スタック・フレームが縮小されない場合に限られます。縮小や終了を開始しない言語の場合は、その他に次のような悪影響が生じる可能性があります。

- 当該言語の通常の cleanup または exit 関数 (実行単位終了時の COBOL によるファイルのクローズ、強制終了された言語による動的獲得リソースのクリーンアップなど) が実行されない可能性があります。
- ユーザー指定の出口または関数 (デストラクターや C/C++ の atexit() 関数など) が出口や終了に対して呼び出されない可能性があります。

一般に、スタック・フレームの実行中に発生した例外は、その例外を招いた言語の規則に従って処理されます。COBOL のインプリメンテーションで COBOL 言語セマンティクスをサポートする場合は、システム・サービスを介した例外の代行受信に依存しないため、TRAP(OFF) ランタイム・オプションを指定して、非 COBOL 言語の例外処理セマンティクスを有効にすることができます。

COBOL for AIX は、COBOL ランタイム環境の初期設定時の例外環境を保存し、COBOL 環境の終了時にはその例外環境を復元します。COBOL では、インターフェース言語とツールが同じ規約に従うことを期待します。

関連参照

328 ページの『TRAP』

COBOL および C/C++ のデータ型

次の表に、COBOL および C/C++ で使用可能なデータ型の対応を示します。

表 65. COBOL および C/C++ のデータ型

C/C++ データ型	COBOL データ・タイプ
wchar_t	USAGE NATIONAL (PICTURE N)
char	PIC X
signed char	相当する COBOL データ型なし
unsigned char	相当する COBOL データ型なし
short signed int	PIC S9-S9(4) COMP-5。TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
short unsigned int	PIC 9-9(4) COMP-5。TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
long int	PIC 9(5)-9(9) COMP-5。TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
long long int	PIC 9(10)-9(18) COMP-5。TRUNC(BIN) コンパイラー・オプションを使用した場合は、COMP、COMP-4、または BINARY が使用可能。
float	COMP-1
double	COMP-2
enumeration	レベル 88 に類似するが、同一ではない。
char(n)	PICTURE X(n)
array pointer (*) to type	相当する COBOL データ型なし
pointer(*) to function	PROCEDURE-POINTER または FUNCTION-POINTER

関連タスク

509 ページの『COBOL と C/C++ 間でのデータの受け渡し』

関連参照

312 ページの『TRUNC』

例: C 関数を呼び出す COBOL プログラム

以下の例では、CALL ステートメントを使用して C 関数を呼び出す COBOL プログラムを示しています。

この例では、以下の概念を示しています。

- この CALL ステートメントは、呼び出し先プログラムが COBOL と C のどちらで記述されるかを示すものではありません。
- COBOL では、大小混合名を持つプログラムの呼び出しが可能です。
- 引数は、さまざまな方法で C プログラムに渡すことができます (例: BY REFERENCE または BY VALUE)。

- 非 void の C 関数を呼び出す CALL ステートメントでは、関数の戻り値を宣言する必要があります。
- COBOL データ型を適切な C データ型と対応させる必要があります。

```

CBL PGMNAME(MIXED)
* This compiler option allows for case-sensitive names for called programs.
*
IDENTIFICATION DIVISION.
PROGRAM-ID. "COBCALLC".
*
DATA DIVISION.
WORKING-STORAGE SECTION.
01 N4          PIC 9(4)  COMP-5.
01 NS4        PIC S9(4) COMP-5.
01 N9          PIC 9(9)  COMP-5.
01 NS9        PIC S9(9) COMP-5.
01 NS18       USAGE COMP-2.
01 D1          USAGE COMP-2.
01 D2          USAGE COMP-2.
01 R1.
   02 NR1      PIC 9(8)  COMP-5.
   02 NR2      PIC 9(8)  COMP-5.
   02 NR3      PIC 9(8)  COMP-5.
PROCEDURE DIVISION.
MOVE 123 TO N4
MOVE -567 TO NS4
MOVE 98765432 TO N9
MOVE -13579456 TO NS9
MOVE 222.22 TO NS18
DISPLAY "Call MyFun with n4=" N4 " ns4=" NS4 " N9=" n9
DISPLAY "          ns9=" NS9 " ns18=" NS18
* The following CALL illustrates several ways to pass arguments.
*
CALL "MyFun" USING N4 BY VALUE NS4 BY REFERENCE N9 NS9 NS18
MOVE 1024 TO N4
* The following CALL returns the C function return value.
*
CALL "MyFunR" USING BY VALUE N4 RETURNING NS9
DISPLAY "n4=" N4 " and ns9= n4 times n4= " NS9
MOVE -357925680.25 TO D1
CALL "MyFunD" USING BY VALUE D1 RETURNING D2
DISPLAY "d1=" D1 " and d2= 2.0 times d2= " D2
MOVE 11111 TO NR1
MOVE 22222 TO NR2
MOVE 33333 TO NR3
CALL "MyFunV" USING R1
STOP RUN.

```

関連タスク

509 ページの『COBOL と C/C++ 間でのデータの受け渡し』

関連参照

CALL ステートメント (COBOL for AIX 言語解説書)

例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プログラム

次の例は、呼び出し先の C 関数が、COBOL CALL ステートメントで渡された順に引数を受け取ることを示しています。

ファイル MyFun.c には、次のソース・コードが含まれています。このソース・コードは COBOL プログラム tprog1 を呼び出します。

```
#include <stdio.h>
extern void TPROG1(double *);
void
MyFun(short *ps1, short s2, long *k1, long *k2, double *m)
{
    double x;
    x = 2.0*( *m);
    printf("MyFun got s1=%d s2=%d k1=%d k2=%d x=%f\n",
           *ps1, s2, *k1,*k2, x);
}

long
MyFunR(short s1)
{
    return(s1 * s1);
}

double
MyFunD(double d1)
{
    double z;
    /* calling COBOL */
    z = 1122.3344;
    (void) TPROG1(&z);
    /* returning a value to COBOL */
    return(2.0 * d1);
}

void
MyFunV(long *pn)
{
    printf("MyFunV got %d %d %d\n", *pn, *(pn+1), *(pn+2));
}
```

MyFun.c は、次の関数で構成されています。

MyFun

さまざまな引数を渡す方法を示します。

MyFunR

long 型変数を渡して戻す方法を示します。

MyFunD

C が COBOL プログラムを呼び出す方法と、double 型変数を渡して戻す方法を示します。

MyFunV

ポインタをレコードに渡す方法と、C プログラム内のレコードの項目にアクセスする方法を示します。

例: C プログラムによって呼び出される COBOL プログラム

次の例に、C プログラムによって呼び出される COBOL プログラムの記述方法を示します。

COBOL プログラム tprog1 がプログラム MyFun.c 内の C 関数 MyFunD によって呼び出されます (『512 ページの『例: COBOL プログラムによって呼び出される C

プログラムと COBOL を呼び出す C/C++ プログラム』を参照)。呼び出される COBOL プログラムには、以下のソース・コードが含まれます。

```
*
  IDENTIFICATION DIVISION.
  PROGRAM-ID. TPROG1.
*
  DATA DIVISION.
  LINKAGE SECTION.
*
  01 X                      USAGE COMP-2.
*
  PROCEDURE DIVISION USING X.
    DISPLAY "TPROG1 got x= " X
    GOBACK.
```

関連タスク

508 ページの『COBOL および C/C++ プログラム間の呼び出し』

例: コンパイルおよび実行の結果の例

この例は、COBOL プログラム `cobcallc.cbl` と `tprog.cbl`、および C プログラム `MyFun.c` をコンパイル、リンク、および実行する方法を示し、プログラムの実行結果を表示しています。

`cobcallc.cbl`、`tprog.cbl`、および `MyFun.c` をコンパイルしてリンクするには、以下のコマンドを発行します。

1. `xlc -c MyFun.c`
2. `cob2 cobcallc MyFun.o tprog1.cbl -o cobcallc`

プログラムを実行するには、コマンド `cobcallc` を発行します。結果は次のようになります。

```
call MyFun with n4=00123 ns4=-00567 n9=0098765432
               ns9=-0013579456 ns18=.222220000000000000E 03
MyFun got s1=123 s2=-567 k1=98765432 k2=-13579456 x=444.440000
n4=01024 and ns9= n4 times n4= 0001048576
TPROG1 got x= .112233440000000000E+04
d1=-.357925680250000000E+09 and d2= 2.0 times d2= -.715851360500000000E+09
MyFunV got 11111 22222 33333
```

511 ページの『例: C 関数を呼び出す COBOL プログラム』

512 ページの『例: COBOL プログラムによって呼び出される C プログラムと COBOL を呼び出す C/C++ プログラム』

513 ページの『例: C プログラムによって呼び出される COBOL プログラム』

例: C++ 関数を呼び出す COBOL プログラム

以下の例では、引数 BY REFERENCE を渡す CALL ステートメントを使用して、C++ 関数を呼び出す COBOL プログラムを示しています。

この例では、以下の概念を示しています。

- この CALL ステートメントは、呼び出し先プログラムが COBOL と C++ のどちらで記述されるかを示すものではありません。

- 非 void の C++ 関数を呼び出す CALL ステートメントでは、関数の戻り値を宣言する必要があります。
- COBOL データ型は、適切な C++ データ型にマップされなければなりません。
- C++ 関数は、extern "C" として宣言する必要があります。
- COBOL 引数は、BY REFERENCE で受け渡されます。C++ 関数は、参照パラメータを使用してその引数を受け取ります。

COBOL プログラム driver:

```

cb1 pgmname(mixed)
Identification Division.
Program-Id.
"driver".
Data division.
Working-storage section.
01 A pic 9(8) binary value 11111.
01 B pic 9(8) binary value 22222.
01 R pic 9(8) binary.
Procedure Division.
    Display "Hello World, from COBOL!"
    Call "sub" using by reference A B
        returning R
    Display R
    Stop Run.

```

C++ 関数 sub:

```

#include <iostream.h>
extern "C" long sub(long& A, long& B) {
    cout << "Hello from C++" << endl;
    return A + B;
}

```

出力:

```

Hello World, from COBOL!
Hello from C++
00033333

```

関連タスク

509 ページの『COBOL と C/C++ 間でのデータの受け渡し』

関連参照

CALL ステートメント (*COBOL for AIX* 言語解説書)

再帰呼び出しの実行

呼び出し先プログラムは、その呼び出し側を直接または間接に実行することができます。例えば、プログラム X がプログラム Y を呼び出し、プログラム Y がプログラム Z を呼び出し、そして、プログラム Z がプログラム X を呼び出します。このような呼び出しを再帰的と言います。

再帰呼び出しを行うには、再帰的に呼び出されるプログラムの PROGRAM-ID 段落で RECURSIVE 節をコーディングするか、THREAD コンパイラー・オプションを指定する必要があります。THREAD コンパイラー・オプションが指定されておらず、かつ PROGRAM-ID 段落で RECURSIVE 節がコーディングされていない COBOL プログラムを再帰的に呼び出そうとすると、実行単位が異常終了します。

関連タスク

6 ページの『プログラムを再帰的として識別する』

関連参照

311 ページの『THREAD』

PROGRAM-ID 段落 (COBOL for AIX 言語解説書)

戻りコードの受け渡し

RETURN-CODE 特殊レジスターを使用して、別々にコンパイルされたプログラム間で情報を受け渡すことができます。

呼び出し側に戻る前に RETURN-CODE を呼び出し先プログラムに設定して、戻される値を呼び出し側プログラムでテストすることができます。この手法は、通常は呼び出し先プログラムの成功レベルを示すのに使用します。例えば、ゼロの RETURN-CODE を使用して、呼び出し先プログラムが正常に実行されたことを示すことができます。

非スレッド化サポートの場合: 非スレッド化サポートのメインプログラムが正常に終了すると、RETURN-CODE の値がオペレーティング・システムにユーザー戻りコードとして渡されます。ただし、AIX は、ユーザー戻りコード値を 0 から 255 までに制限します。このため、プログラムが終了するときに例えば RETURN-CODE に 258 が含まれている場合、AIX はサポートされている範囲内の値を循環し、結果としてユーザー戻りコードが 2 になります。

非スレッド化サポートのプログラムが回復不能な例外を検出すると、実行単位が終了し、ユーザー戻りコードは 128 にシグナル番号を加えた値に設定されます。

スレッド化サポートの場合: スレッド化サポートがある場合、RETURN-CODE の値がオペレーティング・システムに渡されることはありません。このような状況は、次のいずれかの場合に発生します。

- メインプログラムが正常に終了したとき
- プログラムが回復不能な例外が検出したとき。(この場合、実行単位ではなく、プログラムが実行されているスレッドが終了します。)

関連参照

RETURN-CODE 特殊レジスター (COBOL for AIX 言語解説書)

第 27 章 データの共用

実行単位が互いに呼び出す、別々にコンパイルされた複数のプログラムで構成される場合、プログラムは互いに通信できなければなりません。また、通常、共通データへのアクセス権限も必要です。

ここでは、他のプログラムとデータを共用できるプログラムの作成方法を説明します。ここで言うサブプログラムは、別のプログラムが呼び出す任意のプログラムのことです。

関連タスク

『データの受け渡し』

520 ページの『LINKAGE SECTION のコーディング』

521 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

526 ページの『プロシージャ・ポインターと関数ポインターの使用』

526 ページの『戻りコード情報の引き渡し』

527 ページの『CALL . . . RETURNING の指定』

527 ページの『EXTERNAL 節によるデータの共用』

528 ページの『プログラム間でのファイルの共用 (外部ファイル)』

532 ページの『コマンド行引数の使用』

482 ページの『Java とのデータ共用』

データの受け渡し

プログラム間のデータの受け渡し方法には 3 つあり、それらから選択できます。BY REFERENCE、BY CONTENT、または BY VALUE。

BY REFERENCE

サブプログラムは、データのコピーを処理するのではなく、呼び出し側プログラムのストレージ内のデータ項目を参照して処理します。BY REFERENCE は、パラメーターに関して 3 つの方法のどれも指定されておらず、暗黙指定されてもいない場合の、パラメーターの想定引き渡しメカニズムです。

BY CONTENT

呼び出し側プログラムは、*literal* または *identifier* の内容だけを渡します。呼び出し先プログラムは、呼び出し側プログラム内の *literal* または *identifier* の値を変更できません。たとえ、*literal* または *identifier* を受け取ったデータ項目を変更する場合でも変更できません。

BY VALUE

呼び出し側プログラムまたはメソッドは、送り出しデータ項目を参照せず、*literal* または *identifier* の値を渡します。呼び出されるプログラムまたはメソッドは、その中のパラメーターを変更できます。しかし、サブプログラムまたはメソッドは送り出しデータ項目の一時コピーへのアクセス権しか持っていないので、どの変更内容も呼び出し側プログラムの引数に影響を与えません。

プログラムでどのようにデータを処理したいかに基づいて、上記のデータ受け渡し方法のどれを使用するかを決定してください。

表 66. CALL ステートメントでデータを渡す方法

コード	目的	コメント
CALL . . . BY REFERENCE <i>identifier</i>	呼び出し側プログラムの CALL ステートメントの引数の定義と呼び出されるプログラムのパラメーターの定義に、同じメモリーを共用させる。	サブプログラムがパラメーターに対して行う変更は、呼び出し側プログラムの引数に影響を与えません。
CALL . . . BY REFERENCE ADDRESS OF <i>identifier</i>	<i>identifier</i> のアドレスを呼び出し先プログラムに渡します。ここで、 <i>identifier</i> は LINKAGE SECTION 内の項目です。	サブプログラムがアドレスに対して行う変更は、呼び出し側プログラム内のアドレスに影響を与えません。
CALL . . . BY CONTENT ADDRESS OF <i>identifier</i>	<i>identifier</i> のアドレスのコピーを、呼び出し先プログラムに渡します。	アドレスのコピーに任意の変更を加えても <i>identifier</i> のアドレスには影響しませんが、アドレスのコピーを使用する <i>identifier</i> を変更すると <i>identifier</i> が変更されます。
CALL . . . BY CONTENT <i>identifier</i>	ID のコピーをサブプログラムに渡す。	サブプログラムによってパラメーターを変更しても、呼び出し側の ID には影響しません。
CALL . . . BY CONTENT <i>literal</i>	呼び出し先プログラムにリテラル値のコピーを渡す。	
CALL . . . BY CONTENT LENGTH OF <i>identifier</i>	データ項目の長さのコピーを渡す。	呼び出し側プログラムは、その LENGTH 特殊レジスターから <i>identifier</i> の長さを渡します。
次のような BY REFERENCE と BY CONTENT の組み合わせ: CALL 'ERRPROC' USING BY REFERENCE A BY CONTENT LENGTH OF A.	データ項目とその長さのコピーの両方をサブプログラムに渡す。	
CALL . . . BY VALUE <i>identifier</i>	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用するプログラムにデータを渡す。	ID のコピーがパラメーター・リストとして直接渡されます。
CALL . . . BY VALUE <i>literal</i>	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用するプログラムにデータを渡す。	リテラルのコピーがパラメーター・リストとして直接渡されます。
CALL . . . BY VALUE ADDRESS OF <i>identifier</i>	呼び出し先プログラムに <i>identifier</i> のアドレスを渡す。データに対するポインターを必要とする C/C++ プログラムにデータを渡すのに推奨される方法。	アドレスのコピーに任意の変更を加えても <i>identifier</i> のアドレスには影響しませんが、アドレスのコピーを使用する <i>identifier</i> を変更すると <i>identifier</i> が変更されます。
CALL . . . RETURNING	関数戻り値を使用して C/C++ 関数を呼び出す。	

関連タスク

519 ページの『呼び出し側プログラムの中での引数の記述』

519 ページの『呼び出し先プログラムの中でのパラメーターの記述』

520 ページの『OMITTED 引数に関するテスト』

527 ページの『CALL . . . RETURNING の指定』
527 ページの『EXTERNAL 節によるデータの共用』
528 ページの『プログラム間でのファイルの共用 (外部ファイル)』
482 ページの『Java とのデータ共用』

関連参照

CALL ステートメント (COBOL for AIX 言語解説書)
USING 句 (COBOL for AIX 言語解説書)
INVOKE ステートメント (COBOL for AIX 言語解説書)

呼び出し側プログラムの中での引数の記述

呼び出し側プログラムでは、DATA DIVISION の他のデータ項目と同じ方法で、DATA DIVISION で引数を記述します。

引数のためのストレージは、最高位の最外部プログラムにおいてのみ割り振られます。例えば、プログラム A がプログラム B を呼び出し、プログラム B がプログラム C を呼び出すとします。データ項目はプログラム A で割り振られ、プログラム B と C の LINKAGE SECTION で記述され、そのデータの集合を 3 つのすべてのプログラムで使用できます。

ファイルのデータを参照する場合、データが参照される時には、そのファイルはオープンされていない必要があります。

引数を渡すためには、CALL ステートメントの USING 句をコーディングしてください。データ項目を BY VALUE で渡す場合、データ項目は基本項目でなければなりません。

関連タスク

520 ページの『LINKAGE SECTION のコーディング』
521 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

関連参照

USING 句 (COBOL for AIX 言語解説書)

呼び出し先プログラムの中でのパラメーターの記述

どんなデータが呼び出し側プログラムから渡されるのか知っている必要があります、さらに呼び出し側プログラムが直接的または間接的に呼び出すそれぞれのプログラムの LINKAGE SECTION でそれを記述する必要があります。

呼び出し側プログラムから渡されるデータを受け取るパラメーターを指定するために、USING 句を PROCEDURE DIVISION ヘッダーの後にコーディングしてください。

引数がサブプログラムに BY REFERENCE で渡される場合、メインプログラムで引き渡しが行われ定義されているもの以外のパラメーターおよびフィールドの間の関係をサブプログラムが指定しても無効です。サブプログラムでは、以下を行うことはできません。

- 対応する引数よりバイト総数が大きくなるようパラメーターを定義する。

- 呼び出し側プログラムから引数として渡されたテーブルの限度を超えるエレメントを参照するような添え字参照を使用する。
- 定義されたパラメーターの長さを超えるデータにアクセスする参照変更を使用する。
- 呼び出し側プログラムで定義された以外のデータ項目にアクセスするためにパラメーターのアドレスを操作する。

これらの規則のいずれかに違反していると、予期しない結果となります。

関連タスク

『LINKAGE SECTION のコーディング』

関連参照

USING 句 (*COBOL for AIX 言語解説書*)

OMITTED 引数に関するテスト

CALL ステートメント内の引数の代わりに OMITTED キーワードをコーディングして、1 つ以上の BY REFERENCE 引数が、呼び出し先プログラムに渡されないように指定することができます。

例えば、プログラム sub1 を呼び出すときに、2 番目の引数を省略するには、次のステートメントをコーディングします。

```
Call 'sub1' Using PARM1, OMITTED, PARM3
```

CALL ステートメントの USING 句の引数は、数および位置において呼び出し先プログラムのパラメーターと一致しなければなりません。

呼び出し先プログラムで、対応するパラメーターのアドレスを NULL と比較して、引数が OMITTED として渡されたかどうかをテストすることができます。以下に例を示します。

```
Program-ID. sub1.
...
Procedure Division Using RPARAM1, RPARAM2, RPARAM3.
  If Address Of RPARAM2 = Null Then
    Display 'No 2nd argument was passed this time'
  Else
    Perform Process-Param-2
  End-If
```

関連参照

CALL ステートメント (*COBOL for AIX 言語解説書*)

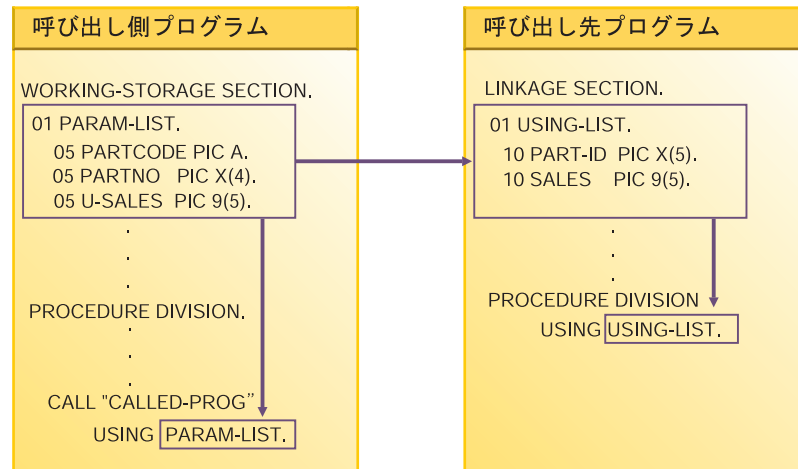
USING 句 (*COBOL for AIX 言語解説書*)

LINKAGE SECTION のコーディング

呼び出し側プログラムの引数と同じ数のデータ名を、呼び出し先プログラムの ID リストにコーディングしてください。位置で同期させます。なぜならコンパイラーは、呼び出し側プログラムの最初の引数を、呼び出し先プログラムの最初の identifier に渡し、以下同様に行うからです。

呼び出し先プログラムの identifier リストのデータ名の数、呼び出し側プログラムから渡される引数の数よりも大きいと、エラーになります。コンパイラーは引数とパラメーターの突き合わせを試行しません。

次の図は、あるプログラムから別のプログラムにデータ項目が渡される様子を示しています (暗黙的に BY REFERENCE)。



呼び出し側プログラムでは、パーツ (PARTCODE) とパーツ・ナンバー (PARTNO) のコードは別のデータ項目です。それに対して、呼び出し先プログラムでは、パーツのコードとパーツ・ナンバーのコードが 1 つのデータ項目 (PART-ID) に結合されています。呼び出し先プログラムでは、PART-ID への参照は、これらの項目に対する唯一有効な参照です。

引数を受け渡すための PROCEDURE DIVISION のコーディング

引数 BY VALUE を渡す場合には、サブプログラムの PROCEDURE DIVISION ヘッダーにある USING BY VALUE 節をコーディングします。引数を BY REFERENCE または BY CONTENT で渡す場合は、引数の受け渡し方法をヘッダーで指示する必要はありません。

```
PROCEDURE DIVISION USING BY VALUE. . .
```

```
PROCEDURE DIVISION USING. . .
PROCEDURE DIVISION USING BY REFERENCE. . .
```

上の最初のヘッダーは、データ項目は渡された BY VALUE を示します。2 番目または 3 番目のヘッダーは、項目が渡された BY REFERENCE または BY CONTENT を示します。

関連参照

手続き部ヘッダー (COBOL for AIX 言語解説書)

USING 句 (COBOL for AIX 言語解説書)

CALL ステートメント (COBOL for AIX 言語解説書)

受け渡されるデータのグループ化

プログラム間で渡す必要があるすべてのデータ項目をグループ化し、それらを 1 つのレベル 01 項目に入れることを考慮してください。そうすると、1 個のレベル 01 レコードを渡すことができます。

データ項目を BY VALUE で渡す場合、データ項目は基本項目でなければならないことに注意してください。

レコード突き合わせの間違いの可能性をより少なくするためには、レベル 01 レコードをコピー・ライブラリーの中に置き、両方のプログラムにそれをコピーするようにします。すなわち、呼び出し側プログラムの WORKING-STORAGE SECTION と呼び出し先プログラムの LINKAGE SECTION の中でコピーします。

関連タスク

520 ページの『LINKAGE SECTION のコーディング』

関連参照

CALL ステートメント (COBOL for AIX 言語解説書)

ヌル終了ストリングの処理

ヌル終了リテラルおよび 16 進リテラル X'00' と合わせてストリング処理動詞を使用する場合、COBOL はヌル終了ストリングをサポートします。

ヌル終了ストリング (例えば、C プログラムから渡された) は、次のコードのようなストリング処理メカニズムを使用して処理することができます。

```
01 L      pic X(20) value z'ab'.
01 M      pic X(20) value z'cd'.
01 N      pic X(20).
01 N-Length pic 99    value zero.
01 Y      pic X(13) value 'Hello, World!'.
```

ヌル終了ストリングの長さを決定して、そのストリングの値および長さを表示するには、次のようにコーディングします。

```
Inspect N tallying N-length for characters before initial X'00'
Display 'N: ' N(1:N-length) ' Length: ' N-length
```

ヌル終了ストリングを英数字ストリングに移動するが、ヌルを削除するには、次のようにコーディングします。

```
Unstring N delimited by X'00' into X
```

ヌル終了ストリングを作成するには、次のようにコーディングします。

```
String Y      delimited by size
              X'00' delimited by size
              into N.
```

2 つのヌル終了ストリングを連結するには、次のようにコーディングします。

```
String L      delimited by x'00'
              M      delimited by x'00'
              X'00' delimited by size
              into N.
```

関連タスク

105 ページの『ヌル終了ストリングの取り扱い』

関連参照

ヌル終了英数字リテラル (COBOL for AIX 言語解説書)

ポインターによるチェーン・リストの処理

レコード域のアドレスを渡したり受信する必要がある場合、ポインター・データ項目を使用できます。これは、USAGE IS POINTER 節を使用して定義されるデータ項目、または ADDRESS OF 特殊レジスターであるデータ項目のいずれかです。

ポインター・データ項目の代表的な適用は、チェーン・リスト (各レコードがそれぞれ次のレコードを指し示す一連のレコード) の処理です。

プログラム相互間のアドレスをチェーン・リストに入れて渡す場合は、NULL を使用して、以下の 2 つの方法のいずれかで、無効なアドレスの値 (非数値 0) をポインター項目に割り当てることができます。

- データ定義の中で VALUE IS NULL 節を使用する。
- SET ステートメントの中で送信フィールドとして NULL を使用する。

最後のレコードのポインター・データ項目がヌル値を含んでいるチェーン・リストの場合、このコードを使用して、リストの終わりを検査できます。

```
IF PTR-NEXT-REC = NULL  
  . . .  
  (logic for end of chain)
```

リストの終わりに達していない場合、プログラムはレコードを処理して次のレコードに移ることができます。

呼び出し側プログラムから渡されるデータには、無視したいヘッダー情報が含まれていることがあります。ポインター・データ項目は数値ではないので、それらに対して直接に算術演算を行うことはできません。しかし、ヘッダー情報をバイパスするために、SET ステートメントを使用して、渡されたアドレスを増分することができます。

『例: チェーン・リストを処理するためのポインターの使用』

関連タスク

520 ページの『LINKAGE SECTION のコーディング』

521 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

関連参照

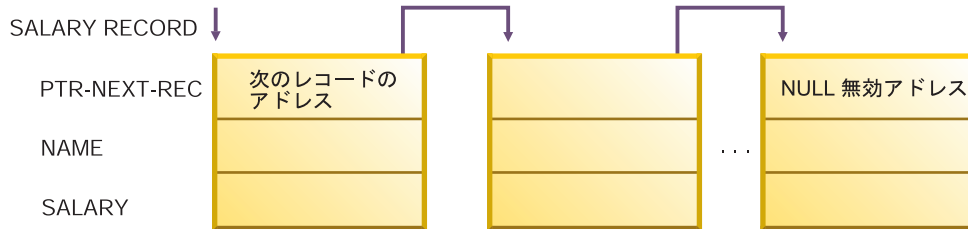
266 ページの『ADDR』

SET ステートメント (COBOL for AIX 言語解説書)

例: チェーン・リストを処理するためのポインターの使用

次の例は、リンク・リスト、つまりデータ項目のチェーン・リストを処理する方法を示しています。

この例では、個々の給与レコードで構成されるデータのチェーン・リストを示しています。次の図は、これらのレコードがストレージの中でどのようにリンクされているかを視覚化する 1 つの方法を示しています。最後のレコードを除いて、各レコードの最初の項目は次のレコードを指し示しています。最後のレコードの最初の項目は、それが最後のレコードであることを示すために、(有効なアドレスではなく)ヌル値を含んでいます。



これらのレコードを処理するアプリケーションの高水準擬似コードは、次のようになります。

```
Obtain address of first record in chained list from routine
Check for end of the list
Do until end of the list
  Process record
  Traverse to the next record
End
```

次のコードは、チェーン・リストを処理するこの例で使用される、呼び出し側プログラム LISTS の概要です。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LISTS.
ENVIRONMENT DIVISION.
DATA DIVISION.
*****
WORKING-STORAGE SECTION.
77 PTR-FIRST          POINTER VALUE IS NULL.          (1)
77 DEPT-TOTAL         PIC 9(4) VALUE IS 0.
*****
LINKAGE SECTION.
01 SALARY-REC.
   02 PTR-NEXT-REC    POINTER.                        (2)
   02 NAME            PIC X(20).
   02 DEPT            PIC 9(4).
   02 SALARY          PIC 9(6).
01 DEPT-X            PIC 9(4).
*****
PROCEDURE DIVISION USING DEPT-X.
*****
* FOR EVERYONE IN THE DEPARTMENT RECEIVED AS DEPT-X,
* GO THROUGH ALL THE RECORDS IN THE CHAINED LIST BASED ON THE
* ADDRESS OBTAINED FROM THE PROGRAM CHAIN-ANCH
* AND CUMULATE THE SALARIES.
* IN EACH RECORD, PTR-NEXT-REC IS A POINTER TO THE NEXT RECORD
* IN THE LIST; IN THE LAST RECORD, PTR-NEXT-REC IS NULL.
* DISPLAY THE TOTAL.
*****
CALL "CHAIN-ANCH" USING PTR-FIRST          (3)
SET ADDRESS OF SALARY-REC TO PTR-FIRST    (4)
*****
PERFORM WITH TEST BEFORE UNTIL ADDRESS OF SALARY-REC = NULL (5)

IF DEPT = DEPT-X
  THEN ADD SALARY TO DEPT-TOTAL
```

```

ELSE CONTINUE
END-IF
SET ADDRESS OF SALARY-REC TO PTR-NEXT-REC

```

(6)

```

END-PERFORM

```

```

*****

```

```

DISPLAY DEPT-TOTAL
GOBACK.

```

- (1) PTR-FIRST は、NULL の初期値を持つポインター・データ項目として定義されます。CHAIN-ANCH への呼び出しから正常に戻ると、PTR-FIRST には、チェーン・リスト内の最初のレコードのアドレスが入れられます。呼び出しで何か間違いが起こり、PTR-FIRST がチェーンの最初のレコードのアドレスの値を受け取っていないと、PTR-FIRST はヌル値のままであり、プログラムのロジックに従って、レコードは処理されません。
- (2) 呼び出し側プログラムの LINKAGE SECTION には、チェーン・リストのレコードの記述が入っています。さらに、CALL ステートメントの USING 節を使用して渡される部門コードの記述も含まれています。
- (3) 最初の SALARY-REC レコード域のアドレスを取得するために、LISTS プログラムはプログラム CHAIN-ANCH を呼び出します。
- (4) SET ステートメントのレコード記述 SALARY-REC の基礎となっているのは、PTR-FIRST に含まれているアドレスです。
- (5) この例のチェーン・リストは、最後のレコードに無効アドレスが含まれるようにセットアップされます。チェーン・リスト終了に対するこの検査は、do-while 構造を使用して行われます (最後のレコードのポインター・データ項目には値 NULL が割り当てられる構造になっています)。
- (6) LINKAGE-SECTION 内のレコードのアドレスは、SALARY-REC の最初のフィールドとして送信されるポインター・データ項目によって、次のレコードのアドレスに等しく設定されます。レコード処理ルーチンが繰り返され、チェーン・リスト内の次のレコードが処理されます。

別のプログラムから受け取ったアドレスを増分するには、LINKAGE SECTION および PROCEDURE DIVISION を次のようにセットアップすることができます。

```

LINKAGE SECTION.
01 RECORD-A.
   02 HEADER          PIC X(12).
   02 REAL-SALARY-REC PIC X(30).
. . .
01 SALARY-REC.
   02 PTR-NEXT-REC    POINTER.
   02 NAME            PIC X(20).
   02 DEPT            PIC 9(4).
   02 SALARY          PIC 9(6).
. . .
PROCEDURE DIVISION USING DEPT-X.
. . .
   SET ADDRESS OF SALARY-REC TO ADDRESS OF REAL-SALARY-REC

```

この時点では、SALARY-REC のアドレスは、REAL-SALARY-REC、または RECORD-A + 12 のアドレスを基底にしています。

関連タスク

523 ページの『ポインターによるチェーン・リストの処理』

プロシージャ・ポインターと関数ポインターの使用

プロシージャ・ポインターとは、USAGE IS PROCEDURE-POINTER 節によって定義されるデータ項目です。関数ポインターとは、USAGE IS FUNCTION-POINTER 節によって定義されるデータ項目です。

ここでは、「ポインター」は、プロシージャ・ポインター・データ項目または関数ポインター・データ項目のどちらかを指します。プロシージャ・ポインター・データ項目または関数ポインター・データ項目は、以下の入り口点の入り口アドレス (ポインター) が入るように設定することができます。

- ネストされていない別の COBOL プログラム。
- 別の言語で作成されているプログラム。例えば、C 関数の入り口アドレスを受け取るためには、CALL ステートメントの CALL RETURNING 形式を使用して関数を呼び出してください。この結果、SET ステートメントの形式を使用してプロシージャ・ポインターに変換できるポインターが戻されます。
- 別の COBOL プログラムの代替入り口点 (ENTRY ステートメントで定義されたもの)。

ポインター・データ項目を設定するには、SET ステートメントを使用する必要があります。以下に例を示します。

```
CALL 'MyCFunc' RETURNING ptr.  
SET proc-ptr TO ptr.  
CALL proc-ptr USING dataname.
```

ポインター項目を、CALL *identifier* ステートメントで呼び出されるロード・モジュールの入り口アドレスに設定し、後でプログラムが呼び出されたモジュールをキャンセルしたとします。この場合、ポインター項目は未定義の状態になり、後でその項目を参照しても、結果は信頼できないものになります。

関連参照

266 ページの『ADDR』

617 ページの『32 ビット・アプリケーションの 64 ビット・モードへの移行』

PROCEDURE-POINTER 句 (COBOL for AIX 言語解説書)

SET ステートメント (COBOL for AIX 言語解説書)

戻りコード情報の引き渡し

プログラム間で戻りコードを渡すには、RETURN-CODE 特殊レジスターを使用します。(メソッドは RETURN-CODE 特殊レジスターに情報を戻しません、プログラムへの呼び出しの後でこのレジスターを検査できます。)

また、メソッドの PROCEDURE DIVISION ヘッダーで RETURNING 句を使用して、起動プログラムまたはメソッドに情報を戻すこともできます。PROCEDURE DIVISION . . . RETURNING で CALL . . . RETURNING を指定しても、RETURN-CODE レジスターは設定されません。

RETURN-CODE 特殊レジスタの理解

COBOL プログラムがその呼び出し側に戻ると、RETURN-CODE 特殊レジスタの内容が呼び出されたプログラムの RETURN-CODE 特殊レジスタの値によって設定されます。

呼び出し先プログラムによる RETURN-CODE の設定は、COBOL プログラム間の呼び出しに制限されています。そのため、COBOL プログラムが C プログラムを呼び出しても、COBOL プログラムの RETURN-CODE 特殊レジスタが設定されることは期待できません。

COBOL プログラムと C プログラムで同じように機能させるためには、COBOL プログラムが RETURNING 句を使用して C プログラムを呼び出すようにしなければなりません。C プログラム (関数) が関数値を正しく宣言していれば、呼び出し COBOL プログラムの RETURNING 値が設定されます。

INVOKE ステートメントを使用して RETURN-CODE 特殊レジスタを設定することはできません。

PROCEDURE DIVISION RETURNING . . . の使用

呼び出し側プログラムに情報を戻すには、プログラムの PROCEDURE DIVISION ヘッダーで RETURNING 句を使用してください。

```
PROCEDURE DIVISION RETURNING dataname2
```

例の呼び出し先プログラムが正常にその呼び出し側に戻ると、*dataname2* の値が、CALL ステートメントの RETURNING 句で指定された ID に保管されます。

```
CALL . . . RETURNING dataname2
```

CALL . . . RETURNING の指定

C/C++ 関数または COBOL サブルーチンへの呼び出しでは、CALL ステートメントの RETURNING 句を指定することができます。

RETURNING 句のフォーマットは次のとおりです。

```
CALL . . . RETURNING dataname2
```

呼び出し先プログラムの戻り値は *dataname2* に保管されます。*dataname2* は、呼び出し側プログラムの DATA DIVISION で定義しなければなりません。ターゲット関数で宣言される戻り値のデータ型は、*dataname2* のデータ型と同じでなければなりません。

EXTERNAL 節によるデータの共用

EXTERNAL 節を使用すると、別々にコンパイルされたプログラムやメソッド (バッチ・シーケンスのプログラムを含む) がデータ項目を共用できるようになります。WORKING-STORAGE SECTION のレベル 01 データ記述に EXTERNAL をコーディングします。

次の規則が適用されます。

- EXTERNAL グループ項目に従属する項目はそれ自身が EXTERNAL です。
- EXTERNAL データ項目の名前を、同じプログラムの中で別の EXTERNAL 項目の名前として使用することはできません。
- VALUE 節は、グループ項目または EXTERNAL である従属項目には指定できません。

実行単位内で、ある COBOL プログラムまたはメソッドの項目のデータ記述が、その項目を含むプログラムのデータ記述と同じ場合は、そのプログラムまたはメソッドはその項目にアクセスして処理できます。例えば、プログラム A に次のデータ記述があるとします。

```
01 EXT-ITEM1    EXTERNAL    PIC 99.
```

プログラム B は、WORKING-STORAGE SECTION に同じデータ記述が存在する場合、そのデータ項目にアクセスすることができます。

EXTERNAL データ項目にアクセス権を持っているプログラムはすべて、その項目の値を変更できます。そのため、保護しなければならないデータ項目には、この節を使用しないでください。

プログラム間でのファイルの共用 (外部ファイル)

実行単位の別々にコンパイルされたプログラムまたはメソッドが共用ファイルとしてファイルにアクセスできるようにするには、そのファイルに EXTERNAL 節を使用します。

以下の指針に従うことをお勧めします。

- ファイル状況コードを検査するすべてのプログラムの FILE STATUS 節で、同じデータ名を使用する。
- 同じファイル状況フィールドを検査するすべてのプログラムについて、ファイル状況フィールドのレベル 01 データ定義で EXTERNAL 節をコーディングする。

外部ファイルを使用すると、次のような利点があります。

- メインプログラムに入出力ステートメントが含まれていなくても、ファイルのレコード域を参照することができます。
- それぞれのサブプログラムが、OPEN や READ のような単一の入出力機能を制御することができます。
- それぞれのプログラムがファイルにアクセスすることができます。

529 ページの『例: 外部ファイルの使用』

関連タスク

12 ページの『入出力操作でのデータの使用』

関連参照

EXTERNAL 節 (COBOL for AIX 言語解説書)

例: 外部ファイルの使用

次の例は、いくつかのプログラムでの外部ファイルの使用を示しています。それぞれのサブプログラムにファイルの同じ記述が含まれていることを保証するために、COPY ステートメントを使用します。

次の表で、メインプログラムとサブプログラムについて説明します。

名前	機能
ef1	メインプログラム。すべてのサブプログラムを呼び出し、レコード域の内容を検査する。
eflopeno	出力用に外部ファイルをオープンし、ファイル状況コードを検査する。
eflwrite	外部ファイルにレコードを書き込み、ファイル状況コードを検査する。
eflopeni	入力用に外部ファイルをオープンし、ファイル状況コードを検査する。
eflread	外部ファイルからレコードを読み取り、ファイル状況コードを検査する。
eflclose	外部ファイルをクローズし、ファイル状況コードを検査する。

各プログラムは、次の 3 つのコピーブックを使用します。

- efselect は FILE-CONTROL 段落に入れます。

```
Select ef1
Assign To ef1
File Status Is efs1
Organization Is Sequential.
```

- effile は FILE SECTION に入れます。

```
Fd ef1 Is External
      Record Contains 80 Characters
      Recording Mode F.
01 ef-record-1.
   02 ef-item-1 Pic X(80).
```

- efwrkstg は WORKING-STORAGE SECTION に入れます。

```
01 efs1          Pic 99 External.
```

外部ファイルを使用する入出力

```
Identification Division.
Program-Id.
    ef1.
*
* This main program controls external file processing.
*
Environment Division.
Input-Output Section.
File-Control.
    Copy efselect.
Data Division.
File Section.
    Copy effile.
Working-Storage Section.
    Copy efwrkstg.
Procedure Division.
    Call "eflopeno"
    Call "eflwrite"
    Call "eflclose"
    Call "eflopeni"
    Call "eflread"
```

```

    If ef-record-1 = "First record" Then
      Display "First record correct"
    Else
      Display "First record incorrect"
      Display "Expected: " "First record"
      Display "Found   : " ef-record-1
    End-If
    Call "eflclose"
    Goback.
  End Program efl.
  Identification Division.
  Program-Id.
  eflopeno.
*
* This program opens the external file for output.
*
  Environment Division.
  Input-Output Section.
  File-Control.
  Copy efselect.
  Data Division.
  File Section.
  Copy effile.
  Working-Storage Section.
  Copy efwrkstg.
  Procedure Division.
  Open Output efl
  If efs1 Not = 0
    Display "file status " efs1 " on open output"
    Stop Run
  End-If
  Goback.
  End Program eflopeno.
  Identification Division.
  Program-Id.
  eflwrite.
*
* This program writes a record to the external file.
*
  Environment Division.
  Input-Output Section.
  File-Control.
  Copy efselect.
  Data Division.
  File Section.
  Copy effile.
  Working-Storage Section.
  Copy efwrkstg.
  Procedure Division.
  Move "First record" to ef-record-1
  Write ef-record-1
  If efs1 Not = 0
    Display "file status " efs1 " on write"
    Stop Run
  End-If
  Goback.
  End Program eflwrite.
  Identification Division.
  Program-Id.
  eflopeni.
*
* This program opens the external file for input.
*
  Environment Division.
  Input-Output Section.
  File-Control.
  Copy efselect.

```

```

Data Division.
File Section.
  Copy effile.
Working-Storage Section.
  Copy efwrkstg.
Procedure Division.
  Open Input ef1
  If efs1 Not = 0
    Display "file status " efs1 " on open input"
  Stop Run
  End-If
  Goback.
End Program eflopeni.
Identification Division.
Program-Id.
  eflread.
*
* This program reads a record from the external file.
*
Environment Division.
Input-Output Section.
File-Control.
  Copy efselect.
Data Division.
File Section.
  Copy effile.
Working-Storage Section.
  Copy efwrkstg.
Procedure Division.
  Read ef1
  If efs1 Not = 0
    Display "file status " efs1 " on read"
  Stop Run
  End-If
  Goback.
End Program eflread.
Identification Division.
Program-Id.
  eflclose.
*
* This program closes the external file.
*
Environment Division.
Input-Output Section.
File-Control.
  Copy efselect.
Data Division.
File Section.
  Copy effile.
Working-Storage Section.
  Copy efwrkstg.
Procedure Division.
  Close ef1
  If efs1 Not = 0
    Display "file status " efs1 " on close"
  Stop Run
  End-If
  Goback.
End Program eflclose.

```

コマンド行引数の使用

コマンド行でメインプログラムに引き数を渡すことができます。オペレーティング・システムはメインプログラムを呼び出す際に、引数を含むヌル終了ストリングを使用します。

入力された引数が、その引数を受け取る COBOL データ名より短い場合は、ヌル終了ストリングの操作に関して下記の関連タスクで示されている手法で、その引数を分離します。

引数の処理方法は、cob2 コマンド (またはそのバリエーションのいずれか) の `-host` オプションを使用しているかどうかによって異なります。

`-host` オプションを指定しない場合、コマンド行引数がネイティブのデータ形式で受け渡され、AIX が次の引数を含むすべてのメインプログラムを呼び出します。

- コマンド行引数の数に 1 を加えた数
- プログラムの名前へのポインター
- 最初の引数へのポインター
- 2 番目の引数へのポインター
- . . .
- n 番目の引数へのポインター

`-host` オプションを指定すると、AIX では、ストリングの長さを含むハーフワード接頭部がある EBCDIC 文字ストリングを使用して、すべてのメインプログラムを呼び出します。コマンド行引数は、引用符 (") で囲んだ単一ストリングとして入力する必要があります。ストリング内の引用符文字を渡すには、引用符の前に円記号 (¥) のエスケープ文字を付けます。

『例: `-host` オプションを使用しないコマンド行引数』

533 ページの『例: `-host` オプションを使用したコマンド行引数』

関連タスク

105 ページの『ヌル終了ストリングの取り扱い』

522 ページの『ヌル終了ストリングの処理』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』

例: `-host` オプションを使用しないコマンド行引数

この例は、コマンド行引数の読み取り方法を示しています。

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. "targaix".  
*  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
*  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
*  
LINKAGE SECTION.
```

```

01 PARM-LEN PIC S9(9) COMP.
01 OS-PARM.
   02 PARMPTR-TABLE OCCURS 1 TO 100 TIMES DEPENDING ON PARM-LEN.
   03 PARMPTR POINTER.
01 PARM-STRING PIC XX.
*
PROCEDURE DIVISION USING BY VALUE PARM-LEN BY REFERENCE OS-PARM.
  display "parm-len=" parm-len
  SET ADDRESS OF PARM-STRING TO PARMPTR(2).
  display "parm-string= '" PARM-STRING "'";
  EVALUATE PARM-STRING
    when "01" display "case one"
    when "02" display "case two"
    when "95" display "case ninety-five"
    when other display "case unknown"
  END-EVALUATE      GOBACK.

```

次のプログラムをコンパイルし、実行するとします。

```

cob2 targaix.cbl
a.out 95

```

結果は次のとおりです。

```

parm-len=000000002
parm-string= '95'
case ninety-five

```

例: -host オプションを使用したコマンド行引数

この例は、コマンド行引数の読み取り方法を示しています。

```

IDENTIFICATION DIVISION.
PROGRAM-ID. "testarg".
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
*
linkage section.
01 os-parm.
   05 parm-len          pic s999 comp.
   05 parm-string.
     10 parm-char      pic x occurs 0 to 100 times
                       depending on parm-len.
*
PROCEDURE DIVISION using os-parm.
  display "parm-len=" parm-len
  display "parm-string=" parm-string ""
  evaluate parm-string
    when "01" display "case one"
    when "02" display "case two"
    when "95" display "case ninety-five"
    when other display "case unknown"
  end-evaluate      GOBACK.

```

以下のように、このプログラムをコンパイルして実行するとします。

```

cob2 -host testarg.cbl
a.out "95"

```

結果の出力は、次のとおりです。

```
parm-len=002  
parm-string='95'  
case ninety-five
```

第 28 章 共用ライブラリーの構築

リンクを使用すると、呼び出し側のソース・コードには含まれていない別のプログラムを呼び出すことができます。呼び出し側プログラムのオブジェクト・モジュールは、実行前または実行中に、呼び出し先プログラムのオブジェクト・モジュールとリンクされます。

共用ライブラリーは、頻繁に使用する関数のライブラリーとして使用することができます。共用ライブラリーに格納されている関数が 1 つしかない場合でも、共用ライブラリーは、必要に応じて呼び出し側プログラムに関数を提供することができます。

COBOL 用語における共用ライブラリーとは、最外部プログラムの集合を意味します。これらの最外部プログラムにはネストされたプログラムが含まれている可能性があります。共用ライブラリーの外部にあるプログラムは、最外部プログラム(入り口点とも呼ばれる)しか呼び出すことができません。複数の COBOL プログラムを 1 つの実行可能ファイルとしてコンパイルおよびリンクできるのと同様に、1 つ以上のコンパイル済み最外部 COBOL プログラムをリンクし、共用ライブラリーを作成することができます。

共用ライブラリー内の最外部プログラムはプログラム・ライブラリーの一部なので、各プログラムは共用ライブラリー内ではサブプログラムと呼ばれます。共用ライブラリーにプログラムが 1 つしか入っていない場合でも、そのプログラムは共用ライブラリーのサブプログラムと見なされます。

共用 COBOL ライブラリーを呼び出したり、COBOL および C の共用ライブラリーを呼び出したりできます。

537 ページの『例: サンプルの共用ライブラリーの作成』

540 ページの『例: CALL identifier を使用した共用ライブラリーの呼び出し』

542 ページの『例: COBOL および C の共用ライブラリーを呼び出す COBOL プログラム』

関連概念

『スタティック・リンクおよび共用ライブラリー』

536 ページの『共用ライブラリーへの参照をリンカーが解決する方法』

スタティック・リンクおよび共用ライブラリー

スタティック・リンクは、1 つの実行可能モジュール内で、呼び出し側プログラムが呼び出し先プログラムにリンクされる場合に行われます。プログラムのロード時には、オペレーティング・システムによって、実行可能コードおよびデータが含まれる 1 つのファイルがメモリーに格納されます。

プログラムを静的にリンクすると、共用ライブラリー内の実行可能ファイルまたはサブプログラムに、複数のプログラムに対する実行可能コードが含まれます。

スタティック・リンクの主な利点は、必要なものを完備した独立プログラムを作成できることです。言い換えれば、実行可能プログラムを構成する一部分（実行可能ファイル）は、追跡を行う必要があります。スタティック・リンクには、次のような欠点があります。

- リンクされる外部プログラムは実行可能ファイルに組み込まれるため、ファイルのサイズが大きくなります。
- 実行可能ファイルの動作を変更するには、これらのファイルをリンクし直す必要があります。
- 外部呼び出し先プログラムを共用することはできません。このため、複数の呼び出し側プログラムが呼び出し先プログラムにアクセスする必要がある場合は、メモリー内に呼び出し先プログラムの重複コピーをロードする必要があります。

これらの欠点に対処するには、共用ライブラリーを使用します。

共用ライブラリーを使用すると、複数のプログラムが実行可能モジュールの単一コピーを共用することができます。実行可能モジュールは、使用側のプログラムから完全に独立しています。複数のサブプログラムを共用ライブラリーに組み込むと、呼び出し側プログラムはこれらのサブプログラムを、呼び出し側プログラムの実行可能コードの一部であるかのように使用することができます。共用ライブラリー・サブプログラムを変更する際には、呼び出し側プログラムの再コンパイルや再リンクを行う必要がありません。

通常、共用ライブラリーは、多数のプログラムで使用できる共通関数を提供します。例えば、共用ライブラリーを使用して、サブプログラム・パッケージ、サブシステム、および他のプログラムのインターフェースをインプリメントすることや、オブジェクト指向のクラス・ライブラリーを作成することが可能です。

提供されたランタイム共用ライブラリーや、ユーザー独自の COBOL 共用ライブラリーと動的にリンクすることができます。

537 ページの『例: サンプルの共用ライブラリーの作成』

関連概念

『共用ライブラリーへの参照をリンカーが解決する方法』

共用ライブラリーへの参照をリンカーが解決する方法

プログラムをコンパイルするときには、コンパイラーによって、プログラム内のコードに対応するオブジェクト・モジュールが生成されます。外部オブジェクト・モジュールに含まれる任意のサブプログラム（C では関数、他の言語ではサブルーチン）を使用すると、コンパイラーは外部プログラム参照をプログラムのオブジェクト・モジュールに追加します。

リンカーはこのような外部参照を解決します。インポート・ライブラリー内または共用ライブラリーのモジュール定義ファイル内で外部サブプログラムへの参照が検出された場合、その外部サブプログラムのコードは共用ライブラリー内にあります。リンカーは、共用ライブラリーへの外部参照を解決するために、実行可能ファイルのロード時に共用ライブラリー・コードの場所をローダーに通知する情報を実行可能ファイルに追加します。

COBOL CALL ステートメントによる共用ライブラリーへの参照は、リンカーでは解決されません。DYNAM コンパイラー・オプションが有効な場合は、CALL *identifier* および CALL *literal* ステートメントが実行されるときに、COBOL がこれらの呼び出しを解決します。

『例: サンプルの共用ライブラリーの作成』

関連概念

535 ページの『スタティック・リンクおよび共用ライブラリー』

関連参照

251 ページの『リンカーの入出力ファイル』

例: サンプルの共用ライブラリーの作成

次の 3 つの例のファイル (alpha.cbl、beta.cbl、gamma.cbl) のそれぞれに、COBOL プログラムが含まれています。サンプルの共用ライブラリーは、例のプログラムとその後で説明する手順によって作成することができます。

例 1: alpha.cbl

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. alpha.  
*  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
*  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 hello1  pic x(30)  value is "message from alpha to beta".  
01 hello2  pic x(30)  value is "message from alpha to gamma".  
*  
PROCEDURE DIVISION.  
    display "alpha begins"  
    call "beta" using hello1  
    display "alpha after beta"  
    call "gamma" using hello2  
    display "alpha after gamma"  
    goback.
```

例 2: beta.cbl

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. beta.  
*  
ENVIRONMENT DIVISION.  
*  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
*  
Linkage section.  
01 msg          pic x(30).  
*  
PROCEDURE DIVISION using msg.  
    DISPLAY "beta gets msg=" msg.  
    goback.
```

例 3: gamma.cbl

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. gamma.  
*  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
*  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
*  
Linkage section.  
01 msg          pic x(30).  
*  
PROCEDURE DIVISION using msg.  
    DISPLAY "gamma gets msg=" msg.  
    goback.
```

プロシージャ

これらのプログラムを実行する最も単純な方法は、それらのプログラムを次のコマンドを使用して単一モジュール内にコンパイルすることです。

```
cob2 -o m_abg alpha.cbl beta.cbl gamma.cbl
```

その後、コマンド `m_abg` を発行してプログラムを実行できます。出力結果は次のようになります。

```
alpha begins  
beta gets msg=message from alpha to beta  
alpha after beta  
gamma gets msg=message from alpha to gamma  
alpha after gamma
```

プログラムを単一モジュール内にコンパイルする代わりに、共用ライブラリー (`sh_bg` など) にベータとガンマを入れ、そのライブラリーを使用するモジュールに `alpha.cbl` を入れることができます。このケースでは、次のステップを実行します。

1. (オプション) エクスポート・ファイル (`bg.exp`) を作成し、そのファイルに共用ライブラリーがエクスポートするシンボルを定義します。`bg.exp` の内容は次のようになります。

```
#!  
* symbols exported by shared library sh_bg (or archive libbg.a)  
BETA  
GAMMA
```

COBOL はデフォルトでは外部シンボルに大文字の名前を使用するため、ファイル内のシンボル名は大文字です。大/小文字混合の名前が必要な場合は、`PGMNAME(MIXED)` コンパイラー・オプションを使用します。

`bg.exp` というエクスポート・ファイルを使用する場合は、共用ライブラリーを作成するときに `-bE:bg.exp` オプションを使用する必要があります。手順を次に示します。

2. 次のコマンドを使用して、`sh_bg` という共用ライブラリー・オブジェクト内にベータとガンマを結合します。

```
cob2 -o sh_bg beta.cbl gamma.cbl {-bE:bg.exp | -bexpall} -bM:SRE -bnoentry
```

このコマンドは、コンパイラーおよびリンカーに以下の情報を提供します。

- `-o sh_bg beta.cbl gamma.cbl` は、出力モジュールに `sh_bg` という名前を付け、`beta.cbl` と `gamma.cbl` をコンパイルおよびリンクします。
 - `-bE:bg.exp` は、エクスポート・ファイル `bg.exp` に指定されたシンボルをエクスポートするよう、リンカーに指示します。このオプションを使用するのは、エクスポート・ファイルを前述のステップ 1 で示されているように作成した場合のみです。
 - `-bexpall` は、最もグローバルなシンボルをエクスポートするよう、リンカーに指示します。このオプションを使用するのは、エクスポート・ファイルを使用していない場合のみです。
 - `-bM:SRE` は、出力モジュールを共有の再入可能オブジェクトにするよう、リンカーに指示します。
 - `-bnoentry` は、このオブジェクトがメイン入り口点を持たないよう、リンカーに指示します (これは `BETA` と `GAMMA` で実行されます)。
3. `sh_bg` は即時に使用できます。複数の共有オブジェクトがある場合は、便宜上それらをアーカイブ・ライブラリーに入れることもできます。アーカイブ・ライブラリーを作成するには、次のコマンドを使用します。

```
ar qv libbg.a sh_bg
```

4. 次のコマンドのいずれかを発行して、ここではアーカイブ・ライブラリーを使用して、`alpha.cbl` を再コンパイルします。

```
cob2 -o m_alpha alpha.cbl -L. libbg.a
cob2 -o m_alpha alpha.cbl -L. -lbg
```

`-lxx` は、`libxx.a` に対する標準 AIX 規則です。`-L.` オプションは、`libbg.a` が現行ディレクトリー (.) であることをリンカーに伝えます。次のように完全なディレクトリーを指定すると無難です。

```
cob2 -o m_alpha alpha.cbl -L/usr/jane/libraries -lbg
```

`-L.` 形式は、`m_alpha` および `libbg.a` が現行ディレクトリー内にある場合のみ機能します。

最後の 3 つのステップに代わるものとして、コマンドを結合する `Make` ファイルを作成することができます。

540 ページの『例: サンプルの共用ライブラリー用の `Make` ファイルの作成』

関連タスク

248 ページの『リンカーへのオプションの引き渡し』

関連参照

245 ページの『`cob2`、`cob2_j`、および `cob2_r` オプション』

251 ページの『リンカーの入出力ファイル』

AIX プログラミングの一般概念: プログラムの作成およびデバッグ

AIX Commands Reference, Volume 1 (ar command)

例: サンプルの共用ライブラリー用の Make ファイルの作成

次の例は、サンプル共用ライブラリー用の Makefile の作成方法を示しています。(この Make ファイルは、エクスポート・ファイル bg.exp により共用ライブラリーがエクスポートするシンボルが定義されると仮定しています。)

```
#
#
all:    m_abg libbg.a m_alpha

# Create m_abg containing alpha, beta, and gamma

m_abg:  alpha.cbl beta.cbl gamma.cbl
        cob2 -o m_abg alpha.cbl beta.cbl gamma.cbl

# Create libbg.a containing beta and gamma
# sh_bg.o is a shared object that exports the symbols defined in bg.exp
# libbg.a is a shared library that contains one shared object
# More objects can be added to the library

libbg.a: beta.cbl gamma.cbl bg.exp
        rm -rf libbg.a
        cob2 -o sh_bg beta.cbl gamma.cbl -bE:bg.exp -bM:SRE -bnoentry
        ar qv libbg.a sh_bg
        rm -f sh_bg

# Create m_alpha containing alpha and using library libbg.a
m_alpha: alpha.cbl
        cob2 -o m_alpha alpha.cbl -L. libbg.a
clean:
        rm -f m_abg m_alpha sh_bg libbg.a *.lst
```

このプログラムの終わりでは、rm コマンドにより、sh_bg と、libbg.a の古いコピーが除去されています。

コマンド m_abg またはコマンド m_alpha を実行すると、同じ出力が得られます。

共用ライブラリーは、AIX では広く使用されます。COBOL ランタイムはいくつかの共用ライブラリー内にあり、そのまま C ランタイムです。共用ライブラリーは、アプリケーションをパッケージ化するための便利で効率的な方法を提供します。

例: CALL identifier を使用した共用ライブラリーの呼び出し

次の例は、CALL *identifier* を使用して共用ライブラリーを呼び出すモジュールを、作成および実行することができることを示しています。

最初のプログラム dl1.cbl は、CALL *identifier* を使用して 2 番目のプログラム dl1a.cbl を呼び出します。

dl1.cbl

* Simple dynamic call to dl1a

Identification Division.
Program-id. dl1.

*
Environment Division.
Configuration Section.
Input-Output Section.

```

File-control. *
Data Division.
File Section.
Working-storage Section.
01 var pic x(10).
Linkage Section.
*
Procedure Division.
    move "DL1A" to var.
    display "Calling " var.
    call var.
    move "dl1a      " to var.
    display "Calling " var.
    call var.
    stop run.
End program dl1.

```

dl1a.cbl

* Called by dl1.cbl by call identifier.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. dl1a.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
OBJECT-COMPUTER.
ANY-THING.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
77 num pic 9(4) binary value is zero.
*
PROCEDURE DIVISION.
LA-START.
    display "COBOL DL1A function." upon console.
    add 11 to num.
    display "num = " num
    goback.

```

プロシージャ

この例を作成して実行するには、次の手順を実行します。

1. `cob2 dl1.cbl -o dl1` と入力して、実行可能モジュール `dl1` を生成します。
2. `cob2 dl1a.cbl -o DL1A` と入力して、実行可能モジュール `DL1A` を生成します。
3. コマンド `export COBPATH=.` と入力して、現行ディレクトリーが動的呼び出しの検索場所となるようにします。
4. `dl1` と入力してプログラムを実行します。

`DL1A` は大文字です。 `PGMNAME(MIXED)` コンパイラー・オプションを使用しない場合、プログラム名は大文字に変更されます。

ランタイム・ロードの制約事項: `CALL identifier` または `CALL literal` を `DYNAM` オプションとともに使用して、共用ライブラリーに対して呼び出しを行う場合:

- エクスポートできる名前は 1 つだけです。
- エクスポートされる名前は、共用ライブラリーと同じ名前であればなりません。

- 共用ライブラリーが COBOL 共用ライブラリーの場合は、共用ライブラリーと同じ COBOL ライブラリーとリンクする必要があります: `libcob2s.a` または `libcob2_r.a`。

デフォルトのライブラリーは、`-lcob2s` および `-lcob2_r` コマンド・フラグによって判別されます。フラグ `-lcob2s` は `libcob2s.a` を使用するようリンカーに指示し、`-lcob2_r` は `libcob2_r.a` を使用するようリンカーに指示します。

CALL *identifier* のターゲット・ファイル名はプログラムの名前と一致する必要があるため、上の例の実行可能モジュールは、`d11a` としてではなく、`DL1A` として生成する必要があります。

例: COBOL および C の共用ライブラリーを呼び出す COBOL プログラム

この例は、メイン COBOL プログラムが COBOL プログラムと C 関数の両方を動的に呼び出すことができることを示しています。

この例は、次の 3 つのプログラムで構成されます。

- `d16.cbl`: CALL *identifier* を使用して `d16a` および `d16b` を呼び出す COBOL メインプログラム
- `d16a.c`: `d16` が呼び出す C 関数
- `d16b.cbl`: `d16` が呼び出す COBOL プログラム

例 1: `d16.cbl`

```

cbl pgmname(longmixed)
* d16:
* - call C function - d16a executable
* - call "d16b" with pgmname(mixed)
  Identification Division.
  Program-id. d16.
*
  Environment Division.
  Configuration Section.
  Input-Output Section.
  File-control. *
  Data Division.
  File Section.
  Working-storage Section.
  01 var pic x(10).
  Linkage Section.
*
  Procedure Division.
    move "d16a" to var.
    display "Calling " var.
    call var.
    move "d16a" to var.
    display "Calling " var.
    call var.
    move "d16b" to var.
    display "Calling " var.
    call var.
    move "d16b" to var.
    display "Calling " var.
    call var.
    stop run.
  End program d16.

```

例 2: d16a.c

```
void d16a() {
    static int n = 123;
    printf("C d16a function.¥n");
    n = n * 10;
    printf("n = %d¥n", n);
}
```

例 3: d16b.cbl

```
cb1 pgmname(longmixed)
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "d16b".
*
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  OBJECT-COMPUTER.
  ANY-THING.
*
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  77 num pic 9(4) binary value is zero.
*
  PROCEDURE DIVISION.
  LA-START.
    display "COBOL d16b function." upon console.
    add 111 to num.
    display "num = " num
    goback.
```

プロシージャ

この例を作成して実行するには、次の手順を実行します。

1. `cob2 d16.cbl -o d16` と入力して、実行可能モジュール `d16` を生成します。
2. `cob2 d16b.cbl -o d16b` と入力して、実行可能モジュール `d16b` を生成します。
3. `x1c -o d16a -ed16a d16a.c` と入力して、入り口点 `d16a` を含む実行可能モジュール `d16a` を生成します。
4. コマンド `export COBPATH=` を入力して、動的呼び出しを検索するディレクトリを設定します。このケースでは、現行ディレクトリが検索されます。
5. `d16` と入力してプログラムを実行します。

第 29 章 マルチスレッド化のための COBOL プログラムの準備

COBOL プログラムではプログラム・スレッドを開始したり管理したりはできませんが、COBOL プログラムを準備して、マルチスレッド化環境で実行させることができます。、プロセス内の複数のスレッドで COBOL プログラムを実行することができます。

マルチスレッド化による実行のための明示的な COBOL 言語は存在しません。正しくは、THREAD コンパイラー・オプションを使用してコンパイルを行います。

THREAD コンパイラー・オプションを使用して COBOL プログラムをコンパイルすると、他のアプリケーションがこれらの COBOL プログラムを呼び出して、これらのプログラムを 1 つのプロセス内の複数のスレッドで実行するか、あるいは 1 つのスレッド内の複数のプログラム呼び出しインスタンスとして実行することができます。

551 ページの『例: マルチスレッド環境での COBOL の使用』

関連概念

『マルチスレッド化』

関連タスク

547 ページの『マルチスレッド化による言語エレメントの処理』

549 ページの『マルチスレッド化サポートのための THREAD の選択』

549 ページの『マルチスレッド化されたプログラムへの制御権移動』

550 ページの『マルチスレッド化されたプログラムの終了』

550 ページの『マルチスレッド化による COBOL 制限の処理』

関連参照

311 ページの『THREAD』

PROGRAM-ID 段落 (COBOL for AIX 言語解説書)

マルチスレッド化

マルチスレッド化の COBOL サポートを使用するには、プロセス、スレッド、実行単位、プログラム起動インスタンスの相互関係を理解する必要があります。

オペレーティング・システムおよびマルチスレッド化アプリケーションは、プロセス内の実行フローを処理することができます。実行フローとは、プログラムのすべて、または一部が実行時に発生する一連のイベントです。1 つのプロセス内で実行されるプログラムはリソースを共用することができます。プロセスは操作することができます。例えば、システムがプロセスの実行に使用する時間において、プロセスに高い、または低い優先順位を持たせることができます。

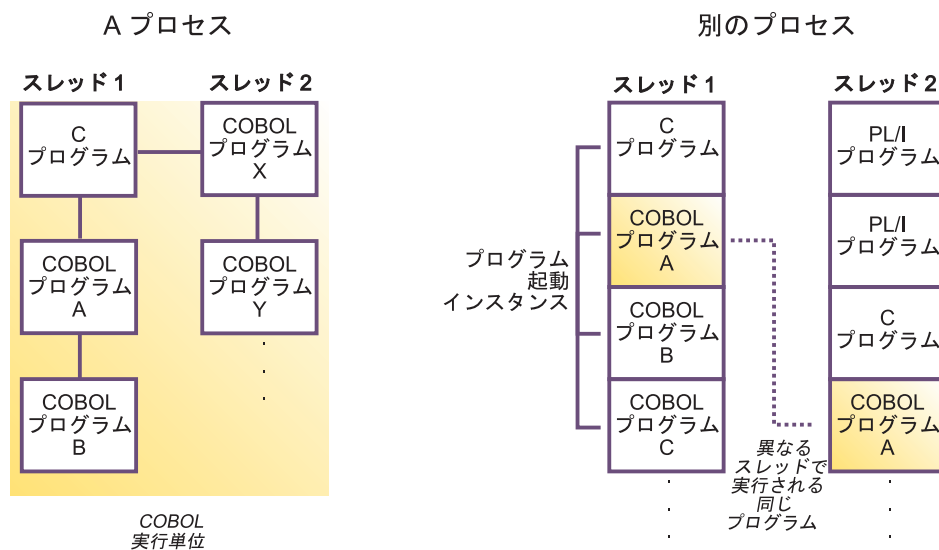
プロセス内で、アプリケーションは 1 つ以上のスレッドを開始することができます。スレッドはそれぞれ、そのスレッドを制御するコンピューター命令のストリームです。マルチスレッド化プロセスは、1 つの命令ストリーム (スレッド) で始ま

り、タスクを実行する他の命令ストリームを後で作成することができます。これらの複数のスレッドは並行して実行することができます。スレッド内では、実行プログラム間で制御権が移動します。

マルチスレッド化環境で、COBOL 実行単位 は、実行中のアクティブな COBOL プログラムが含まれているスレッドを含むプロセスの部分です。COBOL 実行単位は、スレッドのいずれかの実行スタックでアクティブな COBOL プログラムがなくなるまで継続します。例えば、呼び出される COBOL プログラムには GOBACK ステートメントが含まれていて、C プログラムに制御を戻します。実行単位内では、COBOL プログラムは非 COBOL プログラムを呼び出すことができ、逆に、非 COBOL プログラムは COBOL プログラムを呼び出すことができます。

スレッド内では、別々の COBOL および 非 COBOL プログラムの間で制御が移動します。例えば、COBOL プログラムは別の COBOL プログラムまたは C プログラムを呼び出すことができます。別々に呼び出されたプログラムはそれぞれ、プログラム起動インスタンス です。特定プログラムのプログラム起動インスタンスは、指定したプロセス内で複数のスレッドに存在することができます。

次の図に、プロセス、スレッド、実行単位、およびプログラム起動インスタンスの間の関係を示します。



マルチプロセッシングまたはマルチスレッド化とマルチタスキングを混同しないでください。後者は、通常、アプリケーションの外部動作を説明するために使用される用語です。マルチタスキングでは、オペレーティング・システムが複数のアプリケーションを同時に実行しているように見えます。マルチタスキングは COBOL で実装されるマルチスレッド化とは関係がありません。

551 ページの『例: マルチスレッド環境での COBOL の使用』

関連タスク

547 ページの『マルチスレッド化による言語要素の処理』

549 ページの『マルチスレッド化サポートのための THREAD の選択』

549 ページの『マルチスレッド化されたプログラムへの制御権移動』

550 ページの『マルチスレッド化されたプログラムの終了』
550 ページの『マルチスレッド化による COBOL 制限の処理』

関連参照

311 ページの『THREAD』

マルチスレッド化による言語エレメントの処理

COBOL プログラムは 1 つのプロセス内で複数のスレッドとして実行できるため、1 つの言語エレメントが、実行単位の有効範囲またはプログラム呼び出しインスタンスの有効範囲という 2 つの異なる有効範囲で解釈される可能性があります。これら 2 つの有効範囲のタイプは、参照できる項目の場所や、項目がストレージ内で持続する時間を判断する際に重要となります。

実行単位の有効範囲

COBOL 実行単位の実行中は、言語エレメントが持続し、スレッド内の他のプログラムが言語エレメントを使用することができます。

プログラムの呼び出しインスタンスの有効範囲

言語エレメントは、プログラム呼び出しの特定のインスタンス内でのみ持続します。

項目の参照は、その項目が宣言された有効範囲またはその項目が含まれる有効範囲から行うことができます。例えば、実行単位の有効範囲を持つデータ項目は、その実行単位内のプログラム呼び出しの任意のインスタンスから参照することができます。

ある項目は、それが宣言されている項目が持続する限り、ストレージ内に存続します。例えば、プログラム呼び出しインスタンスの有効範囲を持つデータ項目は、そのインスタンスが実行中の間だけストレージ内に存続します。

関連タスク

『実行単位の有効範囲を持つエレメントの処理』

548 ページの『プログラム呼び出しインスタンスの有効範囲を持つエレメントの処理』

関連参照

548 ページの『マルチスレッド化を使用した COBOL 言語エレメントの有効範囲』

実行単位の有効範囲を持つエレメントの処理

リソースに実行単位の有効範囲 (WORKING-STORAGE で宣言された GLOBAL データなど) がある場合は、アプリケーション内のロジックを使用して、複数のスレッドからそのデータへのアクセスを同期する必要があります。

次のいずれか、または両方の処置をとることができます。

- 実行単位の有効範囲を持つ複数のスレッド・リソースから同時にアクセスしないようにアプリケーションを構成します。
- 別々のスレッドから同時にリソースにアクセスできるようにするには、C またはプラットフォームの関数による機能を使用して、アクセスを同期します。

実行単位の有効範囲を持つリソースを、個々のプログラム呼び出しインスタンス (個々のデータ・コピーを持つプログラムなど) 内で分離させたい場合は、LOCAL-STORAGE SECTION でデータを定義します。これにより、そのデータがプログラム呼び出しインスタンスの有効範囲を持つようになります。

プログラム呼び出しインスタンスの有効範囲を持つ要素の処理

このような言語要素がある場合は、プログラム呼び出しインスタンスごとにストレージが割り振られます。したがって、1つのプログラムが複数のスレッド間で複数回呼び出される場合でも、プログラムが呼び出されるたびに、別個のストレージが割り振られます。

例えば、プログラム X が 2 つ以上のスレッドで呼び出される場合、呼び出される X の各インスタンスは専用のリソース・セット (ストレージなど) を取得します。

これらの言語要素と関連付けられたストレージは、プログラム呼び出しインスタンスの有効範囲を持つため、複数のスレッド間でデータにアクセスすることはできません。データ・アクセスの同期化を考慮する必要はありませんが、データを明示的に渡さない限り、プログラムの呼び出し間でこのデータを共用することはできません。

マルチスレッド化を使用した COBOL 言語要素の有効範囲

次の表に、さまざまな COBOL 言語要素の有効範囲をまとめます。

表 67. マルチスレッド化を使用した COBOL 言語要素の有効範囲

言語要素	参照元	存続時間
ADDRESS-OF 特殊レジスタ	関連レコードと同じ	プログラム起動インスタンス
ファイル	実行単位	実行単位
索引データ	プログラム	プログラム起動インスタンス
LENGTH OF 特殊レジスタ	関連 ID と同じ	関連 ID と同じ
LINAGE-COUNTER 特殊レジスタ	実行単位	実行単位
LINKAGE-SECTION データ	実行単位	基本データの有効範囲に基づく
LOCAL-STORAGE データ	スレッド内	プログラム起動インスタンス
RETURN-CODE	実行単位	プログラム起動インスタンス
SORT-CONTROL、SORT-CORE-SIZE、SORT-RETURN、TALLY 特殊レジスタ	実行単位	プログラム起動インスタンス
WHEN-COMPILED 特殊レジスタ	実行単位	実行単位
WORKING-STORAGE データ	実行単位	実行単位

マルチスレッド化サポートのための THREAD の選択

マルチスレッド化をサポートするために、THREAD コンパイラー・オプションを使用します。プログラムが 1 つのアプリケーション によって単一プロセス内の複数のスレッドで呼び出される場合には、THREAD を使用します。ただし、THREAD は、シリアライゼーション・ロジックが自動的に生成されるため、パフォーマンスに悪影響を与える可能性があります。

COBOL プログラムを複数のスレッドで実行するには、THREAD コンパイラー・オプションを使用して、アプリケーション内の COBOL プログラムのすべてをコンパイルしなければなりません。同じアプリケーション内では、THREAD を使用してコンパイルされたプログラムと NOTTHREAD を使用してコンパイルされたプログラムを混在させることはできません。

オブジェクト指向 (OO) のクライアントおよびクラスは、cob2_j コマンドを使用してコンパイルします。このコマンドにより、THREAD オプションがデフォルトの呼び出しオプションとして自動的に組み込まれます。

言語制限: THREAD オプションを使用する場合は、ある特定の言語エレメントを使用することができません。詳細は、下記の関連参照を参照してください。

再帰: THREAD コンパイラー・オプションを使用してプログラムをコンパイルした場合、スレッド化または非スレッド化環境で、そのプログラムを再帰的に呼び出すことができます。この再帰の機能は、PROGRAM-ID 段落で RECURSIVE 句を指定したかにかかわらず、使用可能です。

関連タスク

17 ページの『再帰的またはマルチスレッド化されたプログラムでのデータの共有』
257 ページの『オブジェクト指向アプリケーションのコンパイル』

関連参照

311 ページの『THREAD』

マルチスレッド化されたプログラムへの制御権移動

マルチスレッド化環境用に COBOL プログラムを書くときは、適切なプログラム・リンケージ・ステートメントを選択します。

単一スレッド環境の場合のように、実行単位内で最初に呼び出される時、および呼び出されるプログラムに対する CANCEL 後に最初に呼び出される時は、呼び出されるプログラムは初期状態にあります。

一般に、複数のスレッドを開始して管理するプログラムは、COBOL 事前初期設定インターフェースを使用することをお勧めします。

プログラムが、複数の COBOL スレッドを開始する場合 (例: C プログラムが COBOL プログラムを呼び出して、データの入出力を実行する場合)、COBOL プログラムが、必要でなくなったストレージの解放など、環境のクリーンアップを行うとは想定しないでください。特に、ファイルが自動的に閉じられるとは想定しない

でください。アプリケーションが COBOL クリーンアップを制御できるように、COBOL 環境を事前初期設定する必要があります。

関連概念

555 ページの『第 30 章 COBOL ランタイム環境の事前初期設定』

関連タスク

『マルチスレッド化されたプログラムの終了』

502 ページの『メインプログラムまたはサブプログラムの終了と再入』

マルチスレッド化されたプログラムの終了

GOBACK または EXIT PROGRAM を使用して、マルチスレッド・プログラムを終了させることができます。

プログラムの呼び出し側に戻るには、GOBACK を使用します。あるスレッド内の最初のプログラムから GOBACK を使用するとき、そのスレッドは終了します。

メインプログラムからの場合を除き、GOBACK と同様に、EXIT PROGRAM を使用します。メインプログラムでは EXIT PROGRAM は無効です。

COBOL 環境が事前初期設定されておらず、COBOL ランタイムが実行単位内に他の COBOL プログラムがアクティブではないと決定できる場合には、このスレッドの最初のプログラムから GOBACK が実行されたときに、実行単位を終了させる COBOL プロセス (すべての開かれている COBOL ファイルのクローズなど) が実行されます。この決定は、実行単位内で呼び出されるすべての COBOL プログラムが GOBACK または EXIT PROGRAM で呼び出し元に戻った場合に行うことができます。この決定は、次のような特定の状態では行うことができません。

- 1 つ以上のアクティブな COBOL プログラムがあるスレッドが終了された (例えば、例外のため、pthread_exit によって)。
- longjmp が実行され、呼び出しスタックのアクティブな COBOL プログラムが失敗した。

スレッド化環境で、STOP RUN を効果的に実行する COBOL 関数はありません。このような性質が必要な場合には、COBOL プログラムから C 出口機能呼び出しで、ランタイム終了出口の後に `_iwzCOBOLTerm` を使用することを検討してください。

関連タスク

502 ページの『メインプログラムまたはサブプログラムの終了と再入』

マルチスレッド化による COBOL 制限の処理

一部の COBOL アプリケーションは、サブシステムまたは他のアプリケーションに依存します。マルチスレッド化環境において、これらの依存性などに起因して、COBOL プログラムに対するいくつかの制限が発生します。

一般的に、実行単位内のアプリケーションに可視であるリソースへのアクセスを同期化する必要があります。この要件の例外には、DISPLAY および ACCEPT があります。これらは複数のスレッドから使用できます。これらに対する同期化は、すべてランタイム環境によって提供されます。

DB2: 複数のスレッドで DB2 アプリケーションを実行することができます。ただし、DB2 データにアクセスするために必要なすべての同期を提供する必要があります。

SORT と MERGE、および VSAM ファイル I/O: SORT と MERGE は、同時に 1 つのスレッドのみでアクティブになっている必要があります。同様に、VSAM ファイルの入出力も同時に 1 つのスレッドのみでアクティブになっている必要があります。ただし、COBOL ランタイム環境では、これらの制約のどちらも適用されません。そのため、アプリケーションでそうする必要があります。

関連タスク

515 ページの『再帰呼び出しの実行』

例: マルチスレッド環境での COBOL の使用

このマルチスレッド化のサンプルは、1 つの C メインプログラムと 2 つの COBOL プログラムで構成されています。

thrcob.c

2 つの COBOL スレッドを作成し、それらが完了してから終了する C メインプログラム

subd.cbl

thrcob.c で作成されたスレッドによって実行される COBOL プログラム

sube.cbl

thrcob.c で作成されたスレッドによって実行される 2 番目の COBOL プログラム

マルチスレッド化の例を作成および実行するには、AIX シェル・プロンプトで以下のコマンドを入力します。

- thrcob.c をコンパイルする: `xlc_r -c thrcob.c`
- subd.cbl をコンパイルする: `cob2_r -c -qTHREAD subd.cbl`
- sube.cbl をコンパイルする: `cob2_r -c -qTHREAD sube.cbl`
- 実行可能な thrcob を生成する: `cob2_r -o thrcob thrcob.o subd.o sube.o`
- プログラム thrcob を実行する: `thrcob`

thrcob.c のソース・コード

```
/* Creates COBOL threads, SUBD and SUBE.
   Threads must be created with undetached state for join to work. */
typedef int (*PFN)();
#include <pthread.h>
#include <stdio.h>
#include <setjmp.h>

extern int errno;
extern void *SUBD(void *);
```

```

extern void *SUBE(void *);

extern void _iwzCOBOLInit(int fcode, PFN StopFun, int *err_code, void *StopArg);
extern void _iwzCOBOLTerm(int fcode, int *err_code);

jmp_buf Jmpbuf;

int StopFun(long *stoparg)
{
    printf("inside StopFun. Got stoparg = %d\n", *stoparg);
    *stoparg = 123;
    longjmp(Jmpbuf,1);
}

long StopArg = 0;

void testrc(int rc, const char *s)
{
    if (rc != 0){
        printf("%s: Fatal error rc=%d\n",s,rc);
        exit(-1);
    }
}

void pgmy()
{
    pthread_t t1, t2;
    int rc;
    long i;
    long j;
    void *r1, *r2;
    pthread_attr_t attr;

    _iwzCOBOLInit(1, StopFun, &rc, &StopArg);
    printf( "_iwzCOBOLInit got %d\n",rc);
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr,PTHREAD_CREATE_UNDETACHED);

    i = 2000;
    rc = pthread_create( &t1, &attr, &SUBD, &i);
    testrc(rc,"create 1");
    j = 1000;
    rc = pthread_create( &t2, &attr, &SUBE, &j);
    testrc(rc,"create 2");
    printf("threads are %x and %x\n",t1, t2);

    rc = pthread_join(t1, &r1);
    printf("join %x got %d\n",t1,rc);
    rc = pthread_join(t2, &r2);
    printf("join %x got %d\n",t2,rc);

    _iwzCOBOLTerm(1, &rc);
    printf( "_iwzCOBOLTerm expects rc=0, got rc=%d\n",rc);
}

main()
{
    if (setjmp(Jmpbuf) ==0) (
        pgmy();
    )
}

```


subd.cbl のソース・コード

```
PROCESS PGMNAME(MIXED)
  IDENTIFICATION DIVISION
  PROGRAM-ID. "SUBD".
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL NAMES.
    DECIMAL-POINT IS COMMA.
  INPUT-OUTPUT SECTION.
  DATA DIVISION.
  FILE SECTION.
  Working-Storage SECTION.

  Local-Storage Section.
  01 n2 pic 9(8) comp-5 value 0.

  Linkage Section.
  01 n1 pic 9(8) comp-5.

  PROCEDURE DIVISION using by Reference n1.
    Display "In SUBD "

    perform n1 times
      compute n2 = n2 + 1
      Display "From Thread 1: " n2
      CALL "Sleep" Using by value 1000
    end-perform

  GOBACK.
```

sube.cbl のソース・コード

```
PROCESS PGMNAME(MIXED)
  IDENTIFICATION DIVISION.
  PROGRAM-ID. "SUBE".
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.
  INPUT-OUTPUT SECTION.
  DATA DIVISION.
  FILE SECTION.
  Working-Storage SECTION.

  Local-Storage Section.
  01 n2 pic 9(8) comp-5 value 0.

  Linkage Section.
  01 n1 pic 9(8) comp-5.

  PROCEDURE DIVISION using by reference n1.

    perform n1 times
      compute n2 = n2 + 1
      Display "From Thread 2: " n2
    *Go to sleep for 3/4 sec.
      CALL "Sleep" Using by value 750
    end-perform

  GOBACK.
```

第 30 章 COBOL ランタイム環境の事前初期設定

事前初期設定を行うと、アプリケーションが COBOL ランタイム環境を一度初期化した後、その環境を使用して複数の処理を実行してから、その環境を明示的に終了することができます。

事前初期設定を使用すると、C/C++ などの非 COBOL 環境から COBOL プログラムを複数回呼び出すことができます。

事前初期設定には、主に次のような利点があります。

- COBOL 環境を、いつでもプログラム呼び出し可能な状態にすることができる。

COBOL 実行単位内の最初の COBOL プログラムから戻っても、その実行単位は終了しないため、COBOL プログラムが非 COBOL 環境から呼び出される場合でも、最後に使われた状態で呼び出すことができます。

- パフォーマンスが高い。

COBOL ランタイム環境の作成と解除を何度も行うとオーバーヘッドが生じ、アプリケーションの処理速度が遅くなる可能性があります。

非 COBOL プログラムが、COBOL プログラムを最後に使われた状態で使用する必要がある場合は、複数言語アプリケーションに対して事前初期設定サービスを使用します。例えば、COBOL プログラムに対する最初の呼び出し時にファイルをオープンした場合、呼び出し側プログラムは、同じプログラムに対する後続の呼び出し時でも、そのファイルがオープンしていることを期待します。

制約事項: CICS では事前初期設定を行うことができません。

永続的な COBOL ランタイム環境の初期設定と終了を行うには、関連タスクで説明するインターフェースを使用します。事前初期設定された環境で使用される COBOL プログラムが共用ライブラリーに含まれている場合は、事前初期設定された環境が終了するまで、その共用ライブラリーを削除できません。

557 ページの『例: COBOL 環境の事前初期設定』

関連タスク

『永続的な COBOL 環境の初期設定』

556 ページの『事前初期設定された COBOL 環境の終了』

永続的な COBOL 環境の初期設定

以下のインターフェースを使用して、永続的な COBOL 環境を初期設定します。

CALL `init_routine` の構文

```
▶▶—CALL—init_routine(function_code,routine,error_code,token)————▶▶
```

CALL `init_routine` の呼び出し。呼び出しの作成元言語に適した言語エレメントを使用します。

init_routine

初期設定ルーチンの名前: `_iwzCOBOLInit` または `IWZCOBOLINIT`

function_code (入力)

値で渡される 4 バイトの 2 進数。`function_code` は次のようになります。

- 1 この関数呼び出しの後に呼び出される最初の COBOL プログラムは、サブプログラムとして扱われます。

routine (入力)

実行単位が終了した場合に呼び出されるルーチンのアドレス。この関数に渡されるトークン引数は、実行単位の終了出口ルーチンに渡されます。このルーチンは、実行単位の終了時に呼び出された際に、ルーチンの呼び出し側へ戻らないで、代わりに `longjmp()` または `exit()` を使用する必要があります。

出口ルーチン・アドレスを指定しない場合は、事前初期設定が失敗したことを示す `error_code` が生成されます。

error_code (出力)

4 バイトの 2 進数。`error_code` は次のようになります。

- 0 事前初期設定は正常に行われました。
- 1 事前初期設定は失敗しました。

token (入力)

上の実行単位の終了時にこのルーチンが呼び出された際に指定された出口ルーチンに渡される 4 バイトのトークン。

関連タスク

『事前初期設定された COBOL 環境の終了』

事前初期設定された COBOL 環境の終了

以下のインターフェースを使用して、事前初期設定された永続的な COBOL 環境を終了します。

CALL `term_routine` の構文

```
▶▶—CALL—term_routine(function_code,error_code)————▶▶
```

CALL *term_routine* の呼び出し。呼び出しの作成元言語に適した言語エレメントを使用します。

term_routine

終了ルーチンの名前: `_iwzCOBOLTerm` または `IWZCOBOLTERM`

function_code (入力)

値で渡される 4 バイトの 2 進数。*function_code* は次のようになります。

- 1 COBOL の STOP RUN ステートメントが実行されたかのように、事前初期設定された COBOL ランタイム環境をクリーンアップします。例えば、すべての COBOL ファイルがクローズされます。ただし、制御権はこのサービスの呼び出し元に戻ります。

error_code (出力)

4 バイトの 2 進数。*error_code* は次のようになります。

- 0 終了は正常に行われました。
- 1 終了は失敗しました。

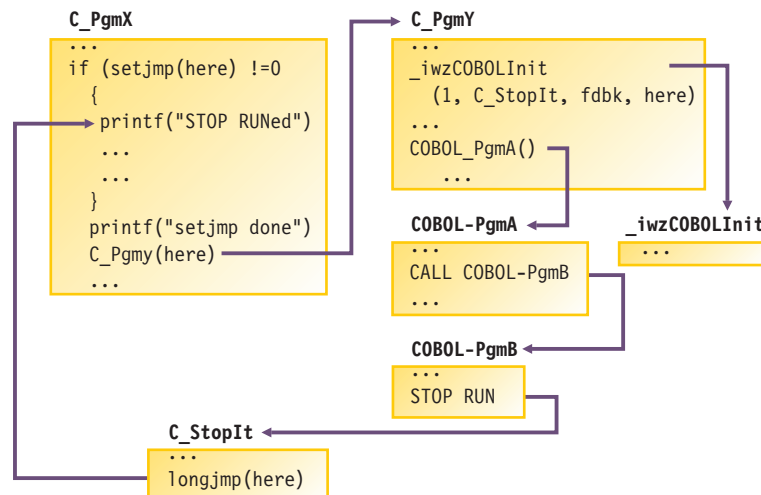
事前初期設定ルーチンの呼び出し後に呼び出される最初の COBOL プログラムは、サブプログラムとして扱われます。したがって、この (初期) プログラムからの GOBACK は、ファイルのクローズなどの実行単位の終了セマンティクスをトリガーしません。(STOP RUN などを使用した) 実行単位の終了では、実行単位の出口ルーチンが呼び出される前に、事前初期設定された COBOL 環境が解放されます。

アクティブでない場合: プログラムが終了ルーチンを呼び出した際に、COBOL 環境がまだアクティブでない場合は、この呼び出しは実行には影響を与えず、制御がエラー・コード 0 で呼び出し側に戻されます。

『例: COBOL 環境の事前初期設定』

例: COBOL 環境の事前初期設定

次の図は、事前初期設定された COBOL 環境の仕組みを示しています。この例では、C プログラムが COBOL 環境を初期設定し、COBOL プログラムを呼び出した後、COBOL 環境を終了します。



次の例は、COBOL 事前初期設定の使用法を示しています。C メインプログラムは、COBOL プログラム XIO を数回呼び出します。XIO の最初の呼び出しでファイルをオープンし、2 回目の呼び出しでレコードを 1 つ書き込みます (以下同様)。そして最後の呼び出しで、ファイルをクローズします。その後、C プログラムは C ストリーム I/O を使用して、このファイルをオープンし、読み取ります。

このプログラムをテストおよび実行するには、コマンド・シェルから次のコマンドを入力します。

```
xlc -c testinit.c
cob2 testinit.o xio.cbl
a.out
```

結果は次のとおりです。

```
_iwzCOBOLinit got 0
xio entered with x=0000000000
xio entered with x=0000000001
xio entered with x=0000000002
xio entered with x=0000000003
xio entered with x=0000000004
xio entered with x=0000000099
StopArg=0
_iwzCOBOLTerm expects rc=0 and got rc=0
FILE1 contains ----
11111
22222
33333
---- end of FILE1
```

この例では、実行単位が COBOL の STOP RUN で終了するのではなく、メインプログラムが `_iwzCOBOLTerm` を呼び出したときに終了している点に注意してください。

次の C プログラムは、ファイル `testinit.c` に入っています。

```
#ifdef _AIX
typedef int (*PFN)();
#define LINKAGE
#else
#include <windows.h>
#define LINKAGE _System
#endif

#include <stdio.h>
#include <setjmp.h>

extern void _iwzCOBOLInit(int fcode, PFN StopFun, int *err_code, void *StopArg);
extern void _iwzCOBOLTerm(int fcode, int *err_code);
extern void LINKAGE XIO(long *k);

jmp_buf Jmpbuf;
long StopArg = 0;

int LINKAGE
StopFun(long *stoparg)
{
    printf("inside StopFun\n");
    *stoparg = 123;
    longjmp(Jmpbuf,1);
}

main()
{
    int rc;
```

```

long k;
FILE *s;
int c;

if (setjmp(Jmpbuf) ==0) {
    _iwzCOBOLInit(1, StopFun, &rc, &StopArg);
    printf( "_iwzCOBOLInit got %d\n",rc);
    for (k=0; k <= 4; k++) XIO(&k);
    k = 99; XIO(&k);
}
else printf("return after STOP RUN\n");
printf("StopArg=%d\n", StopArg);
_iwzCOBOLTerm(1, &rc);
printf("_iwzCOBOLTerm expects rc=0 and got rc=%d\n",rc);
printf("FILE1 contains ---- %n");
s = fopen("FILE1", "r");
if (s) {
    while ( (c = fgetc(s) ) != EOF ) putchar(c);
}
printf("---- end of FILE1\n");
}

```

次の COBOL プログラムは、ファイル xio.cb1 に入っています。

```

IDENTIFICATION DIVISION.
PROGRAM-ID.    xio.
*****
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT file1 ASSIGN TO FILE1
    ORGANIZATION IS LINE SEQUENTIAL
    FILE STATUS IS file1-status.
. . .
DATA DIVISION.
FILE SECTION.
FD FILE1.
01 file1-id pic x(5).
. . .
WORKING-STORAGE SECTION.
01 file1-status pic xx    value is zero.
. . .
LINKAGE SECTION.
*
01 x                PIC S9(8) COMP-5.
. . .
PROCEDURE DIVISION using x.
. . .
    display "xio entered with x=" x
    if x = 0 then
        OPEN output FILE1
    end-if
    if x = 1 then
        MOVE ALL "1" to file1-id
        WRITE file1-id
    end-if
    if x = 2 then
        MOVE ALL "2" to file1-id
        WRITE file1-id
    end-if
    if x = 3 then
        MOVE ALL "3" to file1-id
        WRITE file1-id
    end-if

```

```
if x = 99 then
  CLOSE file1
end-if
GOBACK.
```

第 31 章 2 桁年の日付の処理

2000 年言語拡張 (MLE) を使用すれば、COBOL プログラムの簡単な変更を行うだけで日付フィールドを定義できます。コンパイラーは、世紀ウィンドウを使用して整合性を保証したうえで、これらの日付を認識し、作用します。

COBOL プログラムで日付の自動認識が行われるようにするには、以下のステップを実行してください。

1. プログラム内のデータ項目のうち、日付を含むデータ項目のデータ記述記入項目に DATE FORMAT 節を追加します。比較で使用されないものを含め、すべての日付を DATE FORMAT 節を使用して識別しなければなりません。
2. 日付を拡張する場合は、MOVE または COMPUTE ステートメントを使用して、ウィンドウ表示日付フィールドの内容を拡張日付フィールドにコピーします。
3. 必要であれば、DATEVAL および UNDATE 組み込み関数を使用して、日付フィールドと非日付の間で変換を行います。
4. YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウを固定ウィンドウまたはスライディング・ウィンドウとして設定します。
5. DATEPROC(FLAG) コンパイラー・オプションを使用してプログラムをコンパイルし、診断メッセージを検討して、日付処理によって予期しない副次作用が起こっていないかどうかを調べます。
6. コンパイルの結果、通知レベルの診断メッセージだけしかなければ、DATEPROC(NOFLAG) コンパイラー・オプションを使用してプログラムを再コンパイルして、簡潔なリストを作成することができます。

特定のプログラミング手法を使用して、日付処理を利用したり、日付フィールド使用の効果を制御することができます (例えば、日付の比較、日付によるソートとマージ、および日付を伴う算術演算の実行など)。2000 年言語拡張は、比較、移動と保管、増減など、日付フィールドに関してよく使われる操作について、年先頭型、年単独型、年末尾型の日付フィールドをサポートします。

関連概念

562 ページの『2000 年言語拡張 (MLE)』

関連タスク

564 ページの『日付に関連したロジック問題の解決』

570 ページの『年先行型、年単独型、および年末尾型の日付フィールドの使用』

573 ページの『リテラルを日付として操作する』

576 ページの『日付フィールドに対する算術の実行』

579 ページの『日付処理の明示的制御』

581 ページの『日付関連診断メッセージの分析および回避』

583 ページの『日付処理上の問題の回避』

関連参照

278 ページの『DATEPROC』

317 ページの『YEARWINDOW』

DATE FORMAT 節 (COBOL for AIX 言語解説書)

2000 年言語拡張 (MLE)

2000 年言語拡張 とは、西暦 2000 年以降の日付が関係するロジック問題に対処するのに役立つ、DATEPROC コンパイラー・オプションによってアクティブにされる COBOL for AIX の機能のことです。

使用可能にされた場合、言語拡張には以下のものが含まれます。

- DATE FORMAT 節。この節は、日付フィールドを識別し、日付における年部分の位置を指定するために、DATA DIVISION 内の項目に追加されます。

DATE FORMAT 節にはいくつかの使用上の制約事項があります。例えば、USAGE NATIONAL を持つ項目にこの節を指定することはできません。詳細については、以下の関連参照を参照してください。

- 以下の組み込み関数について、関数からの戻り値を日付フィールドとして再解釈すること。
 - DATE-OF-INTEGERS
 - DATE-TO-YYYYMMDD
 - DAY-OF-INTEGERS
 - DAY-TO-YYYYDDD
 - YEAR-TO-YYYY
- 次の形式の ACCEPT ステートメントにおける概念上のデータ項目 DATE、DATE YYYYMMDD、DAY、および DAY YYYYDDD を日付フィールドとして再解釈すること。
 - ACCEPT *identifier* FROM DATE
 - ACCEPT *identifier* FROM DATE YYYYMMDD
 - ACCEPT *identifier* FROM DAY
 - ACCEPT *identifier* FROM DAY YYYYDDD
- 組み込み関数 UNDATE と DATEVAL。これらは、日付フィールドおよび非日付の選択的再解釈に使用されます。
- 組み込み関数 YEARWINDOW。これは、YEARWINDOW コンパイラー・オプションによって設定された世紀ウィンドウの開始年を検索します。

DATEPROC コンパイラー・オプションは、識別された日付フィールドの特殊な日付中心処理を使用可能にします。YEARWINDOW コンパイラー・オプションは、2 桁のウィンドウ表示西暦年を解釈するために使用する 100 年ウィンドウ (世紀ウィンドウ) を指定します。

関連概念

563 ページの『この拡張の原則と目標』

関連参照

278 ページの『DATEPROC』

317 ページの『YEARWINDOW』

日付フィールドの使用に関する制約事項 (COBOL for AIX 言語解説書)

この拡張の原則と目標

2000 年言語拡張から最大のメリットが得られるようにするには、COBOL 言語にこれが導入された理由を理解する必要があります。

2000 年言語拡張が焦点を当てているのは、次の原則だけです。

- 日付のセマンティクスを使用して再コンパイルされるプログラムは、十分にテストされ、企業にとって価値のある資産です。関連する制限事項は、プログラムの 2 桁の年号が 1900-1999 の範囲に制限されることだけです。
- 日付の年号以外の部分には特別な処理は行われません。このため、サポートされる日付形式の年号以外の部分は X で表されます。それ以外の表記を使用すると、既存のプログラムの意味が変わってしまうことがあります。提供されている唯一の日付依存型セマンティクスは、プログラムの世紀ウィンドウに関して、日付の 2 桁の年部分を自動的に拡張（および短縮）するということです。
- 4 桁の年号部分を持つ日付が重要になるのは、通常、ウィンドウ表示日付と組み合わせて使用される場合です。それ以外の場合、4 桁年号日付と非日付の間に違いはほとんどありません。

上記の原則に基づき、2000 年言語拡張はいくつかの目標を達成するよう設計されています。日付処理問題を解決するために満足しなければならない目標を評価し、それらを 2000 年言語拡張の目標と比較して、アプリケーションがそこからどのように利益を得ることができるかを判断しなければなりません。新規アプリケーション、または既存アプリケーションへの拡張では、もっと後になるまで拡張できない古いデータをアプリケーションで使用している場合を除いて、拡張部分の使用を考慮してはなりません。

2000 年言語拡張の目標は、次のとおりです。

- 現在指定されているアプリケーション・プログラムの実用的な存続期間を拡張します。
- ソース変更を最小限にとどめ、可能であれば、DATA DIVISION の日付フィールドの宣言の増加だけに限定します。世紀ウィンドウ・ソリューションをインプリメントするためには、PROCEDURE DIVISION のプログラム・ロジックを変更する必要がないようにします。
- 日付フィールドを追加するときは、プログラムの既存のセマンティクスを保持しなければなりません。例えば、次のステートメントの場合のように、日付がリテラルとして表される場合、そのリテラルは、比較対象の日付フィールドと互換性があると見なされます（ウィンドウ操作または拡張されます）。

```
If Expiry-Date Greater Than 980101 . . .
```

既存のプログラムは、リテラルとして表された 2 桁年号の日付が 1900-1999 の範囲内にあると想定するので、拡張でこの想定が変更されることはありません。

- ウィンドウ操作機能は長期間の使用を目的としたものではありません。後で実施できる長期の解決策が採用されるまで、アプリケーションの実用的な存続期間を拡張することを意図しています。
- 拡張日付フィールド機能は、ファイルおよびデータベースの日付フィールドの拡張を支援するものとして、長期間の使用を意図しています。

拡張部分は、完全指定または完全な日付中心のデータ型に、認識できるセマンティクス (例えばグレゴリオ暦の月と日の部分) を与えません。拡張部分は、日付の年号部分に特別なセマンティクスを与えるだけです。

日付に関連したロジック問題の解決

日付処理問題を解決するための助けとして、3 つのアプローチ (世紀ウィンドウの使用、内部ブリッジング、または全フィールド拡張) のいずれかを採用することができます。

世紀ウィンドウ

世紀ウィンドウを定義し、ウィンドウ表示日付を含むフィールドを指定します。コンパイラーは、これらのデータ・フィールドの 2 桁の年号を世紀ウィンドウに従って解釈します。

内部ブリッジング

ファイルおよびデータベースはまだ 4 桁年の日付に移行されていないが、プログラムでは 4 桁に拡張した年のロジックを使用したい場合、そのような日付を 4 桁年の日付として処理するための内部ブリッジング手法を使用することができます。

全フィールド拡張

このソリューションは、2 桁の年の日付フィールドを、ファイルおよびデータベースの中で完全な 4 桁の年になるように明示的に拡張した後、そのフィールドをプログラムの中で拡張形式で使用方法です。これは、すべてのアプリケーションについて信頼できる日付処理がなされる唯一の方法です。

それぞれの方法で 2000 年言語拡張を使用して解決することができますが、以下に示すようにそれぞれに利点と欠点があります。

表 68. 2000 年問題のソリューションの利点および欠点

局面	世紀ウィンドウ	内部ブリッジング	全フィールド拡張
インプリメンテーション	速くて容易ですが、すべてのアプリケーションに合うわけではありません。	データ破壊のリスクがあります。	データベース、コピーブック、およびプログラムに対する変更をすべて同期させる必要があります。
テスト	プログラム・ロジックの変更はないので、テストはほとんど必要ありません。	プログラム・ロジックに行われる変更が直接的なものなので、テストは簡単です。	
修正期間	プログラムは 2000 年を過ぎても機能しますが、長期的なソリューションとはなり得ません。	プログラムは 2000 年を過ぎても機能しますが、永続的なソリューションとはなり得ません。	永続的なソリューションです。

表 68. 2000 年問題のソリューションの利点および欠点 (続き)

局面	世紀ウィンドウ	内部ブリッジング	全フィールド拡張
パフォーマンス	パフォーマンスが低下する可能性があります。	良好なパフォーマンスが得られます。	最適なパフォーマンスが得られます。
保守			保守が容易になります。

566 ページの『例: 世紀ウィンドウ』

567 ページの『例: 内部ブリッジング』

568 ページの『例: 拡張日付形式へのファイルの変換』

関連タスク

『世紀ウィンドウの使用』

566 ページの『内部ブリッジングの使用』

567 ページの『完全フィールド拡張への移行』

世紀ウィンドウの使用

世紀ウィンドウは、任意の 2 桁年が固有となる 100 年の間隔 (1950 から 2049 など) です。ウィンドウ表示日付フィールドについては、YEARWINDOW コンパイラー・オプションを使用して、世紀ウィンドウ開始日を指定することができます。

DATEPROC オプションが有効であると、コンパイラーは、プログラムの 2 桁の日付フィールドにこのウィンドウを適用します。例えば、1930-2029 の世紀ウィンドウの場合、COBOL は 2 桁の年号を次のように解釈します。

- 00 から 29 の年の値は、2000 から 2029 として解釈されます。
- 30 から 99 の年の値は、1930 から 1999 として解釈されます。

この世紀ウィンドウをインプリメントするには、プログラム内で DATE FORMAT 節を使用して日付フィールドを識別し、YEARWINDOW コンパイラー・オプションを使用して世紀ウィンドウを固定ウィンドウまたはスライディング・ウィンドウとして定義します。

- 固定ウィンドウの場合は、YEARWINDOW オプションの値として 1900 から 1999 の 4 桁の年号を指定します。例えば、YEARWINDOW(1950) は、1950-2049 の固定ウィンドウを定義します。
- スライディング・ウィンドウの場合は、YEARWINDOW オプションの値として -1 から -99 の負の整数を指定します。例えば、YEARWINDOW(-50) は、プログラムが稼働する年の 50 年前に始まるスライディング・ウィンドウを定義します。そのため、プログラムが 2008 年に稼働中である場合、世紀ウィンドウは 1958 から 2057 であり、2009 年には自動的に 1959 から 2058 になります。以下同様です。

コンパイラーは、識別されている日付フィールドに対する操作に、世紀ウィンドウを自動的に適用します。ウィンドウ操作をインプリメントするための余分のプログラム・ロジックは必要ありません。

566 ページの『例: 世紀ウィンドウ』

関連参照

278 ページの『DATEPROC』

317 ページの『YEARWINDOW』

DATE FORMAT 節 (COBOL for AIX 言語解説書)

日付フィールドの使用に関する制約事項 (COBOL for AIX 言語解説書)

例: 世紀ウィンドウ

次の例は、DATE FORMAT 節によってプログラムを変更して自動日付ウィンドウ操作の機能を使用する方法を (太字で) 示しています。

```
CBL LIB,QUOTE,NOOPT,DATEPROC(FLAG),YEARWINDOW(-60)
```

```
..  
01 Loan-Record.  
   05 Member-Number   Pic X(8).  
   05 DVD-ID          Pic X(8).  
   05 Date-Due-Back   Pic X(6) Date Format yyxxxx.  
   05 Date-Returned   Pic X(6) Date Format yyxxxx.  
..  
   If Date-Returned > Date-Due-Back Then  
     Perform Fine-Member.
```

PROCEDURE DIVISION に対する変更はありません。2つの日付フィールドに DATE FORMAT 節を追加した意味は、コンパイラーがそれらをウィンドウ表示日付フィールドとして認識し、したがって、IF ステートメントを処理するときには世紀ウィンドウを適用するということです。例えば、Date-Due-Back が 080102 (2008年1月2日) を含み、Date-Returned が 071231 (2007年12月31日) を含んでいる場合、Date-Returned は Date-Due-Back より小さいので (より以前なので)、プログラムは Fine-Member 段落を実行しません。(プログラムは、時間通りに DVD が戻されたかどうかを検査します。)

内部ブリッジングの使用

内部ブリッジングの場合、プログラムを適切に構成する必要があります。

以下の手順を実行します。

1. 2桁年号の日付を持つ入力ファイルを読み取る。
2. これらの2桁の日付をウィンドウ表示日付フィールドとして宣言し、コンパイラーがこれらの日付を自動的に4桁年号の日付に拡張できるように、これらを拡張日付フィールドに移動する。
3. プログラムの本体では、すべての日付処理に4桁年号の日付を使用する。
4. ウィンドウ操作により日付を2桁の年号に戻す。
5. 2桁年号の日付を出力ファイルに書き込む。

このプロセスは、完全拡張日付ソリューションへの便利な移行パスになり、ウィンドウ表示日付を使用するよりもパフォーマンス上の利点もあるかもしれません。

この手法を使用すると、プログラム・ロジックへの変更は最小で済みます。単に、日付を拡張および短縮するステートメントを追加し、また日付を参照するステートメントを、レコードの中の2桁年号のフィールドではなく WORKING-STORAGE の中の4桁年号の日付フィールドを使用するように変更するだけです。

出力のために日付を変換して 2 桁の年に戻しているため、年が世紀ウィンドウの範囲外になる可能性を考慮しておいてください。例えば、日付フィールドに 2020 年が入っているが、世紀ウィンドウが 1920 から 2019 である場合、日付は世紀ウィンドウの外側になります。年を 2 桁年フィールドに移動させるだけでは誤りになります。この問題が生じないようにするには、COMPUTE ステートメントを使用して日付を保管し、ON SIZE ERROR 句を指定して日付が世紀ウィンドウの外側であるかどうかを検出することができます。

『例: 内部ブリッジング』

関連タスク

565 ページの『世紀ウィンドウの使用』

576 ページの『日付フィールドに対する算術の実行』

『完全フィールド拡張への移行』

例: 内部ブリッジング

次の例は、プログラムを変更して内部ブリッジングをインプリメントする方法を(太字で)示しています。

```
CBL  DATEPROC(FLAG),YEARWINDOW(-60)
      . . .
      File Section.
      FD Customer-File.
      01 Cust-Record.
         05 Cust-Number      Pic 9(9) Binary.
         . . .
         05 Cust-Date        Pic 9(6) Date Format yyxxxx.
      Working-Storage Section.
      77 Exp-Cust-Date      Pic 9(8) Date Format yyyyxxxx.
      . . .
      Procedure Division.
      Open I-O Customer-File.
      Read Customer-File.
      Move Cust-Date to Exp-Cust-Date.
      . . .
      *=====*
      * Use expanded date in the rest of the program logic *
      *=====*
      . . .
      Compute Cust-Date = Exp-Cust-Date
      On Size Error
      Display "Exp-Cust-Date outside century window"
      End-Compute
      Rewrite Cust-Record.
```

完全フィールド拡張への移行

2000 年言語拡張を使用すれば、日付フィールドを完全に拡張するソリューションに向けて段階的に移行することができます。

以下の手順を実行します。

1. 世紀ウィンドウ・ソリューションを適用し、より永続的なソリューションをインプリメントするためのリソースが用意されるまで、このソリューションを使用する。
2. 内部ブリッジング・ソリューションを適用する。これは、ファイルでは 2 桁年号の形式で日付を保持した状態のまま、プログラムの中では拡張日付を使用する

方法です。プログラムの本体の中ではロジックにそれ以上の変更を行わないので、全フィールド拡張ソリューションにより容易に進むことができます。

3. ファイル設計およびデータベース定義を、4桁年号の日付を使用するように変更する。
4. COBOL コピーブックを、4桁年号の日付フィールドを反映するように変更する。
5. ユーティリティー・プログラム (または特殊な目的の COBOL プログラム) を実行して、古い形式のファイルから新しい形式にコピーする。
6. プログラムを再コンパイルし、レグレッション・テストおよび日付テストを行う。

最初の 2 つのステップを完了したら、それ以降のステップは何回でも繰り返すことができます。すべてのファイルのすべての日付フィールドを同時に変更する必要はありません。この方法では、段階的に変換するファイルを、業務上の必要性または他のアプリケーションとのインターフェースなどを基準にして選択することができます。

この方法を使用する場合、特別目的のプログラムを作成してファイルを拡張日付形式に変換する必要があります。

『例: 拡張日付形式へのファイルの変換』

例: 拡張日付形式へのファイルの変換

次の例は、日付フィールドを拡張しながら、あるファイルから別のファイルにコピーする簡単なプログラムを示しています。日付が拡張されるため、出力ファイルのレコード長は入力ファイルのレコード長より長くなっています。

```
CBL  LIB,QUOTE,NOOPT,DATEPROC(FLAG),YEARWINDOW(-80)
*****
** CONVERT - Read a file, convert the date    **
**          fields to expanded form, write   **
**          the expanded records to a new   **
**          file.                            **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID.  CONVERT.

ENVIRONMENT DIVISION.

INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT INPUT-FILE
           ASSIGN TO INFIL
           FILE STATUS IS INPUT-FILE-STATUS.

    SELECT OUTPUT-FILE
           ASSIGN TO OUTFIL
           FILE STATUS IS OUTPUT-FILE-STATUS.

DATA DIVISION.
FILE SECTION.
FD  INPUT-FILE
   RECORDING MODE IS F.
01  INPUT-RECORD.
   03  CUST-NAME.
       05  FIRST-NAME  PIC X(10).
       05  LAST-NAME   PIC X(15).
```



```

03 ACCOUNT-NUM      PIC 9(8).
03 DUE-DATE         PIC X(6) DATE FORMAT YYYYXX.   (1)
03 REMINDER-DATE   PIC X(6) DATE FORMAT YYYYXX.
03 DUE-AMOUNT      PIC S9(5)V99 COMP-3.

FD OUTPUT-FILE
RECORDING MODE IS F.
01 OUTPUT-RECORD.
03 CUST-NAME.
   05 FIRST-NAME   PIC X(10).
   05 LAST-NAME    PIC X(15).
03 ACCOUNT-NUM    PIC 9(8).
03 DUE-DATE       PIC X(8) DATE FORMAT YYYYXXXX.   (2)
03 REMINDER-DATE  PIC X(8) DATE FORMAT YYYYXXXX.
03 DUE-AMOUNT     PIC S9(5)V99 COMP-3.

WORKING-STORAGE SECTION.

01 INPUT-FILE-STATUS PIC 99.
01 OUTPUT-FILE-STATUS PIC 99.

PROCEDURE DIVISION.

    OPEN INPUT INPUT-FILE.
    OPEN OUTPUT OUTPUT-FILE.

READ-RECORD.
    READ INPUT-FILE
    AT END GO TO CLOSE-FILES.
    MOVE CORRESPONDING INPUT-RECORD TO OUTPUT-RECORD.   (3)
    WRITE OUTPUT-RECORD.

    GO TO READ-RECORD.

CLOSE-FILES.
    CLOSE INPUT-FILE.
    CLOSE OUTPUT-FILE.

EXIT PROGRAM.

END PROGRAM CONVERT.

```

注:

- (1) 入力レコード内のフィールド DUE-DATE と REMINDER-DATE は 2 桁年コンポネントを持つグレゴリオ日付です。これらのフィールドは DATE FORMAT 節で定義されているので、コンパイラーはこれらをウィンドウ表示日付フィールドとして認識します。
- (2) 出力レコードには、同じ 2 つのフィールドが拡張日付形式で入れられます。これらのフィールドは DATE FORMAT 節で定義されているので、コンパイラーはこれらを 4 桁年号の日付フィールドとして扱います。
- (3) MOVE CORRESPONDING ステートメントは、INPUT-RECORD 内の各項目を OUTPUT-RECORD 内の対応する項目に移動します。2 つのウィンドウ表示日付フィールドが対応する拡張日付フィールドに移動されると、コンパイラーは現行の世紀ウィンドウを使用して年号値を拡張します。

年先行型、年単独型、および年末尾型の日付フィールドの使用

年先行型または年単独型のどちらかのタイプの 2 つの日付フィールドを比較するとき、その 2 つの日付には互換性がなければなりません。すなわち、その 2 つは、年以外の文字の数は同じでなければなりません。年部分の桁数は同じである必要はありません。

年先行型日付フィールドは、DATE FORMAT 指定が、YY または YYYY とその後続く 1 つ以上の X で構成される日付フィールドです。年単独型日付フィールドの日付形式は YY または YYYY だけです。年末尾型日付フィールドは、DATE FORMAT 節で 1 つ以上の X とその後 YY または YYYY を指定している日付フィールドです。

年末尾型の日付形式は日付の表示によく使われますが、コンピューターにとってはあまり便利ではありません。それは、日付の中で最も比重の大きい部分である年が日付表現における比重の小さい部分になるからです。

機能的な年末尾型日付フィールドのサポートは、等しいか等しくないかの比較とある種の代入操作のみに限られます。オペランドは、日付形式が同一 (年末尾型) の日付であるか、または日付と非日付でなければなりません。コンパイラ自体には、年末尾型日付の操作のための自動ウィンドウ操作機能は含まれていません。年末尾型日付の算術演算など、サポートされていない方法で使用されると、コンパイラはエラー・レベル・メッセージを提供します。

年末尾型日付のためのより一般的な日付処理が必要な場合は、日付の年部分を分離して処理する必要があります。

571 ページの『例: 年先行型日付フィールドの比較』

関連概念

『互換性のある日付』

関連タスク

572 ページの『その他の日付形式の使用』

互換性のある日付

互換性のある日付 という用語の意味は、日付が DATA DIVISION で使用される場合と PROCEDURE DIVISION で使用される場合とでは異なります。

DATA DIVISION で使用する場合、日付フィールドの宣言、および従属データ項目や REDEFINES 節などの COBOL 言語エレメントを制御する規則が関係します。次の例では、Review-Year は Review-Date への従属データ項目として宣言することができるので、Review-Date と Review-Year には互換性があります。

```
01 Review-Record.  
   03 Review-Date           Date Format yxxxxx.  
   05 Review-Year Pic XX    Date Format yy.  
   05 Review-M-D Pic XXXX.
```

PROCEDURE DIVISION での使用は、比較、移動、算術式などの演算で日付フィールドを一緒に使用する方法に関連しています。年先行型および年単独型フィールドに互換性があるとみなされるには、日付フィールドに同じ数の年以外の文字が含まれて

いる必要があります。例えば、DATE FORMAT YYXXXX を持つフィールドは、同じ日付形式の別のフィールドや YYYYXXXX フィールドと互換性がありますが、YYXXXX フィールドとの互換性はありません。

年末尾型日付フィールドの場合は、DATE FORMAT 節が同じである必要があります。特に、ウィンドウ表示日付フィールドと拡張年末尾型日付フィールドとの相互演算は許されません。例えば、日付形式が XXXXY 日付フィールドを別の XXXXY 日付フィールドへ移動させることはできますが、XXXXYYY 形式の日付フィールドへ移動させることはできません。

演算の中の日付フィールドに互換性があれば、日付フィールドに対して、または日付フィールドと非日付が組み合わされたものに対して演算を実行できます。例えば、次のように定義されているとします。

```
01 Date-Gregorian-Win Pic 9(6) Packed-Decimal Date Format yyxxxx.  
01 Date-Julian-Win Pic 9(5) Packed-Decimal Date Format yyxxx.  
01 Date-Gregorian-Exp Pic 9(8) Packed-Decimal Date Format yyyyxxxx.
```

2 つのフィールドの年以外の数字の数が異なるので、次のステートメントは成立しません。

```
If Date-Gregorian-Win Less than Date-Julian-Win . . .
```

2 つのフィールドの年以外の数字の数が同じなので、次のステートメントは受け入れられます。

```
If Date-Gregorian-Win Less than Date-Gregorian-Exp . . .
```

この場合は、比較が確実に意味のあるものになるようにするために、世紀ウィンドウがウィンドウ表示日付フィールド (Date-Gregorian-Win) に適用されます。

非日付を日付フィールドと一緒に使用した場合、非日付は日付フィールドと互換性があると見なされるか、または単純な数値として扱われます。

例: 年先行型日付フィールドの比較

次の例は、拡張日付フィールドと比較されるウィンドウ表示日付フィールドを示しています。

```
77 Todays-Date Pic X(8) Date Format yyyyxxxx.  
01 Loan-Record.  
05 Date-Due-Back Pic X(6) Date Format yyxxxx.  
. . .  
If Date-Due-Back > Todays-Date Then . . .
```

世紀ウィンドウが Date-Due-Back に適用されます。Todays-Date に DATE FORMAT 節を指定して、それを拡張日付フィールドとして定義しなければなりません。そうしないと、そのフィールドは非日付フィールドとして扱われ、その結果、そのフィールドは年の桁数が Date-Due-Back と同じであるものと見なされます。コンパイラは 1900-1999 の仮定による世紀ウィンドウを適用するため、矛盾した比較が作成されます。

その他の日付形式の使用

自動ウィンドウ操作に適格であるためには、日付フィールドは、フィールドの最初の部分または唯一の部分として 2 桁年を含んでいる必要があります。フィールドの残り (ある場合) は、1 から 4 文字を含んでいる必要がありますが、どんな内容であるかは重要ではありません。

アプリケーションの中に、このような基準に合わない日付フィールドがある場合は、DATE FORMAT 節によって日付の年部分だけを日付フィールドとして定義するためのコード変更が必要かもしれません。このようなタイプの日付形式の例として、次のようなものがあります。

- 2 桁年、月の省略語を含む 3 文字、および日を表す 2 桁で構成される 7 文字フィールド。日付フィールドには 1 から 4 文字の年以外の文字しか使えないため、この形式はサポートされません。
- 形式 DDMMYY のグレゴリオ暦日付。年の部分が日付の最初の部分ではないので、自動ウィンドウ操作は適用されません。このような年末尾型日付は、SORT または MERGE ステートメントではウィンドウ表示キーとして全面的にサポートされており、また一部の COBOL 操作でもサポートされています。

このような場合に日付のウィンドウ操作を使用する必要があるなら、日付の年部分を分離するために何らかのコードを追加する必要があります。

例: 年の分離

次の例は、DDMMYY という形式である日付フィールドの年部分を切り離すにはどうすればよいとかを示しています。

```
03 Last-Review-Date Pic 9(6).
03 Next-Review-Date Pic 9(6).
. . .
Add 1 to Last-Review-Date Giving Next-Review-Date.
```

上記のコードでは、Last-Review-Date が 230108 (2008 年 1 月 23 日) を含んでいる場合、ADD ステートメントの実行後に Next-Review-Date には 230109 (2009 年 1 月 23 日) が入ることになります。これは、次の年次検査 (review) 日付を設定するための簡単な方法です。しかし、Last-Review-Date が 230199 の場合に 1 を加算すると 230200 になり、意図した結果にはなりません。

これらの日付フィールドは年が日付フィールドの最初の部分ではないので、年の部分を分離するための何らかのコードを追加しないと、DATE FORMAT 節を適用することはできません。次の例では、2 つの日付フィールドの年部分が分離されており、COBOL は世紀ウィンドウを適用して矛盾のない結果を維持できます。

```
03 Last-Review-Date Date Format xxxxyy.
   05 Last-R-DDMM Pic 9(4).
   05 Last-R-YY Pic 99 Date Format yy.
03 Next-Review-Date Date Format xxxxyy.
   05 Next-R-DDMM Pic 9(4).
   05 Next-R-YY Pic 99 Date Format yy.
. . .
Move Last-R-DDMM to Next-R-DDMM.
Add 1 to Last-R-YY Giving Next-R-YY.
```

リテラルを日付として操作する

ウィンドウ表示日付フィールドに、関連したレベル 88 条件名がある場合、VALUE 節のリテラルは、想定された世紀ウィンドウ 1900-1999 ではなく、コンパイル単位の世紀ウィンドウでウィンドウ操作が行われます。

例えば、次のようなデータ定義があるとします。

```
05 Date-Due      Pic 9(6)  Date Format yyxxxx.  
   88 Date-Target Value 081220.
```

世紀ウィンドウが 1950-2049 で、Date-Due の内容が 081220 (2008 年 12 月 20 日) であれば、以下の最初の条件は真に評価され、2 番目の条件は偽に評価されません。

```
If Date-Target. . .  
If Date-Due = 081220
```

リテラル 081220 は非日付として扱われるため、1900-1999 の仮定による世紀ウィンドウに対してウィンドウ操作され、1908 年 12 月 20 日を表すこととなります。しかし、レベル 88 の条件名の VALUE 節にリテラルが指定された場合には、そのリテラルは、それが付加されるデータ項目の一部となります。このデータ項目はウィンドウ表示日付フィールドなので、それが参照されるときには、必ず世紀ウィンドウが適用されます。

比較式の中で DATEVAL 組み込み関数を使用してリテラルを日付フィールドに変換することも可能です。結果の日付フィールドは、ウィンドウ表示日付フィールドまたは拡張日付フィールドとして扱われるので、比較に矛盾は生じません。例えば、上記の定義を使用すれば、以下の条件はどちらも真に評価されます。

```
If Date-Due = Function DATEVAL (081220 "YYXXXX")  
If Date-Due = Function DATEVAL (20081220 "YYYYXXXX")
```

レベル 88 条件名では、VALUE 節に THRU オプションを指定できますが、スライディング・ウィンドウではなく固定世紀ウィンドウを YEARWINDOW コンパイラー・オプションで指定する必要があります。以下に例を示します。

```
05 Year-Field Pic 99  Date Format yy.  
   88 In-Range Value 98 Thru 06.
```

この形式の場合、範囲内の 2 番目の項目のウィンドウ操作値は、1 番目の項目のウィンドウ操作値より大きくなければなりません。ただし、コンパイラーがこの差を検査できるのは、YEARWINDOW コンパイラー・オプションで固定世紀ウィンドウが指定されている場合のみです (例えば、YEARWINDOW(-68) ではなく YEARWINDOW(1940) が指定されている場合)。

ウィンドウ操作の順序に関する要件は、年末尾型日付フィールドには適用されません。年末尾型日付フィールドに対して THROUGH 句を含む条件名 VALUE 節を指定する場合、2 つのリテラルは通常の COBOL 規則に従っていなければなりません。つまり、最初のリテラルは 2 番目のリテラルより小さくなければなりません。

関連概念

574 ページの『仮定による世紀ウィンドウ』

575 ページの『非日付の処理』

関連タスク

579 ページの『日付処理の明示的制御』

仮定による世紀ウィンドウ

プログラムがウィンドウ表示日付フィールドを使用すると、コンパイラーは YEARWINDOW コンパイラー・オプションによって定義された世紀ウィンドウをコンパイル単位に適用します。ウィンドウ表示日付フィールドを非日付と一緒に使用する場合は、コンテキスト上、非日付をウィンドウ表示日付として扱う必要がある場合、コンパイラーは仮定による世紀ウィンドウを使用して非日付フィールドを解決します。

仮定による世紀ウィンドウは 1900-1999 であり、これは、一般にはコンパイル単位の世紀ウィンドウと同じではありません。

多くの場合、特にリテラルの非日付の場合、この仮定による世紀ウィンドウは正しい選択です。次の構成では、リテラルは、世紀ウィンドウが例えば 1975-2074 の場合でも、1972 年 1 月 1 日という元の意味を保たなければならない、2072 に変わるべきではありません。

```
01 Manufacturing-Record.  
   03 Makers-Date Pic X(6) Date Format yyxxxx.  
   . . .  
   If Makers-Date Greater than "720101" . . .
```

この仮定が正しくても、DATEVAL 組み込み関数を使用することによって年を明示し、警告レベルの診断メッセージ (仮定による世紀ウィンドウを適用することから生じるメッセージ) が出ないようにするのが適切です。

```
If Makers-Date Greater than  
   Function Dateval("19720101" "YYYYXXXX") . . .
```

時には、仮定が正しくない場合があります。次の例の場合、Project-Controls が COPY メンバーの中にあり、このメンバーが、西暦 2000 年処理用にまだアップグレードされていない他のアプリケーションによって使用され、このため Date-Target では DATE FORMAT 節を指定することができないものとします。

```
01 Project-Controls.  
   03 Date-Target Pic 9(6).  
   . . .  
01 Progress-Record.  
   03 Date-Complete Pic 9(6) Date Format yyxxxx.  
   . . .  
   If Date-Complete Less than Date-Target . . .
```

この例で、Date-Complete が Date-Target より前の日付 (より小) になるようにするには、次の 3 つの条件が真でなければなりません。

- 世紀ウィンドウが 1910-2009 である。
- Date-Complete が 991202 (グレゴリオ日付 1999 年 12 月 2 日) である。
- Date-Target が 000115 (グレゴリオ日付 2000 年 1 月 15 日) である。

ただし、Date-Target は、DATE FORMAT 節を持っていないので、非日付です。したがって、これに適用される世紀ウィンドウは、仮定による世紀ウィンドウ

1900-1999 であり、1900 年 1 月 15 日として処理されることとなります。この結果、Date-Complete は Date-Target より大きいことになり、これは意図した結果ではありません。

この場合には、DATEVAL 組み込み関数を使用して、この比較のために Date-Target を日付フィールドに変換すべきです。以下に例を示します。

```
If Date-Complete Less than
    Function Dateval (Date-Target "YYXXXX") . . .
```

関連タスク

579 ページの『日付処理の明示的制御』

非日付の処理

コンパイラーが非日付をどのように扱うかはコンテキストによって異なります。

以下の項目は非日付です。

- リテラル値。
- データ記述に DATE FORMAT 節が含まれていないデータ項目。
- 一部の算術式の結果 (中間または最終的)。例えば、2 つの日付フィールドの差は非日付ですが、日付フィールドと非日付の和は日付フィールドです。
- UNDATE 組み込み関数からの出力。

非日付を日付フィールドと一緒に使用する場合、コンパイラーは非日付を形式が日付フィールドと互換性のある日付、または単純な数値と解釈します。この解釈は、次に示すように、日付フィールドおよび非日付が使われたコンテキストによって異なります。

- 比較

日付フィールドを非日付と比較する場合、非日付は、年と年以外の文字の文字数の点で日付フィールドと互換性があると見なされます。次の例では、971231 という非日付リテラルをウィンドウ表示日付フィールドと比較します。

```
01 Date-1    Pic 9(6) Date Format yyxxxx.
. . .
    If Date-1 Greater than 971231 . . .
```

非日付リテラル 971231 は、Date-1 と同じ DATE FORMAT を持っているかのように処理されますが、開始年 (base year) は 1900 です。

- 算術演算

サポートされるすべての算術演算で、非日付フィールドは単純数値として処理されます。次の例では、Date-2 のグレゴリオ日付に数値 10000 を加算しています。つまり、日付に 1 年が加えられます。

```
01 Date-2    Pic 9(6) Date Format yyxxxx.
. . .
    Add 10000 to Date-2.
```

- MOVE ステートメント

日付フィールドを非日付に移動することはサポートされません。しかし、UNDATE 組み込み関数を使用してこれを行うことができます。

非日付を日付フィールドに移動する場合、送信フィールドは、年と年以外の文字の文字数の点で、受信フィールドと互換性があると見なされます。例えば、非日付をウィンドウ表示日付フィールドに移動する場合、非日付フィールドには 2 桁年と互換性のある日付が入っているものと想定されます。

符号条件の使用

アプリケーションの中には、日付フィールドで、トリガーとして機能する（つまり、ある特殊な処理が必要であることを知らせる）ゼロのような特殊値を使用するものがあります。

例えば、Orders ファイルで Order-Date に 0 の値を使用すると、レコードはオーダー・レコードではなく顧客合計レコードであることを意味するという場合です。プログラムは、次のように日付を 0 と比較します。

```
01 Order-Record.  
   05 Order-Date      Pic S9(5) Comp-3 Date Format yyxxx.  
   . . .  
   If Order-Date Equal Zero Then . . .
```

しかし、この比較は有効ではありません。リテラル値の Zero は非日付であるため、仮定による世紀ウィンドウに対してウィンドウ操作されて値は 1900000 になります。

あるいは、次のように、リテラル比較の代わりに符号条件を使用できます。符号条件を使用すると、Order-Date は非日付として扱われ、世紀ウィンドウは考慮に入れられません。

```
If Order-Date Is Zero Then . . .
```

このアプローチが適用されるのは、符号条件内のオペランドが算術式ではなく単純な ID である場合だけです。式を指定した場合は、適宜世紀ウィンドウが適用されてその式が最初に評価されます。そのあと、その式の結果に対して符号条件が比較されます。

代わりに UNDATE 組み込み関数を使用しても、同じ結果が得られます。

関連概念

575 ページの『非日付の処理』

関連タスク

579 ページの『日付処理の明示的制御』

関連参照

278 ページの『DATEPROC』

日付フィールドに対する算術の実行

任意の数値データ項目と同様に、数値日付フィールドにも算術操作を実行できます。必要に応じて、世紀ウィンドウが計算に使用されます。

しかし、算術式やステートメントのどこで日付フィールドを使用できるかについては制限があります。日付フィールドが含まれる算術演算は、次のものに限定されません。

- 日付フィールドへの非日付データの加算
- 日付フィールドからの非日付データの減算
- 互換性のある日付フィールドから日付フィールドを減算し、非日付の結果を与えること

以下の算術演算は使用できません。

- 互換性のない日付フィールド間の演算
- 2つの日付フィールドの加算
- 非日付データからの日付フィールドの減算
- 日付フィールドに適用される単項減算
- 日付フィールドの関係した乗算、除算、またはべき乗計算
- 年末尾型日付フィールドを指定する算術式
- 年末尾型日付フィールドを指定する算術式 (送信フィールドが非日付の場合に、受け取りデータ項目として指定する場合を除く)

日付フィールドの年部分については日付のセマンティクスが提供されていますが、年以外の部分については提供されていません。例えば、値 980831 を含むグレゴリオ暦のウィンドウ操作日付フィールドに 1 を加算すると、980901 ではなく 980832 の結果になります。

関連タスク

『ウィンドウ表示日付フィールドのオーバーフローの考慮』
578 ページの『評価の順序の指定』

ウィンドウ表示日付フィールドのオーバーフローの考慮

(年末尾型でない) ウィンドウ表示日付フィールドが算術演算に関与するときは、世紀ウィンドウに応じて、まずフィールドの年号構成要素の値が 1900 または 2000 だけ増分されたかのように処理されます。

```
01 Review-Record.  
   03 Last-Review-Year Pic 99 Date Format yy.  
   03 Next-Review-Year Pic 99 Date Format yy.  
   . . .  
   Add 10 to Last-Review-Year Giving Next-Review-Year.
```

上の例で、世紀ウィンドウが 1910-2009 で、Last-Review-Year の値が 98 である場合は、1998 を与えるため、まず Last-Review-Year が 1900 だけ増分されたかのようにして計算が進められます。次に、ADD 演算が実行され、2008 の結果が与えられます。この結果は、08 として Next-Review-Year に保管されます。

しかし、次のステートメントを使用すると、2018 という結果になります。

```
Add 20 to Last-Review-Year Giving Next-Review-Year.
```

この結果は、世紀ウィンドウの範囲の外側になります。結果が Next-Review-Year に保管されると、それ以降に Next-Review-Year が参照された場合に 1918 として

解釈されるので、誤りになります。この場合、演算の結果は、ON SIZE ERROR 句が ADD ステートメントに指定されているかどうかによって異なります。

- SIZE ERROR が指定されている場合、受信フィールドは変更されず、SIZE ERROR 命令ステートメントが実行されます。
- SIZE ERROR が指定されていない場合、結果は、左端から桁が切り捨てられて受信フィールドに保管されます。

この考慮事項は、内部ブリッジングを使用するときには大切です。出力ファイルに書き出すために 4 桁年号の日付フィールドを 2 桁に短縮するときは、日付が世紀ウィンドウの範囲内になるようにする必要があります。そうすると、2 桁年号の日付がフィールドで正しく表されるようになります。

適切な計算が行われるようにするには、短縮を行うための COMPUTE ステートメントに、ウィンドウ外条件を処理するための SIZE ERROR 句を指定しなければなりません。以下に例を示します。

```
Compute Output-Date-YY = Work-Date-YYYY  
On Size Error Perform CenturyWindowOverflow.
```

ウィンドウ表示日付受け取り側についての SIZE ERROR 処理は、世紀ウィンドウの範囲外となるすべての年号値を認識します。すなわち、世紀ウィンドウの開始年より小さい年号値も、世紀ウィンドウの終了年より大きい年号値の場合と同様に、SIZE ERROR 条件を引き起こします。

関連タスク

566 ページの『内部ブリッジングの使用』

評価の順序の指定

算術式の中の日付フィールドに関する制限のために、以前は正常にコンパイルされたプログラムが、今では一部のデータ項目が日付フィールドに変更された結果、診断メッセージが出るようになることがあります。

```
01 Dates-Record.  
   03 Start-Year-1 Pic 99 Date Format yy.  
   03 End-Year-1   Pic 99 Date Format yy.  
   03 Start-Year-2 Pic 99 Date Format yy.  
   03 End-Year-2   Pic 99 Date Format yy.  
   . . .  
   Compute End-Year-2 = Start-Year-2 + End-Year-1 - Start-Year-1.
```

上の例では、評価される最初の算術式は次のものです。

```
Start-Year-2 + End-Year-1
```

しかし、2 つの日付フィールドを加算することは許可されません。これらの日付フィールドを解決するためには、括弧を使用して、許可される算術式の部分を分離しなければなりません。以下に例を示します。

```
Compute End-Year-2 = Start-Year-2 + (End-Year-1 - Start-Year-1).
```

この場合、評価される最初の算術式は次のものです。

```
End-Year-1 - Start-Year-1
```

ある日付フィールドから別の日付フィールドを減算することは許可され、非日付の結果が与えられます。次に、この非日付の結果が日付フィールド End-Year-1 に加算され、日付フィールドの結果が与えられ、これが End-Year-2 に保管されます。

日付処理の明示的制御

ある特定条件下でのみ、あるいはプログラムの特定部分でのみ、COBOL データ項目を日付フィールドとして処理することがあります。あるいは、アプリケーションに含まれる 2 桁年の日付フィールドを、他のソフトウェア・プロダクトとの対話があるためにウィンドウ表示日付フィールドとして宣言できないことがあるかもしれません。

例えば、日付フィールドが、特に解釈のための操作をすることなく純粋な 2 進数の内容によってのみ認識されるコンテキストでは、そのフィールドの日付をウィンドウ操作することはできません。このような日付フィールドには、以下のものがあります。

- VSAM ファイルのキー
- DB2 のようなデータベース・システムの検索フィールド
- CICS コマンドのキー・フィールド

逆に日付フィールドを、プログラムの特定の部分では非日付として扱いたいときがあるかもしれません。

COBOL は、このような条件を扱うために 2 つの組み込み関数を提供します。

DATEVAL

非日付を日付フィールドに変換します。

UNDATE 日付フィールドを非日付に変換します。

関連タスク

『DATEVAL の使用』

580 ページの『UNDATE の使用』

DATEVAL の使用

DATEVAL 組み込み関数を使用して、非日付を日付フィールドに変換できます。その結果、COBOL により関連する日付処理がそのフィールドに適用されることとなります。

この関数の最初の引数には変換対象の非日付を指定し、2 番目の引数には日付形式を指定します。2 番目の引数は、DATE FORMAT 節の日付パターンの指定に類似した指定を持つリテラル・ストリングです。

ほとんどの場合コンパイラーは、非日付の解釈について正しい仮定をしますが、その仮定に伴って警告レベルの診断メッセージを出します。一般にこのメッセージは、ウィンドウ表示日付がリテラルと比較された場合に発生します。

```
03 When-Made          Pic x(6) Date Format yyxxxx.
. . .
If When-Made = "850701" Perform Warranty-Check.
```

リテラルは、互換性のあるウィンドウ表示日付と見なされますが、1900 から 1999 の世紀ウィンドウが使われているので、これは 1985 年 7 月 15 日を表しています。DATEVAL 組み込み関数を使用してリテラルの日付の年を明示して、警告メッセージを出さないようにすることができます。

```
If When-Made = Function Dateval("19850701" "YYYYXXXX")
    Perform Warranty-Check.
```

『例: DATEVAL』

UNDATE の使用

UNDATE 組み込み関数を使用して、日付フィールドを非日付に変換し、日付処理なしでそれを参照することができます。

重要: UNDATE を使用するの最後の手段の場合を除いて、できる限り使わないようにしてください。プログラムでの日付フィールドのフローをコンパイラが見失ってしまうからです。もし見失った場合、日付の比較が正しくウィンドウ操作されないことがあります。

MOVE および COMPUTE では、関数 UNDATE の代わりに DATE FORMAT 節を使用するようにしてください。

『例: UNDATE』

例: DATEVAL

この例は、フィールドを非日付としておいたまま、比較ステートメントの中で DATEVAL 組み込み関数を使用するのが適切である事例を示しています。

プログラム内でフィールド Date-Copied が何回も参照されるが、その参照の大半はレコード間でその値を移動したり、印刷のために再フォーマットしたりするだけのものであると想定します。1 つの参照箇所においてのみ、(他の日付との比較の目的で) その内容を日付であると見なしています。この場合、このフィールドは非日付としておいたまま、比較ステートメントの中で DATEVAL 組み込み関数を使用するのが適切です。以下に例を示します。

```
03 Date-Distributed Pic 9(6) Date Format yyxxxx.
03 Date-Copied      Pic 9(6).
. . .
If Function DATEVAL(Date-Copied "YYXXXX") Less than Date-Distributed . . .
```

この例では、DATEVAL は Date-Copied を日付フィールドに変換し、比較を意味のあるものにしていきます。

関連参照

DATEVAL (COBOL for AIX 言語解説書)

例: UNDATE

次の例は、日付フィールドを非日付に変換する事例を示しています。

フィールドは Invoice-Date は、ウィンドウ操作される年間通算日の日付フィールドです。レコードによっては、00999 という値を含んでいるものがあり、これはレコードが真の送り状レコードではなく、ファイル制御情報を含んでいるレコードであることを示します。

Invoice-Date には DATE FORMAT 節があります。プログラム内でのこのフィールドへの参照のほとんどが日付固有であるからです。しかし、制御レコードの有無が検査される場合には、年部分の値が 00 になっていると、何らかの混乱を招きます。Invoice-Date の 00 という年の値は、世紀ウィンドウに応じて、1900 または 2000 のいずれかを表すことがあります。これは、非日付 (例の中ではリテラル 00999) と比較されますが、それは、常に仮定による世紀ウィンドウに対してウィンドウ操作されるため、常に 1900 年を表します。

比較が矛盾した結果にならないようにするためには、UNDATE 組み込み関数を使用して Invoice-Date を非日付に変換しなければなりません。したがって、IF ステートメントがどの日付フィールドも比較しない場合には、ウィンドウ操作を適用させる必要はありません。以下に例を示します。

```
01 Invoice-Record.  
   03 Invoice-Date    Pic x(5) Date Format yyxxx.  
   . . .  
   If FUNCTION UNDATE(Invoice-Date) Equal "00999" . . .
```

関連参照

UNDATE (COBOL for AIX 言語解説書)

日付関連診断メッセージの分析および回避

DATEPROC(FLAG) コンパイラー・オプションが有効である場合、日付フィールドを定義または参照しているすべてのステートメントについて、コンパイラーは診断メッセージを作成します。

コンパイラー生成のすべてのメッセージと同じく、日付関連の各メッセージも以下の重大度レベルの 1 つを持っています。

- 通知レベル。これは、日付フィールドの定義または使用に注意を喚起するためのものです。
- 警告レベル。プログラムにコーディングされている情報が不十分なために、日付フィールドまたは非日付に関してコンパイラーがなんらかの仮定を設定したことを示すため、または正しいことを手作業で検査しなければならない日付ロジックの場所を示すためのものです。コンパイルは続行し、仮定は適用されます。
- エラー・レベル。日付フィールドの使い方が誤っていることを表します。コンパイルは継続されますが、ランタイムの結果は予測不能です。
- 重大レベル。日付フィールドの使い方が誤っていることを表します。このエラーが生成される原因となったステートメントは、コンパイルから廃棄されます。

MLE メッセージを最も簡単に使用するには、FLAG オプション設定を使用してコンパイルします。これによって、ソース・リストの中でメッセージの参照する行の直後にメッセージが出力されるようになります。すべての MLE メッセージを表示したり、重大度によって選択したりできます。

すべての MLE メッセージを表示するには、FLAG(I,I) および DATEPROC(FLAG) コンパイラー・オプションを指定します。自分のプログラム内の日付フィールドが MLE でどのように処理されるかを理解するため、最初はすべてのメッセージを表示することをお勧めします。例えば、コンパイル・リストを使用してプログラム内のデータ使用の静的分析をしたい場合は、FLAG (I,I) を使用してください。

しかし、MLE 固有のコンパイルでは FLAG(W,W) を指定することをお勧めします。重大レベル (S レベル) エラー・メッセージとエラー・レベル (E レベル) メッセージはすべて訂正する必要があります。警告レベル (W レベル) のメッセージについては、各メッセージを調べ、以下の指針に従って、メッセージを除去するか、または回避不能なメッセージの場合はコンパイラーが正しい仮定を行うようにしなければなりません。

- 診断メッセージが、DATE FORMAT 節を持っていないと見られる日付データ項目を示すことがあります。これらの項目に DATE FORMAT 節を追加するか、またはこれらの項目への参照で DATEVAL 組み込み関数を使用してください。
- 日付フィールドを対象にする比較条件の中、または日付フィールドを含む算術式の中でリテラルを使用する場合には、特別な注意が必要です。リテラル (および非日付データ項目) に DATEVAL 関数を使用して、使用する DATE FORMAT パターンを指定することができます。UNDATE 関数は、最後の手段として、日付中心の動作を望まないコンテキストで使用される日付フィールドを使用可能にするために使用することができます。
- REDEFINES および RENAMES 節が指定されている場合、日付フィールドと非日付が同じ保管場所を占有していると、コンパイラーは警告レベルの診断メッセージを出すことがあります。このような場合には注意深く調べて、種々の別名の付いたデータ項目のすべての使い方が正しいこと、および非日付と認識された再定義が実際にどれも日付でないこと、またはプログラム内の日付ロジックに悪影響を与えていないことを確認してください。

W レベル・メッセージが出ても気にしなければいいのですが、コードを変更して戻りコード = 0 のコンパイルを達成するのも良い方法です。

警告レベルの診断メッセージを回避するには、以下の簡単な指針に従ってください。

- 日付が入るデータ項目には DATE FORMAT 節を追加してください。その項目が比較で使用されない場合でもそのようにします。ただし、日付フィールドの使用上の制約事項に関しては、以下の関連参照を参照してください。例えば、暗黙的または明示的に USAGE NATIONAL として記述されているデータ項目では DATE FORMAT 節を使用できません。
- 日付フィールドが意味をなさないコンテキスト、例えば FILE STATUS、PASSWORD、ASSIGN USING、LABEL RECORD、または LINAGE 項目などでは、日付フィールドを指定しない。指定すると、警告レベルのメッセージが出されて、日付フィールドが非日付として扱われます。
- 日付フィールドの暗黙の別名または明示された別名が、日付フィールドだけからなるグループ項目などの中で互換性があることを確認する。
- 日付フィールドが VALUE 節を定義されている場合、その値が日付フィールド定義と互換性があることを確認する。

- 非日付を日付フィールドとして扱いたい場合 (例えば、非日付を日付フィールドに移動する場合や、ウィンドウ表示日付を非日付と比較する場合など) で、ウィンドウ表示日付比較を行う場合は、DATEVAL 組み込み関数を使用する。DATEVAL を使わないと、コンパイラーは非日付の使用に関してある仮定を行い、警告レベルの診断メッセージを作成します。その仮定が正しくても、DATEVAL を使用すればメッセージを排除できます。
- 日付フィールドが非日付として処理されるようにしたい場合は、UNDATE 組み込み関数を使用してください。例えば、日付フィールドを非日付へ移動させる場合や、ウィンドウ表示比較を行いたくないときに非日付とウィンドウ表示日付フィールドを比較する場合などです。

関連タスク

579 ページの『日付処理の明示的制御』

COBOL 2000 年言語拡張の手引き (日付関連診断メッセージの分析)

関連参照

日付フィールドの使用に関する制約事項 (COBOL for AIX 言語解説書)

日付処理上の問題の回避

COBOL プログラムを変更して 2000 年言語拡張を使用する場合、予想外の振る舞いが生じるのを解決するために、プログラムの一部に特別の注意を向けなければならないことがあります。例えば、パック 10 進フィールドの問題、および拡張フィールドからウィンドウ日付フィールドに移行させるときに発生する問題を回避する必要が生じることがあります。

関連タスク

『パック 10 進数フィールドの問題の回避』

584 ページの『拡張日付フィールドからウィンドウ表示日付フィールドへの移動』

パック 10 進数フィールドの問題の回避

COMPUTATIONAL-3 フィールド (パック 10 進数形式) は、奇数桁数を持つものとして定義されることがよくあります。フィールドがその大きさの数値を保持しない場合でもそうです。それは、内部表現ではパック 10 進数の桁数が常に奇数になっているからです。

例えば、6 桁のグレゴリオ暦日付を入れるフィールドを PIC S9(6) COMP-3 として宣言できます。この宣言によって、4 バイトのストレージが予約されることとなります。しかし、プログラマーは、最高次の桁が常に 0 の 4 バイトが予約されることを考慮して、PIC S9(7) としてフィールドを宣言している可能性があります。

このフィールドに DATE FORMAT YYXXXX という節を追加した場合、コンパイラーから診断メッセージが出ます。PICTURE 節内の桁数が日付形式指定のサイズに一致しないからです。その場合、フィールド使用を 1 つずつ慎重に調べる必要があります。高位桁を使わない場合は、単にフィールド定義を PIC S9(6) に変更することができます。使用する場合 (例えば同じフィールドに日付以外の値も入れることがある場合)、他のなんらかのアクションが必要になります。

- REDEFINES 節を使用して、日付および非日付の両方としてフィールドを定義する (この場合も、警告レベルの診断メッセージが出ます)。
- 日付を入れる別の WORKING-STORAGE フィールドを定義し、数値フィールドを新規フィールドに移動する。
- データ項目に DATE FORMAT 節を追加せず、それを日付フィールドとして参照するところで DATEVAL 組み込み関数を使用する。

拡張日付フィールドからウィンドウ表示日付フィールドへの移動

拡張英数字日付フィールドをウィンドウ表示日付フィールドへ移動させる場合、この移動は、英数字移動に関する通常の COBOL 規則に従いません。送信フィールドと受信フィールドのどちらも日付フィールドである場合、移動は通常の左寄せではなく右寄せになります。拡張からウィンドウ操作への (縮小) 移動の場合、年の先頭の 2 桁が切り捨てられることになります。

送信フィールドの内容によっては、このような移動は結果的に誤りを生じることがあります。以下に例を示します。

```
77 Year-Of-Birth-Exp Pic x(4) Date Format yyyy.  
77 Year-Of-Birth-Win Pic xx Date Format yy.  
...  
  Move Year-Of-Birth-Exp to Year-Of-Birth-Win.
```

Year-Of-Birth-Exp が '1925' を含んでいる場合、Year-Of-Birth-Win は '25' が含まれます。しかし、世紀ウィンドウが 1930-2029 の場合、Year-Of-Birth-Win のあとの参照は 2025 と見なされます。しかし、それは誤っています。

第 8 部 パフォーマンスおよび生産性の向上

第 32 章 プログラムのチューニング	587
最適なプログラミング・スタイルの使用	588
構造化プログラミングの使用	588
一括表示表現	588
シンボリック定数の使用	589
定数計算のグループ化	589
重複計算のグループ化	589
効率的なデータ型の選択	590
効率的な計算データ項目の選択	590
一貫性のあるデータ型の使用	591
算術式の効率化	591
指数計算の効率化	591
テーブルの効率的処理	592
テーブル参照の最適化	593
定数項目と変数項目の最適化	594
重複項目の最適化	594
可変長項目の最適化	595
直接指標付けと間接指標付けの比較	595
コードの最適化	595
最適化	596
含まれているプログラム・プロシージャの統合	596
パフォーマンスを向上させるコンパイラ機能の選択	597
パフォーマンスに関連するコンパイラ・オプション	597
パフォーマンスの評価	600
第 33 章 コーディングの単純化	601
反復コーディングの除去	601
例: COPY ステートメントの使用	602
日時の取り扱い	603
日時呼び出し可能サービスからのフィードバックの取得	604
日時の呼び出し可能サービスからの条件の処理	604
例: 日付の操作	604
例: 出力用の日付形式	605
フィードバック・トークン	606
ピクチャー文字項およびストリング	607
例: 日時のピクチャー・ストリング	609
世紀ウィンドウ	610
例: 世紀ウィンドウの照会および変更	610

第 32 章 プログラムのチューニング

プログラムが分かりやすいものであってこそ、パフォーマンスの評価を行うことができます。制御フローが混乱したプログラムは、理解や維持が困難です。また、制御フローが混乱していると、コードの最適化も禁止されます。

このため、パフォーマンスの向上を直接試みる前に、プログラムのいくつかの局面を評価する必要があります。

1. プログラムの基礎アルゴリズムを調べる。最高のパフォーマンスを得るためには、適切なアルゴリズムが不可欠です。例えば、百万個の品目をソートするような洗練されたアルゴリズムは、単純なアルゴリズムよりも何百万倍も高速になります。
2. データ構造を調べる。データ構造はアルゴリズムに適したものにする必要があります。プログラムが頻繁にデータにアクセスする場合は、可能であれば、データにアクセスするために必要なステップの数を減らします。
3. アルゴリズムとデータ構造を改善したら、パフォーマンスに影響を与える COBOL ソース・コードのその他の詳細を調べる。

より優れたコード・シーケンスを生成し、システム・サービスをより活用するようなプログラムを作成することができます。プログラムのパフォーマンスに影響を与えるのは、次の分野です。

- コーディング技法。これには、最適化プログラムを援助するプログラミング・スタイルの使用、効率的なデータ型の選択、およびテーブルの効率的な処理が含まれます。
- 最適化。OPTIMIZE コンパイラー・オプションを使用してコードを最適化することができます。
- コンパイラー・オプションおよび USE FOR DEBUGGING ON ALL PROCEDURES。特定のコンパイラー・オプションおよび言語は、プログラムの効率に影響を与えません。
- ランタイム環境。どのランタイム・オプションを選択するかについて、またコンパイルされたプログラムの実行方法を制御するその他のランタイム考慮事項について、注意深く考慮してください。
- CICS のもとでの実行 EXEC CICS LINK のインスタンスを CALL に変換して、トランザクションの応答時間を改善します。

関連概念

596 ページの『最適化』

関連タスク

588 ページの『最適なプログラミング・スタイルの使用』

590 ページの『効率的なデータ型の選択』

592 ページの『テーブルの効率的処理』

595 ページの『コードの最適化』

597 ページの『パフォーマンスを向上させるコンパイラー機能の選択』

151 ページの『SFS パフォーマンスの向上』

関連参照

597 ページの『パフォーマンスに関連するコンパイラー・オプション』

325 ページの『第 16 章 ランタイム・オプション』

最適なプログラミング・スタイルの使用

使用するコーディング・スタイルは、最適化プログラムがコードを処理する方法に影響を与えることがあります。構造化プログラミング手法の使用、式の因数処理、シンボリック定数の使用、および定数と重複計算のグループ化によって、最適化を向上させることができます。

関連タスク

『構造化プログラミングの使用』

『一括表示表現』

589 ページの『シンボリック定数の使用』

589 ページの『定数計算のグループ化』

589 ページの『重複計算のグループ化』

構造化プログラミングの使用

構造化プログラミング・ステートメント (EVALUATE やインライン PERFORM) を使用すると、プログラムが一層分かりやすいものになり、より直線的な制御フローができあがります。その結果、最適化プログラムはプログラムのより多くの領域に作用することができるため、より効率のよいコードが与えられます。

トップダウン・プログラミング構成を使用してください。ライン外の PERFORM ステートメントは、トップダウン・プログラミングを行う本来の手段です。ライン外 PERFORM ステートメントがインラインの PERFORM ステートメントと同じくらい効率的になることがよくあります。これは、最適化プログラムがリンケージ・コードを簡略化または除去するためです。

次の構成は使用しないでください。

- ALTER ステートメント
- 逆方向ブランチ (PERFORM が不適當であるループに必要な場合を除く)
- 変則的な制御フローを伴う PERFORM プロシージャ。例えば、プロシージャの終わりに制御が渡されないために、PERFORM ステートメントに戻れないなど。

一括表示表現

プログラム内の式を因数処理することによって、多数の不要な計算を除去できる可能性があります。

例えば、次のコードの最初のブロックは、2 番目のブロックより効率的になっています。

```
MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
  COMPUTE TOTAL = TOTAL + ITEM(I)
END-PERFORM
COMPUTE TOTAL = TOTAL * DISCOUNT
```

```
MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
  COMPUTE TOTAL = TOTAL + ITEM(I) * DISCOUNT
END-PERFORM
```

最適化プログラムは式の因数処理を行いません。

シンボリック定数の使用

プログラム全体で最適化プログラムがデータ項目を定数として認識するようにさせるには、データ項目を VALUE 節で初期化し、プログラム内のどの場所でもそれを変更しないでください。

データ項目を BY REFERENCE によってサブプログラムに渡すと、最適化プログラムはその項目を外部データ項目と見なして、サブプログラムを呼び出すたびに、その項目が変更されるものと想定します。

リテラルをデータ項目に移動すると、最適化プログラムはそのデータ項目を定数と見なしますが、それは、MOVE ステートメントに続くプログラムの限られた領域内においてのみです。

定数計算のグループ化

式のいくつかの項目が定数である場合、最適化プログラムがそれらの項目を最適化できることを確認してください。コンパイラーは COBOL の左から右への評価規則に従います。したがって、すべての定数を式の左側に移動させるか、または括弧で囲んでグループ化します。

例えば、V1、V2、および V3 が変数で、C1、C2、および C3 が定数の場合、下記の左側にある式の方が、右側の対応する式より優れています。

効率がよい

```
V1 * V2 * V3 * (C1 * C2 * C3)
C1 + C2 + C3 + V1 + V2 + V3
```

効率がよくない

```
V1 * V2 * V3 * C1 * C2 * C3
V1 + C1 + V2 + C2 + V3 + C3
```

量産用プログラミングでは、式の右側に定数因子を置く傾向がよく見られます。しかしその結果、最適化が行われないうえに、効率のよくないコードが生成される可能性があります。

重複計算のグループ化

さまざまな式のコンポーネントが重複している場合は、コンパイラーがそれらを最適化できることを確認してください。算術式の場合、コンパイラーは、左から右への COBOL の評価規則に従います。したがって、すべての重複を式の左側に移動させるか、または括弧で囲んでグループ化します。

例えば、V1 から V5 が変数の場合、 $V2 * V3 * V4$ の計算は、次の 2 つのステートメントで重複します (共通の副次式と呼ばれます)。

```
COMPUTE A = V1 * (V2 * V3 * V4)
COMPUTE B = V2 * V3 * V4 * V5
```

次の例では、 $V2 + V3$ が共通の副次式です。

```
COMPUTE C = V1 + (V2 + V3)
COMPUTE D = V2 + V3 + V4
```

次の例には、共通の副次式はありません。

```
COMPUTE A = V1 * V2 * V3 * V4
COMPUTE B = V2 * V3 * V4 * V5
COMPUTE C = V1 + (V2 + V3)
COMPUTE D = V4 + V2 + V3
```

最適化プログラムは重複する計算を除去できます。人工的な一時計算を取り入れる必要はありません。そのようなものがなくても多くの場合、プログラムはずっと分かりやすくなります。

効率的なデータ型の選択

適切なデータ型および PICTURE 節を選択すると、より効率的なコードを得ることができますが、それは USAGE DISPLAY および USAGE NATIONAL データ項目を、計算に頻繁に使用される領域で使用しない場合に効率的なコードが得られるのと同様です。

一貫性のあるデータ型を使用すると、データ項目に演算を実行する際に変換を行う必要性を減らすことができます。また、固定小数点データ型と浮動小数点データ型をいつ使用するかを慎重に判別することで、プログラム・パフォーマンスを向上させることができます。

関連概念

45 ページの『数値データの形式』

関連タスク

『効率的な計算データ項目の選択』

591 ページの『一貫性のあるデータ型の使用』

591 ページの『算術式の効率化』

591 ページの『指数計算の効率化』

効率的な計算データ項目の選択

データ項目を主に算術に、または添え字として使用する場合、その項目のデータ記述項目に USAGE BINARY をコーディングしてください。2 進データを処理する操作は、10 進データを処理する操作よりも高速で行われます。

しかし、固定小数点算術ステートメントが大きな精度 (有効数字) の中間結果を持つ場合、コンパイラーは、オペランドをパック 10 進数形式に変換したうえで、10 進数演算を使用します。固定小数点算術ステートメントについては、コンパイラーは通常、精度が 8 桁以下にとどまる場合には、2 進オペランドを使用した簡単な計算に 2 進数算術演算を使用します。18 桁を超えると、コンパイラーは常に 10 進数算術演算を使用します。9 から 18 桁の精度では、コンパイラーはいずれの形式でも使用できます。

BINARY データ項目について最も効率的なコードを作成するには、以下の特性を持たせるようにしてください。

- 符号 (PICTURE 節の S)。
- 8 桁以下。

8 桁より大きいデータ項目、あるいは DISPLAY または NATIONAL データ項目と一緒に使用されるデータ項目の場合は、PACKED-DECIMAL を使用してください。

PACKED-DECIMAL データ項目について最も効率的なコードを作成するには、以下の特性を持たせるようにしてください。

- 符号 (PICTURE 節の S)。
- ハーフ・バイトを残さずに正確なバイト数を占めるように、奇数の桁数 (PICTURE 節の 9 の数)。

一貫性のあるデータ型の使用

種々の型のオペランドに対する操作では、オペランドの 1 つを残りのものと同じ型に変換する必要があります。各変換ごとにいくつかの命令が必要になります。例えば、いずれかのオペランドは小数点以下の桁数が適切な数になるように位取りを指定する必要があるかもしれません。

一貫性のあるデータ型を使用し、両方のオペランドに同じ使用法を与え、さらに適切な PICTURE 指定を与えることで、変換を大部分は回避できます。つまり、比較、加算、または減算を行う 2 つの数値は、同じ使用法を持つだけでなく、さらに小数部の桁数 (PICTURE 節の V の後の 9 の数) も同じでなければなりません。

算術式の効率化

オペランドをほとんど変換する必要がない場合、浮動小数点で評価される算術式の計算は最も効率的です。COMP-1 または COMP-2 であるオペランドを使用すると、最も効率のよいコードが作成されます。

浮動小数点データに高速変換を行うには、整数項目を BINARY または PACKED-DECIMAL (9 桁以下) として宣言します。さらに、COMP-1 または COMP-2 の項目から、9 桁以下の固定小数点整数への変換は (SIZE ERROR が有効ではない場合)、COMP-1 または COMP-2 項目の値が 1,000,000,000 未満の場合に効率がよくなります。

指数計算の効率化

評価をもっと速く行い、結果をもっと正確にするには、大きな指数に対しては指数の浮動小数点を使用してください。

例えば、以下に示す最初のステートメントは、2 番目のステートメントより速くかつより正確に計算されます。

```
COMPUTE fixed-point1 = fixed-point2 ** 100000.E+00
```

```
COMPUTE fixed-point1 = fixed-point2 ** 100000
```

これは、浮動小数点の指数があるため、べき乗計算の計算に浮動小数点演算が使用されるからです。

テーブルの効率的処理

いくつかの手法を使用して、テーブル処理演算の効率を上げたり、最適化プログラムに影響を及ぼしたりすることができます。特に、テーブル処理演算がアプリケーションの主要部分を占めているときなどは、努力の成果が十分報われる可能性があります。

以下の 2 つのガイドラインは、テーブル・エレメントの参照方法を選択する際に影響を与えるものです。

- 添え字付けではなく索引付けを使用する。

コンパイラーは重複する指標や添え字を除去できますが、テーブル・エレメントへの元の参照は、指標を使用することで (たとえ添え字が BINARY であっても) より効率的になります。これは、指標の値には既にエレメント・サイズが加味されているのに対して、添え字の値は使用時にエレメント・サイズを乗算しなければならないためです。指標には既にテーブルの先頭からの変位が含まれており、実行時にこの値を計算する必要はありません。ただし、添え字の方が理解しやすく維持するのが簡単かもしれません。

- 相対索引付けを使用する。

相対指標参照 (つまり、符号なし数値リテラルが指標名に加えられるか、または指標名から引かれる参照) は、少なくとも直接指標参照と同じ位の速さで、時にはより高速で実行されます。オフセットを含めた代替索引を保管してもメリットはありません。

指標または添え字のいずれを使用する場合でも、以下のコーディング指針は、より良いパフォーマンスを得る助けとなります。

- 定数および重複する指標または添え字を左側に置く。

このように実行時の計算を削減または除去することができます。すべての指標または添え字が可変であっても、プログラム内で互いに近接している参照に関して、右端の添え字がもっとも頻繁に変化するよう、テーブルを使用してみてください。この方法を使用すると、ページングだけでなくストレージ参照のパターンも改善されます。すべての指標または添え字が重複している場合、指標または添え字の計算全体が共通副次式になります。

- 関連するテーブルの長さとも一致するようなエレメントの長さを指定する。

異なるテーブルに添え字または指標を付けるときは、すべてのテーブルのエレメント長が同じ場合に、最も効率がよくなります。このように、テーブルの最後の次元のストライドが同じであるため、最適化プログラムは、1 つのテーブルで計算された右端の指標または添え字を再利用できるようになります。エレメントの長さおよび各次元での出現回数が同じである場合、最後の次元以外は、次元のストライドもまた等しくなり、その結果、添え字計算相互間の共通性はより大きくなります。最適化プログラムは、右端以外の添え字または指標を再使用することができます。

- 指標および添え字検査をプログラムにコーディングすることによって、参照エラーを回避する。

指標および添え字を妥当性検査する必要がある場合は、SSRANGE コンパイラー・オプションを使用するよりも、独自の検査をコーディングする方が速い場合があります。

以下のガイドラインに従うことによって、テーブルの効率を改善することもできます。

- すべての添え字に 2 進数データ項目を使用する。

添え字を使用してテーブルをアドレッシングする場合は、8 桁以下の BINARY 符号付きデータ項目を使用してください。さらに場合によっては、データ項目の桁数を 4 桁以下にすると、処理時間を短縮できます。

- 可変長テーブル項目に 2 進数データ項目を使用する。

可変長項目を持つテーブルの場合は、OCCURS DEPENDING ON (ODO) のコードを改善することができます。可変長項目が参照されるたびに不要な変換が行われないようにするには、OCCURS . . . DEPENDING ON オブジェクトに対して BINARY を指定します。

- 可能であれば固定長データ項目を使用する。

可変長データ項目を使用する場合、それらの使用頻度が高くなる前に、固定長データ項目にコピーすると、オーバーヘッドを緩和することができます。

- 使用する探索メソッドのタイプに従ってテーブルを編成する。

テーブルが順次に探索される場合には、検索基準を満たす可能性が最も高いデータ値をテーブルの始まりに置くようにします。テーブルが二分探索アルゴリズムを使用して探索される場合は、検索キー・フィールドに基づいてアルファベット順にソートされたテーブルに、データ値を入れてください。

関連概念

『テーブル参照の最適化』

関連タスク

69 ページの『テーブル内の項目の参照』

590 ページの『効率的なデータ型の選択』

関連参照

309 ページの『SSRANGE』

テーブル参照の最適化

COBOL コンパイラーは、テーブル参照を、いくつかの方法で最適化します。

テーブル・エレメント参照 ELEMENT(S1 S2 S3) (S1、S2、および S3 は添え字) の場合、コンパイラーは次の式を評価します。

$$\text{comp_s1} * \text{d1} + \text{comp_s2} * \text{d2} + \text{comp_s3} * \text{d3} + \text{base_address}$$

ここで、comp_s1 は 2 進数に変換された後の S1 の値、comp-s2 は 2 進数に変換された後の S2 の値 (以下同様) です。それぞれの次元のストライドは d1、d2、および d3 です。ある特定次元のストライドは、その次元での出現番号が 1 だけ違い、かつ他の出現番号が等しいようなテーブル・エレメント相互間の距離 (バイト

単位) です。例えば、上記の例の 2 次元のストライド d2 は、ELEMENT(S1 1 S3) と ELEMENT(S1 2 S3) との間の距離 (バイト単位) です。

指標計算は添え字計算に類似していますが、指標値ではそれらの中にストライドを含めているので、乗算をする必要がないという点が異なります。指標計算には、指標をレジスターにロードすることも含まれます。これらのデータ転送は、個々の添え字計算の項を最適化する場合とほぼ同様に、最適化することができます。

コンパイラーは式を左から右へと評価していくので、定数または重複する添え字が左端にあると、最適化プログラムが計算を除去する可能性が最も高くなります。

定数項目と変数項目の最適化

C1、C2、. は、定数データ項目であり、V1、V2、. は変数データ項目です。したがって、テーブル・エレメント参照 ELEMENT(V1 C1 C2) の場合、コンパイラーは個々の項 $comp_c1 * d2$ および $comp_c2 * d3$ だけしか、式から定数として除去できません。

```
comp_v1 * d1 + comp_c1 * d2 + comp_c2 * d3 + base_address
```

しかし、テーブル・エレメント参照 ELEMENT(C1 C2 V1) の場合、コンパイラーは、副次式 $comp_c1 * d1 + comp_c2 * d2$ 全体を、式から定数として除去することができます。

```
comp_c1 * d1 + comp_c2 * d2 + comp_v1 * d3 + base_address
```

テーブル・エレメント参照 ELEMENT(C1 C2 C3) では、添え字はすべて定数なので、実行時に添え字計算は行われません。式は次のとおりです。

```
comp_c1 * d1 + comp_c2 * d2 + comp_c3 * d3 + base_address
```

最適化プログラムを使用すると、この参照は、スカラー (テーブルでない) 項目への参照と同じくらい効率的になります。

重複項目の最適化

テーブル・エレメント参照 ELEMENT(V1 V3 V4) および ELEMENT(V2 V3 V4) では、個々の項 $comp_v3 * d2$ および $comp_v4 * d3$ だけが、テーブル・エレメントの参照に必要な式における共通副次式です。

```
comp_v1 * d1 + comp_v3 * d2 + comp_v4 * d3 + base_address  
comp_v2 * d1 + comp_v3 * d2 + comp_v4 * d3 + base_address
```

しかし、2 つのテーブル・エレメント参照 ELEMENT(V1 V2 V3) および ELEMENT(V1 V2 V4) の場合は、副次式 $comp_v1 * d1 + comp_v2 * d2$ 全体が、テーブル・エレメントの参照に必要な 2 つの式間で共通となります。

```
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address  
comp_v1 * d1 + comp_v2 * d2 + comp_v4 * d3 + base_address
```

2 つの参照 ELEMENT(V1 V2 V3) および ELEMENT(V1 V2 V4) では、式は同じです。

```
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address  
comp_v1 * d1 + comp_v2 * d2 + comp_v3 * d3 + base_address
```

最適化プログラムを使用すると、同じエレメントへの 2 度目 (およびそれ以降) の参照は、スカラー (テーブルでない) 項目への参照と同じ効率になります。

可変長項目の最適化

従属 OCCURS DEPENDING ON データ項目の入っているグループ項目は可変長です。プログラムは、可変長データ項目が参照されるたびに特殊コードを実行しなければなりません。

このコードはライン外のもので、最適化の妨げになる可能性があります。さらに、可変長データ項目を処理するためのコードは、固定サイズ・データ項目を処理するコードよりかなり効率が下がり、処理時間も大幅に増加することがあります。例えば、可変長データ項目を比較したり移動したりするためのコードには、ライブラリー・ルーチンの呼び出しが必要なため、固定長データ項目の場合の同じコードよりかなり低速になります。

直接指標付けと間接指標付けの比較

相対指標参照は、直接指標参照と同じくらい迅速に実行されます。

ELEMENT (I5, J3, K2) の直接索引付けには、次のプリプロセスが必要になります。

```
SET I5 TO I
SET I5 UP BY 5
SET J3 TO J
SET J3 DOWN BY 3
SET K2 TO K
SET K2 UP BY 2
```

この処理のため、直接索引付けは、ELEMENT (I + 5, J - 3, K + 2) の相対索引付けよりも効率が悪くなります。

関連概念

596 ページの『最適化』

関連タスク

592 ページの『テーブルの効率的処理』

コードの最適化

プログラムの最終テストの準備ができたなら、OPTIMIZE コンパイラー・オプションを指定して、テスト・コードと実動コードが同一になるようにしてください。

再コンパイルされずにプログラムが頻繁に使用される場合、開発時にこのコンパイラー・オプションを使用することもできます。しかし、アセンブラー言語の拡張部分 (LIST コンパイラー・オプション) を使用してプログラムの微調整を行う場合を除き、再コンパイルを頻繁に行うと、OPTIMIZE のオーバーヘッドが利点を上回ることがあります。

プログラムのユニット・テストについては、最適化されていないコードをデバッグする方が簡単です。

最適化プログラムがプログラム上でどのように機能するかを見るためには、OPTIMIZE オプションを指定した場合としない場合でのコンパイルを行い、そのうえで生成されたコードを比較します。(生成されたコードのアセンブラー・リストを要求するには、LIST コンパイラー・オプションを使用します。)

関連概念

『最適化』

関連参照

291 ページの『LIST』

298 ページの『OPTIMIZE』

最適化

生成されたコードの効率を上げるため、OPTIMIZE コンパイラー・オプションを使用することができます。

OPTIMIZE を使用すると、COBOL 最適化プログラムが以下の最適化を行います。

- 不必要な制御権移動および非効率的な分岐を除去します。ソース・プログラムを見るだけではわからない、コンパイラーが生成する分岐も含まれます。
- 含まれている (ネストされた) プログラムに対する CALL ステートメントのコンパイル済みコードを単純化します。可能であれば、最適化プログラムはステートメントをインラインに設定し、リンケージ・コードがなくて済むようにします。この最適化は、**プロシージャー統合**と呼ばれます。プロシージャー統合を行えない場合、最適化プログラムは、できるだけ単純なリンケージ (2 つ程度の命令) を使用して、呼び出し先プログラムとの間を往復します。
- プログラムの結果に何の影響も与えない重複計算 (添え字計算や繰り返しのステートメントなど) を除去します。
- プログラムのコンパイル時に定数計算を実行することによって、定数計算を除去します。
- 定数条件式を除去します。
- 連続した項目 (MOVE CORRESPONDING の使用によって頻繁に発生するような) の移動を集約して、単一の移動にします。移動を集約するには、移動元も移動先も連続していなければなりません。
- 参照されないデータ項目を DATA DIVISION から廃棄し、これらのデータ項目をその VALUE 節に初期化するコードの生成を抑止します。(最適化プログラムがこのアクションを取るのは、FULL サブオプションが指定された場合だけです。)

含まれているプログラム・プロシージャーの統合

含まれているプログラム・プロシージャーの統合では、含まれているプログラム・コードが、含まれているプログラムへの CALL と置き換わります。結果として生じるプログラムは、CALL リンケージのオーバーヘッドもなく、より線形の制御フローとなり、より速く実行されます。

プログラム・サイズ: 含まれているプログラムを複数の CALL ステートメントが呼び出している場合、それぞれのプログラムがこのような各ステートメントと置き換わるのであれば、収容プログラムが相当大きくなる可能性があります。最適化プログラムはこの増加を 50 % 以内に制限し、それ以降はプログラムを統合しません。これにより最適化プログラムはその CALL ステートメントに、次善の最適化を選択します。リンケージのオーバーヘッドをわずか 2 つの命令にすることができます。

到達不能コード: この統合の結果として、含まれている 1 つのプログラムが何回も繰り返されることがあります。その後の最適化はプログラムの各コピーで進められるため、コードのコピー先のコンテキストによっては、到達不能な部分が検出されることがあります。

関連概念

593 ページの『テーブル参照の最適化』

関連参照

298 ページの『OPTIMIZE』

パフォーマンスを向上させるコンパイラー機能の選択

どんなパフォーマンス関連コンパイラー・オプションを選択するか、また USE FOR DEBUGGING ON ALL PROCEDURES ステートメントを使用するかどうかは、プログラムがどの程度うまく最適化されるかに影響を与えます。

カスタマイズ済みシステムには、最適なパフォーマンスを得るために特定のオプションが必要なものもあります。以下の手順を実行します。

1. システム・デフォルトの内容を調べるには、プログラムの短縮リストを入手し、リストされたオプション設定値を検討する。
2. プログラムをコンパイルするためのパフォーマンス関連オプションを選択する。

重要: COBOL プログラムの調整方法については、システム・プログラマーと相談してください。そうすれば、選択したオプションがインストール先のプログラムに適したものであるかどうかを確認できます。

考慮する必要があるもう 1 つのコンパイラー機能として、USE FOR DEBUGGING ON ALL PROCEDURES ステートメントがあります。これは、コンパイラーの最適化プログラムに大きな影響を与える可能性があります。ON ALL PROCEDURES オプションは、プロシージャ名に移動するたびに、余分のコードを生成します。これはデバッグには大変便利ですが、プログラムは非常に大型になり、実質上最適化を抑制する可能性があります。

関連概念

596 ページの『最適化』

関連タスク

595 ページの『コードの最適化』

342 ページの『リストの入手』

関連参照

『パフォーマンスに関連するコンパイラー・オプション』

パフォーマンスに関連するコンパイラー・オプション

以下の表には、それぞれのオプションの目的、パフォーマンス上の利点と欠点、および使用上の注意 (該当する場合) が示されています。

表 69. パフォーマンスに関連するコンパイラー・オプション

コンパイラー・オプション	目的	パフォーマンス上の長所	パフォーマンス上の欠点	使用上の注意
ARITH(EXTEND) (268 ページの『ARITH』を参照)	10 進数で許可される最大桁数を増やす	一般には、ありません。	ARITH(EXTEND) を使用すると、中間結果が大きくなるため、すべての 10 進数データ型でパフォーマンスがいくらか低下します。	どれほど低下するかは、使用する 10 進数データの量に直接左右されます。
280 ページの『DYNAM』	サブプログラム (CALL ステートメントによって呼び出された) が実行時に動的にロードされるようにする	サブプログラムが変更されても、アプリケーションをリンク・エディットする必要がないので、サブプログラムの保守が容易になります。	呼び出しはライブラリー・ルーチンを介して行う必要があるため、ちょっとしたパフォーマンス・ペナルティーがあります。	不要になった仮想記憶域を解放するには、CANCEL ステートメントを出してください。
OPTIMIZE(STD) (298 ページの『OPTIMIZE』を参照)	パフォーマンスがよくなるように、生成されるコードを最適化する	一般に、もっと効率的な実行時コードが得られます。	コンパイル時間が長くなること。OPTIMIZE は、NOOPTIMIZE に比べ、コンパイルにかかる処理時間が長くなります。	NOOPTIMIZE は通常、頻繁なコンパイルが必要となるプログラム開発の段階で使用されます。これにより、シンボリック・デバッグも可能になります。実稼働用には、OPTIMIZE の使用をお勧めします。
OPTIMIZE(FULL) (298 ページの『OPTIMIZE』を参照)	生成されたコードをパフォーマンスがよくなるように最適化し、さらに DATA DIVISION も最適化する	一般に、もっと効率的な実行時コードが得られ、ストレージ使用量が少なくなります。	コンパイル時間が長くなること。OPTIMIZE は、NOOPTIMIZE に比べ、コンパイルにかかる処理時間が長くなります。	OPT(FULL) は未使用データ項目を削除しますが、それは、ダンプ読み取り用マーカーとしてのみ使用されるタイム・スタンプまたはデータ項目の場合には、望ましくないことがあります。
NOSSRANGE (309 ページの『SSRANGE』を参照)	すべてのテーブル参照および参照変更式が適切な範囲内にあるかどうかを検査する	SSRANGE は、テーブル参照を検査するための追加コードを生成しません。NOSSRANGE を使用すると、コードは生成されません。	なし	一般に、テーブル参照のたびに検査する必要はなく、数度の検査だけで済む場合には、独自の検査をコーディングする方が、SSRANGE を使用するより速くなります。CHECK(OFF) ランタイム・オプションを使用して、実行時に SSRANGE をオフにすることができます。パフォーマンスを重視するアプリケーションについては、NOSSRANGE の使用をお勧めします。

表 69. パフォーマンスに関連するコンパイラー・オプション (続き)

コンパイラー・オプション	目的	パフォーマンス上の長所	パフォーマンス上の欠点	使用上の注意
NOTEST (310 ページの『TEST』を参照)	Debugger for AIX をフルに活用するために必要な追加のオブジェクト・コードを回避する	TEST はデバッグ情報を追加するため、オブジェクト・ファイルが大幅に拡張します。プログラムをリンクするとき、デバッグ情報を除外するようリンカーに指示することができます。そうすると、実行可能モジュールのサイズは、モジュールが NOTEST でコンパイルされた場合に作成されるサイズとほぼ同じになります。デバッグ情報が組み込まれると、実行可能モジュールが大きくなるほどロード時間が長くなり、ページングが増える可能性があるため、パフォーマンスが多少低下することがあります。	なし	TEST は、コンパイラー・オプション NOOPT を強制的に有効にします。実稼働の場合は、NOTEST を使用することをお勧めします。
TRUNC(OPT) (312 ページの『TRUNC』を参照)	算術演算の受信フィールドを切り捨てるためのコードの生成を回避する	余分のコードを生成しないので、一般にパフォーマンスは向上します。	TRUNC(BIN) と TRUNC(STD) はともに、BINARY データ項目が変更されるたびに、余分のコードを生成します。TRUNC(BIN) は、そのパフォーマンスについては COBOL for OS/390 & VM で改善されたとはいえ、上記のオプション中では最も低速のオプションです。	TRUNC(STD) は標準 COBOL 85 に準拠しますが、TRUNC(BIN) および TRUNC(OPT) は準拠しません。TRUNC(OPT) を使用すると、コンパイラーは、データが PICTURE および USAGE の仕様に従っていると見なします。可能な場合は、TRUNC(OPT) を使用することをお勧めします。

関連概念

596 ページの『最適化』

関連タスク

243 ページの『コンパイル・エラー・メッセージのリストの生成』

597 ページの『パフォーマンスを向上させるコンパイラー機能の選択』

592 ページの『テーブルの効率的処理』

関連参照

53 ページの『ゾーンおよびバック 10 進数データのサイン表記』
263 ページの『第 14 章 コンパイラー・オプション』

パフォーマンスの評価

プログラムのパフォーマンスの評価に役立つ次のワークシートに記入してください。各質問に「はい」と答える場合は、おそらくパフォーマンスは向上しています。

パフォーマンスのトレードオフを比較検討する際には、各オプションの機能およびパフォーマンスの利点と欠点を十分に理解するようにしてください。パフォーマンスの向上よりも、機能を重視する場合も多くあります。

表 70. パフォーマンス調整のワークシート

コンパイラー・オプション	考慮事項	はい
DYNAM	NODYNAM を使用できますか。パフォーマンスのトレードオフを比較検討してください。	
OPTIMIZE	実稼働に OPTIMIZE を使用していますか。OPTIMIZE(FULL) を使用できますか。	
SSRANGE	NOSSRANGE を実稼働に使用していますか。	
TEST	実稼働に NOTEST を使用しますか。	
TRUNC	可能な場合、TRUNC(OPT) を使用していますか。	

関連タスク

597 ページの『パフォーマンスを向上させるコンパイラー機能の選択』

関連参照

597 ページの『パフォーマンスに関連するコンパイラー・オプション』

第 33 章 コーディングの単純化

コーディング技法を使用して、生産性を向上させることができます。COPY ステートメント、COBOL 組み込み関数、および呼び出し可能サービスを使用することによって、反復コーディングや、多数の算術計算または他の複雑なタスクをコーディングする必要性を回避することができます。

プログラムに頻繁に使用されるコード・シーケンス (共通データ項目のブロック、入出力ルーチン、エラー・ルーチン、または COBOL プログラム全体) が含まれている場合は、それらのコード・シーケンスを 1 度作成し、それらを COBOL コピー・ライブラリーに入れてください。COPY ステートメントを使用してこれらのコード・シーケンスを取り出し、コンパイル時にプログラムに含めることができます。このようにコピーブックを使用することによって、コーディングの繰り返しがなくなります。

COBOL は、ストリングおよび数値を扱うためのさまざまな機能を提供します。これらの機能がコーディングの単純化に役立ちます。

日時呼び出し可能サービスは、日付をフルワード・バイナリー整数として保管し、タイム・スタンプを長精度 (64 ビット) 浮動小数点値として保管します。これらの形式を使用することにより、算術計算を日時の値に基づいて単純かつ効率的に行うことができます。このような計算を実行するために、言語ライブラリーの外側でサービスを使用する特別なサブルーチンを書く必要はありません。

関連タスク

56 ページの『数字組み込み関数の使用』

『反復コーディングの除去』

111 ページの『データ項目の変換 (組み込み関数)』

114 ページの『データ項目の評価 (組み込み関数)』

603 ページの『日時の取り扱い』

反復コーディングの除去

COPY ステートメントをプログラムの任意の部、また任意のコード・シーケンス・レベルで使用することで、保管されているソース・ステートメントをプログラムに組み込むことができます。COPY ステートメントは任意の深さにネストできます。

複数のコピー・ライブラリーを指定するには、環境変数 SYSLIB を、セミコロン (;) で区切った複数のパス名に設定するか、自身の環境変数を定義して、COPY ステートメントに次の句を組み込みます。

```
IN/OF library-name
```

以下に例を示します。

```
COPY MEMBER1 OF COPYLIB
```

この修飾句を省略した場合、デフォルトは SYSLIB です。

次の例のようなコマンドを使用して、コンパイル時に COPYLIB を定義する環境変数を設定します。

```
SET COPYLIB=D:¥CPYFILES¥COBCOPY
```

COPY とデバッグ行: コピーされたテキストをデバッグ行として (例えば、7 桁目に D が挿入されているかのように) 扱わせるには、COPY ステートメントの最初の行に D を入れてください。COPY ステートメント自体をデバッグ行にすることはできません。これに D が入っていても、WITH DEBUGGING モードが指定されていないければ、COPY ステートメントが処理されることはありません。

『例: COPY ステートメントの使用』

関連参照

319 ページの『第 15 章 コンパイラー指示ステートメント』

例: COPY ステートメントの使用

これらの例は、COPY ステートメントを使用してライブラリー・テキストをプログラムに組み込む方法を示しています。

ライブラリー項目 CFILEA が以下の FD 項目から構成されているものとします。

```
BLOCK CONTAINS 20 RECORDS
RECORD CONTAINS 120 CHARACTERS
LABEL RECORDS ARE STANDARD
DATA RECORD IS FILE-OUT.
01 FILE-OUT      PIC X(120).
```

次のようにソース・プログラムで COPY ステートメントを使用すれば、テキスト名 CFILEA を取得することができます。

```
FD FILEA
   COPY CFILEA.
```

このライブラリー記入項目はプログラムにコピーされ、その結果生じるプログラム・リストは次のようになります。

```
FD FILEA
   COPY CFILEA.
C   BLOCK CONTAINS 20 RECORDS
C   RECORD CONTAINS 120 CHARACTERS
C   LABEL RECORDS ARE STANDARD
C   DATA RECORD IS FILE-OUT.
C   01 FILE-OUT      PIC X(120).
```

コンパイラー・ソース・リストで COPY ステートメントは別個の行に印刷され、コピーされた行の前には C が付けられます。

テキスト名 DOWORK を持つコピーブックが、以下のステートメントによって保管されているとします。

```
COMPUTE QTY-ON-HAND = TOTAL-USED-NUMBER-ON-HAND
MOVE QTY-ON-HAND to PRINT-AREA
```

DOWORK として識別されたコピーブックを取り出すには、次のようにコーディングします。

```
paragraph-name.
  COPY DOWORK.
```

DOWORK プロシージャー内のステートメントは *paragraph-name* の後に置かれます。

EXIT コンパイラー・オプションを使用して LIBEXIT モジュールを指定すると、結果がこの章で示されるものと異なることがあります。

関連タスク

601 ページの『反復コーディングの除去』

関連参照

319 ページの『第 15 章 コンパイラー指示ステートメント』

日時の取り扱い

日付または時刻の呼び出し可能サービスを呼び出すには、このサービス用の正しいパラメーターを指定した CALL ステートメントを使用してください。DATA DIVISION 内の CALL ステートメントにそのサービスに必要なデータ定義を使用してデータ項目を定義します。

```
77 argument          pic s9(9) comp.
01 format.
   05 format-length  pic s9(4) comp.
   05 format-string  pic x(80).
77 result            pic x(80).
77 feedback-code    pic x(12) display.
. . .
CALL "CEEDATE" using argument, format, result, feedback-code.
```

上の例では、呼び出し可能サービス CEEDATE によって、データ項目 *argument* にリリアン日付で表された数値が、データ項目 *result* に書き込まれる文字形式の日付に変換されます。データ項目 *format* に含まれるピクチャー・ストリングは、変換形式を制御します。呼び出しの成功/失敗に関する情報は、データ項目 *feedback-code* に戻されます。

日時の呼び出し可能サービスを呼び出すための CALL ステートメントでは、ID ではなくプログラム名のリテラルを使用する必要があります。

プログラムは、標準のシステム・リンケージ規約を使用して、日時の呼び出し可能サービスを呼び出します。

604 ページの『例: 日付の操作』

関連概念

649 ページの『付録 F. 日時呼び出し可能サービス』

関連タスク

604 ページの『日時呼び出し可能サービスからのフィードバックの取得』

604 ページの『日時の呼び出し可能サービスからの条件の処理』

関連参照

606 ページの『フィードバック・トークン』

607 ページの『ピクチャー文字項およびストリング』

CALL ステートメント (*COBOL for AIX* 言語解説書)

日時呼び出し可能サービスからのフィードバックの取得

日時の呼び出し可能サービスでは、フィードバック・コード・パラメーター (オプション) を指定することができます。このサービスが呼び出しの成功/失敗に関する情報を返さないようにするには、このパラメーターに対して OMITTED を指定します。

ただし、このパラメーターを指定せずに、呼び出し可能サービスが失敗した場合は、プログラムが異常終了します。

日時の呼び出し可能サービスの呼び出し時に、フィードバック・コードに OMITTED を指定すると、サービスが成功した場合は RETURN-CODE 特殊レジスターが 0 に設定されますが、サービスが失敗した場合は特殊レジスターが変更されません。フィードバック・コードが OMITTED でない場合は、サービスが成功したかどうかに関係なく、RETURN-CODE 特殊レジスターは常に 0 に設定されます。

605 ページの『例: 出力用の日付形式』

関連参照

606 ページの『フィードバック・トークン』

日時の呼び出し可能サービスからの条件の処理

COBOL for AIX による条件処理は、ホスト上の IBM Language Environment[®] による条件処理とは大幅に異なります。COBOL for AIX は、ネイティブの COBOL 条件処理スキームに従っており、言語環境プログラムにあるサポートのレベルには対応していません。

フィードバック・トークンを引数として渡す場合は、適切な情報が埋め込まれると単に返されます。呼び出しルーチンにロジックをコーディングして、内容を検証し、必要に応じてアクションを実行することができます。条件はシグナル通知されません。

関連参照

606 ページの『フィードバック・トークン』

例: 日付の操作

次の例では、日付および時刻 の呼び出し可能サービスを使用して日付を別の形式に変換し、このフォーマットされた日付で単純な計算を行う方法を示します。

```
CALL CEEDAYS USING dateof_hire, 'YYMMDD', doh_lilian, fc.  
CALL CEEOCT USING todayLilian, today_seconds, today_Gregorian, fc.  
COMPUTE servicedays = today_Lilian - doh_Lilian.  
COMPUTE serviceyears = service_days / 365.25.
```

上の例では、YYMMDD 形式の雇用日を元にして、従業員の就労年数を計算します。計算は次のようになります。

1. CEEDAYS (日付をリリアン形式へ変換) を呼び出して、日付をリリアン形式に変換します。
2. CEEOCT (現地時間の取得) を呼び出して、現地時間を取得します。

3. today_Lilian から doh_Lilian を減算して (グレゴリオ暦の開始から現地時間までの日数)、従業員の雇用日数を計算します。
4. この日数を 365.25 で除算して、就労年数を求めます。

例: 出力用の日付形式

次のサンプルは、日付および時刻の呼び出し可能サービスを使用して、ACCEPT ステートメントから取得された日付をフォーマットして表示します。

多くの呼び出し可能サービスには、旧バージョンの COBOL を使用した場合に大量のコーディングを必要としていた機能が備わっています。それが CEEDAYS および CEEDATE というサービスです。日付をフォーマットする際には、これらを効率的に利用することができます。

```

CBL QUOTE
  ID DIVISION.
  PROGRAM-ID. HOHOHO.
  *****
  * FUNCTION:  DISPLAY TODAY'S DATE IN THE FOLLOWING FORMAT: *
  *           WWWWWWWW, MMMMMMM DD, YYYY                    *
  *           *
  *           For example:  MONDAY, OCTOBER 20, 2008          *
  *           *
  *****
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  CHRDATE.
      05  CHRDATE-LENGTH    PIC S9(4) COMP VALUE 10.
      05  CHRDATE-STRING   PIC X(10).
  01  PICSTR.
      05  PICSTR-LENGTH    PIC S9(4) COMP.
      05  PICSTR-STRING   PIC X(80).

  77  LILIAN PIC           S9(9) COMP.
  77  FORMATTED-DATE     PIC X(80).

  PROCEDURE DIVISION.
  *****
  *   USE DATE/TIME CALLABLE SERVICES TO PRINT OUT          *
  *   TODAY'S DATE FROM COBOL ACCEPT STATEMENT.            *
  *****
  ACCEPT CHRDATE-STRING FROM DATE.

  MOVE "YYMMDD" TO PICSTR-STRING.
  MOVE 6 TO PICSTR-LENGTH.
  CALL "CEEDAYS" USING CHRDATE , PICSTR , LILIAN , OMITTED.

  MOVE " WWWWWWWWZ, MMMMMMMMZ DD, YYYY " TO PICSTR-STRING.
  MOVE 50 TO PICSTR-LENGTH.
  CALL "CEEDATE" USING LILIAN , PICSTR , FORMATTED-DATE ,
    OMITTED.

  DISPLAY "*****".
  DISPLAY FORMATTED-DATE.
  DISPLAY "*****".

  STOP RUN.

```

フィードバック・トークン

フィードバック・トークンには、フィードバック情報が条件トークンの形式で含まれています。呼び出し可能サービスによって設定される条件トークンは呼び出しルーチンに戻され、サービスが正常に完了したかどうかを示します。

COBOL for AIX では、言語環境プログラムと同じフィードバック・トークンを使用します。これは次のように定義されます。

```
01 FC.
  02 Condition-Token-Value.
  COPY CEEIGZCT.
    03 Case-1-Condition-ID.
      04 Severity PIC S9(4) COMP.
      04 Msg-No PIC S9(4) COMP.
    03 Case-2-Condition-ID.
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) COMP.
      04 Cause-Code PIC S9(4) COMP.
    03 Case-Sev-Ct1 PIC X.
    03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) COMP.
```

各フィールドの内容とホスト上の IBM 言語環境プログラムとの違いは、以下のとおりです。

Severity

重大度数を表します。次の値を持ちます。

- 0 情報のみ (あるいは、トークン全体がゼロの場合は情報なし)。
- 1 警告: サービスは、ほぼ正常に完了しました。
- 2 エラーの検出: 修正が試みられましたが、サービスはおそらく正常には完了しませんでした。
- 3 重大エラー: サービスは完了しませんでした。
- 4 クリティカル・エラー: サービスは完了しませんでした。

Msg-No 関連するメッセージ番号です。

Case-Sev-Ct1

このフィールドには、常に値 1 が入ります。

Facility-ID

このフィールドには、常に文字 CEE が入ります。

I-S-Info

このフィールドには、常に値 0 が入ります。

サンプルのコピーブック CEEIGZCT.CPY は、条件トークンを定義します。ファイル内の条件トークンは、言語環境プログラムに備わっている条件トークンと同じです。ただし、文字表現は EBCDIC ではなく ASCII です。条件トークンを言語環境プログラムに備わっているものと比較する場合は、違いを考慮する必要があります。

各呼び出し可能サービスの記述には、シンボリック・フィードバック・コードのリストが含まれます。これらのコードは、サービスの呼び出し時に指定されたフィードバック・コード出力フィールドに戻される場合があります。このほかに、任意の

呼び出し可能サービスに対してシンボリック・フィールドバック・コード CEE0PD が戻される場合があります。詳細については、メッセージ IWZ0813S を参照してください。

日付および時刻呼び出し可能サービスはすべて、グレゴリオ暦に基づいています。グレゴリオ暦に関連する日付変数には、アーキテクチャー上の制限があります。これらの制限を次に示します。

リリアン日付の開始

リリアン日付範囲の 1 日目は、グレゴリオ暦の 1582 年 10 月 15 日 (金曜日) と同じです。この日付よりも前のリリアン日付は定義されていません。したがって、次のようになります。

- 0 日目 = 1582 年 10 月 14 日 00:00:00
- 1 日目 = 1582 年 10 月 15 日 00:00:00

日付の入力として有効となるのは、1582 年 10 月 15 日の 00:00:00 以降です。

リリアン日付の終了

リリアン日付の終了日は、9999 年 12 月 31 日に設定されます。この日付よりも後のリリアン日付は、9999 年が 4 桁で可能な最大の年であるため定義されていません。

関連参照

781 ページの『付録 J. ランタイム・メッセージ』

ピクチャー文字項およびストリング

いくつかの日付および時刻の呼び出し可能サービスには、ピクチャー文字項 (入力データの形式または出力データの必要な形式を示すテンプレート) を使用します。

表 71. ピクチャー文字項およびストリング

ピクチャー項	説明	有効な値	注
Y YY YYY ZYY YYYY	1 桁の年号 2 桁の年号 3 桁の年号 特定元号での 3 桁の年号 4 桁の年号	0 から 9 00 から 99 000 から 999 1 から 999 1582 から 9999	Y は出力に対してのみ有効です。 YY は CEEScen によって設定された範囲を前提とします。 YYY/ZYY は、<JJJJ>、<CCCC>、および <CCCCCCCC> とともに使用されます。
<JJJJ>	UTF-16 の 16 進数エンコード方式を使用した漢字の日本元号	平成 (NX'5E736210') 昭和 (NX'662D548C') 大正 (NX'59276B63') 明治 (NX'660E6CBB')	YY フィールドに影響します。<JJJJ> が指定されている場合、YY は特定の日本元号の年号を意味します。例えば、1988 年は昭和 63 年に相当します。
MM ZM	2 桁の月数 1 または 2 桁の月数	01 から 12 1 から 12	出力の場合は、先行ゼロが抑制されません。入力の場合は、ZM も MM と同じように扱われます。
RRRR RRRZ	ローマ数字の月数	Ibbb-XIIb (左揃え)	入力の場合は、ソース・ストリングが大文字変換されます。出力の場合は、大文字のみで行われます。 I=Jan, II=Feb, ..., XII=Dec

表 71. ピクチャー文字項およびストリング (続き)

ピクチャー項	説明	有効な値	注
MMM Mmm MMMM...M Mmmm...m MMMMMMMMZ Mmmmmmmz	3 文字の月名、大文字 3 文字の月名、大/小文字混合 3 から 20 文字の月名、大文字 3 から 20 文字の月名、大/小文字混合 末尾ブランクは抑制されます。 末尾ブランクは抑制されます。	JAN から DEC Jan から Dec JANUARY bb -DECEMBER b January bb -December b JANUARY から DECEMBER January から December	入力の場合は、ソース・ストリングが必ず大文字変換されます。出力の場合は、M が大文字、m が小文字を生成します。出力には、ブランク (b) が埋め込まれるか (Z が指定されていない場合)、M の数 (最大 20) まで切り詰められます。
DD ZD DDD	2 桁の日付 1 または 2 桁の日付 年間通算日 (ユリウス日付)	01 から 31 1 から 31 001 から 366	出力の場合は、先行ゼロが常に抑制されます。入力の場合は、ZD も DD と同じように扱われます。
HH ZH	2 桁の時間数 1 または 2 桁の時間数	00 から 23 0 から 23	出力の場合は、先行ゼロが抑制されません。入力の場合は、ZH も HH と同じように扱われます。AP が指定されている場合の有効値は 01 から 12 です。
MI	分数	00 から 59	
SS	2 番目	00 から 59	
9 99 999	10 分の 1 の秒数 100 分の 1 の秒数 1000 分の 1 の秒数	0 から 9 00 から 99 000 から 999	丸めなし
AP ap A.P. a.p.	AM/PM 標識	AM または PM am または pm A.M. または P.M. a.m. または p.m.	AP は HH/ZH フィールドに影響しません。入力の場合は、ソース・ストリングが必ず大文字変換されます。出力の場合は、AP が大文字、ap が小文字を生成します。
W WWW Www WWW...W Www...w WWWWWWWZ Wwwwwwwz	1 文字の曜日名 3 文字の曜日名、大文字 3 文字の曜日名、大/小文字混合 3 から 20 文字の曜日名、大文字 3 から 20 文字の曜日名、大/小文字混合 末尾ブランクは抑制されます。 末尾ブランクは抑制されます。	S, M, T, W, T, F, S SUN から SAT Sun から Sat SUNDAY bbb -SATURDAY b Sunday bbb -Saturday b SUNDAY から SATURDAY Sunday から Saturday	入力の場合は、W が無視されます。出力の場合は、W が大文字、w が小文字を生成します。出力には、ブランクが埋め込まれるか (Z が指定されていない場合)、W の数 (最大 20) まで切り詰められます。
それ以外	区切り文字	X'01'から X'FF' (X'00' は、日付および時刻の呼び出し可能サービスが「内部」使用するために予約済み)	入力の場合は、月、日、年、時間、分、秒、秒の小数部の間の区切り文字として扱われます。出力の場合は、現状のままターゲット・ストリングにコピーされます。
注: ブランク文字は、シンボル b で示されます。			

次の表に、<JJJJ> が指定されている場合に日時サービスによって使用される日本元号の定義を示します。

表 72. 日本元号

各日本元号の 1 日目	年号名	UTF-16 の 16 進数エ ンコード方式を使用 した漢字の元号名	有効な年数値
1868-09-08	明治	NX'660E6CBB'	01 から 45
1912-07-30	大正	NX'59276B63'	01 から 15
1926-12-25	昭和	NX'662D548C'	01 から 64
1989-01-08	平成	NX'5E736210'	01 から 999 (01 = 1989)

『例: 日時のピクチャー・ストリング』

例: 日時のピクチャー・ストリング

日時サービスによって認識されるピクチャー・ストリングの例を次に示します。

表 73. 日時のピクチャー・ストリングの例

ピクチャー・ストリング	例	コメント
YYMMDD YYYYMMDD	880516 19880516	
YYYY-MM-DD	1988-05-16	1988-5-16 も有効な入力になります。
<JJJJ> YY.MM.DD	昭和 63.05.16	昭和 は日本元号名です。昭和 63 年は 1988 年に相当します。
MMDDYY MM/DD/YY ZM/ZD/YY MM/DD/YYYY MM/DD/Y	050688 05/06/88 5/6/88 05/06/1988 05/06/8	1 桁の年号形式 (Y) は、出力に対してのみ有効です。
DD.MM.YY DD-RRRR-YY DD MMM YY DD Mmmmmmmmm YY ZD Mmmmmmmmmz YY Mmmmmmmmmz ZD, YYYY ZDMMMMMMzYY	09.06.88 09-VI -88 09 JUN 88 09 June 88 9 June 88 June 9, 1988 9JUNE88	Z は、ゼロおよびブランクを抑制します。
YY.DDD YYDDD YYYY/DDD	88.137 88137 1988/137	ユリウス日付
YYMMDDHHMISS YYYYMMDDHHMISS YYYY-MM-DD HH:MI:SS.999 WWW, ZM/ZD/YY HH:MI AP Wwwwwwwwwz, DD Mmm YYYY, ZH:MI AP	880516204229 19880516204229 1988-05-16 20:42:29.046 MON, 5/16/88 08:42 PM Monday, 16 May 1988, 8:42 PM	タイム・スタンプは、CEESECS および CEEDATM に対してのみ有効です。CEEDATE とともに使用した場合は、時刻の位置にゼロが埋め込まれます。CEEDAYS とともに使用した場合は、HH、MI、SS、999 の各フィールドが無視されます。
注: 小文字を使用できるのは、英字のピクチャー項のみです。		

世紀ウィンドウ

2000 年以降の 2 桁年号を処理するために、日付および時刻の呼び出し可能サービスではスライド方式を使用しています。この方式では、2 桁年号はすべて、現在のシステム日付よりも 80 年前から始まる 100 年間隔 (世紀ウィンドウ) に属するものと想定されます。

例えば 2008 年では、1928 から 2027 の 100 年間は、日付および時刻の呼び出し可能サービスに使用されるデフォルトの世紀ウィンドウとなります。2008 年の場合、28 から 99 が 1928 年から 1999 年、00 から 27 が 2000 年から 2027 年と認識されます。



2080 年までは、2 桁年号はすべて 20 mn 年と認識されます。2081 年では、00 は 2100 年と認識されます。

アプリケーションによっては、別の 100 年間隔を設定する必要があります。例えば、銀行で取り扱っている 30 年債の期限が 01/31/28 に設定されているとします。2 桁の年 28 は、上で説明した世紀ウィンドウが有効である場合は、1928 として解釈されます。CEEQCEN 呼び出し可能サービスを使用すると、世紀ウィンドウを変更できます。もう一方のサービス CEEQCEN は、現行の世紀ウィンドウを照会します。

CEEQCEN および CEESCEN を使用すると、例えばサブルーチンに、親ルーチンとは異なる日付処理間隔を使用させることができます。サブルーチンは、戻る前にこの間隔を前の値にリセットする必要があります。

『例: 世紀ウィンドウの照会および変更』

例: 世紀ウィンドウの照会および変更

CEEQCEN および CEESCEN サービスを使用して、世紀ウィンドウの開始点を照会、設定、および復元する方法の例を次に示します。

この例では、CEEQCEN を呼び出して整数 (OLDCEN) を取得し、現行の世紀ウィンドウが何年前に始まったかを示します。次に、新しい値 (TEMPCEN) で CEESCEN を呼び出して、現行の世紀ウィンドウの開始点を新しい値に一時的に変更します。この世紀ウィンドウは 30 に設定されているため、CEESCEN 呼び出しに続く 2 桁年号は、現行のシステム日付より 30 年前から始まる 100 年間隔に属するものと想定されます。

最後に、一時的な世紀ウィンドウを使用して日付 (例示なし) を処理した後、再度 CEESCEN を呼び出して、世紀ウィンドウの開始点を元の値にリセットします。

```
WORKING-STORAGE SECTION.  
77 OLDCEN PIC S9(9) COMP.  
77 TEMPCEN PIC S9(9) COMP.  
77 QCENFC PIC X(12).  
.  
77 SCENFC1 PIC X(12).  
77 SCENFC2 PIC X(12).
```

```

. . .
PROCEDURE DIVISION.
. . .
** Call CEEQCEN to retrieve and save current century window
   CALL "CEEQCEN" USING OLDCEN, QCENFC.
** Call CEESCEN to temporarily change century window to 30
   MOVE 30 TO TEMPCEN.
   CALL "CEESCEN" USING TEMPCEN, SCENFC1.
** Perform date processing with two-digit years
. . .
** Call CEESCEN again to reset century window
   CALL "CEESCEN" USING OLDCEN, SCENFC2.
. . .
GOBACK.

```

関連参照

649 ページの『付録 F. 日時呼び出し可能サービス』

第 9 部 付録

付録 A. 移行および互換性

ここでは、COBOL for AIX バージョン 2 のプログラムから COBOL for AIX バージョン 3 に移行する場合に考慮すべきであると考えられる領域についていくつか説明します。

関連参照

『オブジェクト・コード』

『オブジェクト指向構文、および Java 5 または Java 6 SDK』

616 ページの『データ・サイズ制限』

616 ページの『XML 生成』

616 ページの『SFS ファイル』

617 ページの『32 ビット・アプリケーションの 64 ビット・モードへの移行』

予約語 (COBOL for AIX 言語解説書)

オブジェクト・コード

COBOL for AIX プログラムは、COBOL for AIX バージョン 2 でコンパイルされたプログラムを呼び出したり、COBOL for AIX バージョン 2 でコンパイルされたプログラムから呼び出されたりすることができます。

COBOL for AIX バージョン 2 でコンパイルされたプログラムは、最新の COBOL for AIX ランタイム・ライブラリーを使用して実行できます。

オブジェクト指向構文、および Java 5 または Java 6 SDK

Java インターオペラビリティのオブジェクト指向構文を使用する COBOL for AIX バージョン 2 アプリケーションは、Java SDK 1.4.2 とともに使用する場合にサポートされます。

このような既存のアプリケーションを Java 5 または Java 6 で実行するには、以下のステップを実行します。

1. 最新バージョンの COBOL for AIX を使用してアプリケーションを再コンパイルして再リンクします。
2. Java 5 または Java 6 からの javac コマンドを使用して、それぞれのオブジェクト指向 COBOL クラスに関連付けられている生成済みの Java クラスを再コンパイルします。

関連概念

261 ページの『Java SDK サポート』

関連タスク

258 ページの『オブジェクト指向アプリケーションの準備』

データ・サイズ制限

オプション ADDR(32) を有効にして COBOL for AIX プログラムがコンパイルされている場合、データ項目の最大サイズ制限は、COBOL for AIX バージョン 2 でコンパイルされたプログラムのデータ項目のサイズより小さくなることがあります。

コンパイラー限界値について詳しくは、コンパイラー限界値に関する関連参照を参照してください。

関連参照

266 ページの『ADDR』

コンパイラー限界値 (COBOL for AIX 言語解説書)

XML 生成

XML コンテンツで不正となる文字が、エレメント文字コンテンツに含まれている場合、XML GENERATE によって例外コード 417 が返されます。接頭部に「hex」が付いたエレメント・タグ名と、文書内で 16 進表記されている元のデータ値を使用して、XML 生成が続行されます。

生成された XML 文書には、COBOL for AIX バージョン 2 で生成された文書との互換性がありますが、新規の例外コードが発生します。

関連タスク

412 ページの『XML GENERATE 例外の処理』

関連参照

702 ページの『XML GENERATE 例外』

SFS ファイル

COBOL for AIX アプリケーションで SFS ファイルを使用する場合、cob2_r コマンドでそのアプリケーションをコンパイルしてください。cob2_r4 コマンド (SFS ファイル・システムを使用する COBOL for AIX バージョン 2 アプリケーションのコンパイルに以前は推奨されていたコマンド) はサポートされません。

シェル・スクリプトを使用して COBOL for AIX アプリケーションをコンパイルする場合は、シェル・スクリプトを相応のレベルに更新しておいてください。

関連タスク

237 ページの『シェル・スクリプトを使用したコンパイル』

238 ページの『SFS ファイルのためのコンパイル』

関連参照

125 ページの『SFS ファイル・システム』

32 ビット・アプリケーションの 64 ビット・モードへの移行

64 ビット・モードでは、実行結果が変わる可能性があります。32 ビット・アプリケーションを 64 ビット・モードに移行した場合、その相違を調整するためにいくつか記録しなければならない可能性があります。

64 ビット・モードに移行するときに主に考慮しなければならないことは、データ項目のサイズが変わることですが、下記の説明にあるように、アプリケーションの他の領域にも影響が及ぶ可能性があります。

LENGTH OF 特殊レジスター:

32 ビット・モードでは、LENGTH OF 特殊レジスターの暗黙的な定義は PICTURE 9(9) USAGE IS BINARY です。64 ビット・モードでは、暗黙的な定義は PICTURE 9(18) USAGE IS BINARY です。

LENGTH 組み込み関数:

32 ビット・モードでは、FUNCTION LENGTH は 9 桁の整数を返します。64 ビット・モードでは、この関数は 18 桁の整数を返します。

アドレス・データ項目および指標データ項目:

アドレスまたは指標を含むデータ項目に対するストレージ割り振りは、32 ビット・モードでは 4 バイトであり、64 ビット・モードでは 8 バイトです。影響を受けるデータ項目は、以下のいずれかを使用したときに一緒に定義されるデータ項目です。

- POINTER
- FUNCTION-POINTER
- PROCEDURE-POINTER
- OBJECT REFERENCE
- INDEX

特殊レジスター ADDRESS OF および JNIEnvPtr は、USAGE POINTER として暗黙的に定義されます。そのため、USAGE POINTER データ項目について上で説明したように、これらに関してもストレージが割り振られます。

グループ項目のレイアウト:

上記の説明にあるように 32 ビット・モードのプログラムと 64 ビット・モードのプログラムとの間でアドレス・データ項目と指標データ項目の割り振りサイズで変更がある場合、グループ項目のレイアウトに影響が及びます。特に、アドレス・データ項目または指標データ項目の後に配置されるデータ項目は、グループ内の別のオフセットに配置される可能性があります。遊びバイト (正しく位置合わせが行われるようにコンパイラによって挿入されるバイト) の数が異なる可能性があります。そのため、グループの合計サイズが変わる場合があります。

SYNCHRONIZED データ項目:

アドレス・データ項目または指標データ項目に対して SYNCHRONIZED 文節が指定されている場合、そのデータ項目は 32 ビット・モードではフルワード境界に調整され、64 ビット・モードではダブルワード境界に調整されます。上記のグループ項目の説明にあるように、グループ内のデータ項目のオフセット、遊びバイトの数、合計グループ・サイズが変わる可能性があります。

データ・ファイル:

32 ビット・モードのプログラムで作成されたデータ・ファイルは 64 ビット・モードのプログラムで処理できます。また、64 ビット・モードのプログラムで作成されたデータ・ファイルは 32 ビット・モードのプログラムで処理できます。アドレッシング・モードはデータ・ファイルに加えられません。ただし、レコード定義にアドレス・データ項目または指標データ項目が含まれている場合に 64 ビット・モードに移行すると、レコードの長さが変わる (そのため、ファイルの長さが変わる) 可能性があります。

ミドルウェア・サブシステム:

プログラムがデータ処理タスクを実行するときにミドルウェア・サブシステムに依存する場合、そのサブシステムが 64 ビット・プログラムとともに動作することを確認してください。

制約事項: DB2 バージョン 8 および CICS TXSeries は、32 ビット COBOL アプリケーションのみをサポートします。

DB2 バージョン 9 は、32 ビットまたは 64 ビットの COBOL アプリケーションをサポートします。

その他の考慮事項:

AIX では、1 つのアプリケーション内に 32 ビット・モードのプログラムと 64 ビット・モードのプログラムを混在させることはできません。リンカーは、アドレッシング・モードの異なるオブジェクト・ファイル同士のリンクをサポートしません。64 ビットのプログラムで、言語間呼び出しを行ったり、ユーザー・ライブラリーにあるルーチンを読み出したりする場合は、呼び出されるルーチンの 64 ビット・バージョンが使用可能であることを確認してください。

関連参照

266 ページの『ADDR』

616 ページの『データ・サイズ制限』

コンパイラ限界値 (COBOL for AIX 言語解説書)

付録 B. ホスト COBOL との違いの要約

IBM COBOL for AIX は、Enterprise COBOL for z/OS とは別の方法で特定の項目を実装します。詳細については、以下の関連参照を参照してください。

関連タスク

493 ページの『第 25 章 プラットフォーム間でのアプリケーションの移植』

関連参照

『コンパイラー・オプション』
620 ページの『データ表現』
621 ページの『環境変数』
622 ページの『ファイル指定』
622 ページの『言語間通信 (ILC)』
623 ページの『入出力』
623 ページの『ランタイム・オプション』
623 ページの『ソース・コードの行サイズ』
623 ページの『言語エレメント』

コンパイラー・オプション

COBOL for AIX は、コンパイラー・オプション ADV、AWO、BINARY、BUFSIZE、CODEPAGE、DATA、DECK、DBCS、FASTSRT、FLAGMIG、INTDATE、LANGUAGE、NAME、OUTDD、および RENT をコメントとして扱います (コンパイラーは常に再入可能コードを生成します)。これらのオプションには、I レベルのメッセージ付きでフラグが立てられます。

COBOL for AIX は、NOADV コンパイラー・オプションもコメントとして扱います。このオプションには、W レベルのメッセージ付きでフラグが立てられます。このオプションを指定すると、WRITE ステートメントを ADVANCING 句とともに使用したときに、アプリケーションに予期しない結果が生じる可能性があります。

OO COBOL アプリケーションの場合は、cob2_j コマンドの -host オプションを指定したり、CHAR または FLOAT コンパイラー・オプションの何らかの設定を指定したりできます。ただし、メソッドの引数またはパラメーター、および JNI サービスに対する引数として使用される 浮動小数点項目は、ネイティブ形式で指定する (例えば、データ記述記入項目に NATIVE 段落を使用する) 必要があります。

関連タスク

493 ページの『コンパイルするメインフレーム・アプリケーションの取得』

関連参照

245 ページの『cob2、cob2_j、および cob2_r オプション』
271 ページの『CHAR』

データ表現

データの表示は、IBM ホスト COBOL と IBM COBOL for AIX の間で異なることがあります。

ゾーン 10 進数データ

ゾーン 10 進数データの符号表現が ASCII と EBCDIC のどちらに基づくかは、CHAR コンパイラー・オプションの設定 (NATIVE または EBCDIC) と、USAGE 文節が NATIVE 句で指定されているかどうかによって決まります。COBOL for AIX では、ゾーン 10 進数データの符号表現の処理は、コンパイラー・オプション NUMPROC(NOPFD) が有効な場合にホスト上で行われる処理と整合性が取れています。

パック 10 進データ

符号なしパック 10 進数のサイン表記は、COBOL for AIX と ホスト COBOL で異なります。COBOL for AIX は、符号なしパック 10 進数の符号ニブル x'C' を使用します。ホスト COBOL では、符号なしパック 10 進数に対して符号ニブル x'F' が使用されます。AIX および z/OS 間のパック 10 進数を含むデータ・ファイルを共用する場合は、符号なしパック 10 進数の代わりに符号ありパック 10 進数を使用することをお勧めします。

浮動小数点データの表示

FLOAT(S390) コンパイラー・オプションを使用すると、浮動小数点データ項目の表示がネイティブ (IEEE) 形式ではなく zSeries データ表現 (16 進数) であることを指定できます。

zSeries 形式の浮動小数点項目の表示を、INVOKE ステートメントの引数またはメソッドのパラメーターとして指定しないでください。これらの引数やパラメーターは、Java データ型と相互運用可能にするために、ネイティブ形式で指定する必要があります。

国別データ

COBOL for AIX およびホスト COBOL では、国別データに UTF-16 ビッグ・エンディアン形式を使用します。

EBCDIC および ASCII データ

英数字データ項目に EBCDIC 照合シーケンスを指定する際には、次の言語エレメントを使用することができます。

- ALPHABET 文節
- PROGRAM COLLATING SEQUENCE 文節
- SORT または MERGE ステートメントの COLLATING SEQUENCE 句

CHAR(EBCDIC) コンパイラー・オプションを指定すると、DISPLAY データ項目が zSeries データ表現 (EBCDIC) であることを指定できます。

データ変換用のコード・ページの決定

英字、英数字、DBCS、および国別データ項目の場合は、実行時に有効なロケールから、ネイティブ文字の暗黙変換に使用されるソース・コード・ページが決定されます。

英数字、DBCS、および国別リテラルの場合は、コンパイル時に有効なロケールから、文字の暗黙変換に使用されるソース・コード・ページが決定されます。

DBCS 文字ストリング

COBOL for AIX では、ASCII DBCS 文字ストリングの区切りにシフトイン/シフトアウト文字は使用されません。ただし例外として、次に説明するようにダミーのシフトイン/シフトアウト文字を使用することは可能です。

SOSI コンパイラー・オプションを使用すると、AIX ワークステーションのシフトアウト (X'1E') およびシフトイン (X'1F') 制御文字が、ソース・プログラム内の DBCS 文字ストリング (ユーザー定義語、DBCS リテラル、英数字リテラル、国別リテラル、コメントを含む) を区切るように指定することができます。ホストのシフトアウト (X'0E') およびシフトイン (X'0F') 制御文字は一般に、COBOL for AIX ソース・コードのダウンロード時に、使用するダウンロード方法に応じてワークステーションのシフトアウトおよびシフトイン制御文字に変換されます。

英数字リテラル内に制御文字 X'00' から X'1F' を使用すると、予測不能な結果が生じる可能性があります。

関連タスク

209 ページの『第 11 章 ロケールの設定』

494 ページの『データ表現による違いの修正』

関連参照

271 ページの『CHAR』

305 ページの『SOSI』

環境変数

COBOL for AIX は、次のリストにある複数の環境変数を認識します。

- 割り当て名
- COBMSGS
- COBPATH
- COBRTOPT
- DB2DBDFT
- EBCDIC_CODEPAGE
- ENCINA_CDS_ROOT
- ENCINA_SFS_DATA_VOLUME
- ENCINA_SFS_INDEX_VOLUME
- ENCINA_VSAM_AUTO_FLUSH

- ENCINA_VSAM_CACHE
- LANG
- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MESSAGES
- LC_TIME
- LOCPATH
- NLSPATH
- TMP
- TZ

ファイル指定

COBOL for AIX は、すべてのファイルを単一ボリューム・ファイルとして扱います。それ以外のファイル指定はすべてコメントとして扱われます。この変更は、REEL、UNIT、MULTIPLE FILE TAPE 文節、および CLOSE. . .UNIT/REEL に影響しません。

言語間通信 (ILC)

ILC は、C/C++ および PL/I プログラムで使用可能です。

次に、言語環境プログラムを持つホスト上で ILC を使用した場合と比較した、AIX ワークステーション上での ILC の動作の違いを示します。

- COBOL の STOP RUN、C の exit()、または PL/I の STOP を使用した場合は、終了動作に違いがあります。
- 調整された条件処理は、ワークステーション上にはありません。COBOL プログラム間で C の longjmp() を使用することは避けてください。
- ホスト上では、プロセス内で呼び出され、なおかつ言語環境プログラムに対応している最初のプログラムが、「メイン」プログラムと見なされます。AIX 上では、プロセス内で呼び出される最初のプログラムが、COBOL による「メイン」プログラムと見なされます。この違いは、実行単位 (メインプログラムで始まる実行単位) の定義に依存する言語セマンティクスに影響します。例えば、STOP RUN を使用すると、制御権はメインプログラムの呼び出し側に戻りますが、混合言語環境では、前述とは異なる結果になる可能性があります。

関連概念

555 ページの『第 30 章 COBOL ランタイム環境の事前初期設定』

入出力

COBOL for AIX は、STL ファイル・システムおよび VSAM ファイル・システムを使用することで、順次ファイル、相対ファイル、および索引付きファイルの入出力をサポートしています。行順次の入出力は、プラットフォームのネイティブのバイト・ストリーム・ファイル・サポートを使用することでサポートしています。

ファイル・システムから戻される *data-name* のファイル状況に応じて、サイズや値は異なります。

COBOL for AIX では、磁気テープ・ドライブやディスク・ドライブを直接はサポートしていません。

関連概念

124 ページの『ファイル・システム』

ランタイム・オプション

COBOL for AIX では、次のホスト・ランタイム・オプション AIXBLD、ALL31、CBLPSHPOP、CBLQDA、COUNTRY、HEAP、MSGFILE、NATLANG、SIMVRD、STACK は認識されず、無効として扱われます。

ホストでは、STORAGE ランタイム・オプションを使用して、COBOL の WORKING-STORAGE を初期化することができます。COBOL for AIX では、WSCLEAR コンパイラー・オプションを使用します。

関連参照

315 ページの『WSCLEAR』

ソース・コードの行サイズ

COBOL ソース行に含めることのできる文字数は 72 文字未満です。各行は、72 桁目か、または改行制御文字が検出された位置で終了します。

言語エレメント

次の表では、ホスト COBOL コンパイラーと COBOL for AIX コンパイラーの間で異なる言語エレメントをリストし、可能であれば、COBOL for AIX プログラムにおいてそのような相違に対処する方法についてアドバイスしています。

ホスト COBOL で有効な COBOL の文節および句の多くは構文チェックされますが、これらは COBOL for AIX プログラムの実行には影響しません。これらの文節や句は、ダウンロードした既存のアプリケーションにわずかな影響を与えます。COBOL for AIX では、その構文が機能的な影響を持たない場合でも、ホストの COBOL 言語構文を認識します。

表 74. Enterprise COBOL for z/OS と COBOL for AIX 間の言語の違い

言語エレメント	COBOL for AIX インプリメンテーションまたは制約事項
ACCEPT ステートメント	ホスト COBOL プログラムが ACCEPT ステートメントのターゲットとして DD 名を期待する場合は、値が適切なファイル名に設定された同等の環境変数を使用して、これらのターゲットを定義します。COBOL for AIX では、 <i>environment-name</i> および関連する環境変数値 (設定されている場合) によって、ファイル ID が決まります。
APPLY WRITE-ONLY 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
ASSIGN 文節	COBOL for AIX は、 <i>assignment-name</i> に基づいたシステム・ファイル名に対し、異なる構文およびマッピングを使用します。ASSIGN. . . USING <i>data-name</i> は、ホスト COBOL ではサポートされません。
BLOCK CONTAINS 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
CALL ステートメント	COBOL for AIX では、ファイル名を CALL 引数として使用できません。
CLOSE ステートメント	FOR REMOVAL 句、WITH NO REWIND 句、および UNIT/REEL 句は、COBOL for AIX で構文が検査されますが、プログラムの実行には影響しません。移植可能にすることを目的とするプログラムで、これらの句を使用しないでください。
CODE-SET 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
DATA RECORDS 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
DISPLAY ステートメント	ホスト COBOL プログラムが DISPLAY ステートメントのターゲットとして DD 名を期待する場合は、値が適切なファイル名に設定された同等の環境変数を使用して、これらのターゲットを定義します。COBOL for AIX では、 <i>environment-name</i> および関連する環境変数値 (設定されている場合) によって、ファイル ID が決まります。
ファイル状況 <i>data-name-1</i>	ファイル状況 9x の一部の値と意味は、ホスト COBOL と COBOL for AIX とで異なります。
ファイル状況 <i>data-name-8</i>	プラットフォームおよびファイル・システムによって、形式と値が異なります。
INDEX データ項目	ホスト COBOL では、INDEX データ項目は 4 バイトとして暗黙的に定義されます。ADDR(32) でコンパイルされた COBOL for AIX プログラムの場合、サイズは 4 バイトです。コンパイル時に ADDR(64) が使用された場合、サイズは 8 バイトです。
LABEL RECORDS 文節	LABEL RECORD IS <i>data-name</i> 句、USE. . . AFTER. . . LABEL PROCEDURE 句、および GO TO MORE-LABELS 句は、構文検査されますが、COBOL for AIX のプログラムの実行には影響しません。これらのいずれかの句を使用すると、警告が発行されます。ユーザー・ラベルの宣言は実行時には呼び出されません。z/OS QSAM 対応のユーザー・ラベル処理に依存するプログラムは移植できません。
MULTIPLE FILE TAPE	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。AIX ワークステーションでは、すべてのファイルが単一ボリューム・ファイルとして扱われます。
OBJECT REFERENCE データ項目	ホスト COBOL では、OBJECT REFERENCE データ項目は 4 バイトとして暗黙的に定義されます。ADDR(32) でコンパイルされた COBOL for AIX プログラムの場合、サイズは 4 バイトです。コンパイル時に ADDR(64) が使用された場合、サイズは 8 バイトです。
OPEN ステートメント	REVERSED 句および WITH NO REWIND 句は、COBOL for AIX で構文が検査されますが、プログラムの実行には影響しません。
PASSWORD 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。

表 74. Enterprise COBOL for z/OS と COBOL for AIX 間の言語の違い (続き)

言語エレメント	COBOL for AIX インプリメンテーションまたは制約事項
POINTER、PROCEDURE-POINTER、および FUNCTION-POINTER データ項目	ホスト COBOL では、POINTER データ項目および FUNCTION-POINTER データ項目は、特殊レジスタ ADDRESS OF および JNIEnvPtr と同様に、4 バイトとして暗黙的に定義されます。PROCEDURE-POINTER データ項目は、8 バイトとして暗黙的に定義されます。ADDR(32) でコンパイルされた COBOL for AIX プログラムの場合、これらの各データ項目のサイズは 4 バイトです。コンパイル時に ADDR(64) が使用された場合、サイズは 8 バイトです。
READ . . .PREVIOUS	COBOL for AIX でのみ、DYNAMIC アクセス・モードを使用して、相対ファイルまたは索引付きファイルの前のレコードを読み取ることができます。
RECORD CONTAINS 文節	RECORD CONTAINS <i>n</i> CHARACTERS 文節は文節は受け入れられますが、例外が 1 つあります。RECORD CONTAINS 0 CHARACTERS は、COBOL for AIX で構文が検査されますが、プログラムの実行には影響しません。
RECORDING MODE 文節	相対ファイル、索引付きファイル、および行順次ファイルについては、COBOL for AIX で構文が検査されますが、プログラムの実行には影響しません。順次ファイルの場合、RECORDING MODE U は構文が検査されますが、プログラムの実行には影響しません。
RERUN 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
RESERVE 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
SAME AREA 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
SAME SORT 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
SHIFT-IN、SHIFT-OUT 特殊レジスタ	COBOL for AIX コンパイラは、CHAR(EBCDIC) コンパイラ・オプションが有効でないと、これらのレジスタが検出されたときに E レベルのメッセージを出します。
SORT-CONTROL 特殊レジスタ	この特殊レジスタの暗黙的な定義および内容は、ホスト COBOL とワークステーション COBOL とでは異なります。
SORT-CORE-SIZE 特殊レジスタ	この特殊レジスタの内容は、ホスト COBOL とワークステーション COBOL とでは異なります。
SORT-FILE-SIZE 特殊レジスタ	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。この特殊レジスタの値は使用されません。
SORT-MESSAGE 特殊レジスタ	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
SORT-MODE-SIZE 特殊レジスタ	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。この特殊レジスタの値は使用されません。
SORT MERGE AREA 文節	構文が検査されますが、COBOL for AIX のプログラムの実行には影響しません。
START. . .	COBOL for AIX では、IS LESS THAN、IS <、IS NOT GREATER THAN、IS NOT >、IS LESS THAN OR EQUAL TO、IS <= の各関係演算子を使用できます。
STOP RUN	COBOL for AIX のマルチスレッド・プログラムではサポートされません。マルチスレッド・プログラム内で、C exit() 関数に対する呼び出しで置き換えることができます。
WRITE ステートメント	COBOL for AIX では、WRITE. . .ADVANCING を C01 から C12 までの環境名または S01 から S05 までの環境名とともに指定した場合、1 行拡張されます。
XML PARSE ステートメント	ホスト専用オプション XMLPARSE(XMLSS) でコンパイルされたホスト COBOL プログラムでは、COBOL for AIX では利用できない名前空間処理用の追加構文 (ENCODING 句 および RETURNING NATIONAL 句) や特殊レジスタが使用可能です。
プラットフォーム環境に既知の名前	<i>program-name</i> 、 <i>text-name</i> 、 <i>library-name</i> 、 <i>assignment-name</i> 、SORT-CONTROL 特殊レジスタ内のファイル名、 <i>basis-name</i> 、DISPLAY または ACCEPT ターゲット識別、およびシステム依存名は、それぞれ異なる方法で識別されます。

付録 C. zSeries ホスト・データ形式についての考慮事項

zSeries ホスト・データ内部表現の使用に適用される考慮事項、制約事項、および制限を次に示します。

CHAR および FLOAT コンパイラー・オプションは、zSeries ホスト・データ形式またはネイティブのデータ形式が使用されているかどうかを判別します (COMP-5 項目、または USAGE 文節内の NATIVE 句で定義された項目以外)。(この情報の用語「ホスト・データ形式」と「ネイティブのデータ形式」は、データ項目の内部表現を指します。)

CICS アクセス

分離または統合された CICS 変換プログラムによって変換され、TXSeries 上で動作する COBOL プログラムでサポートされる zSeries ホスト・データ形式はありません。

日時呼び出し可能サービス

日付および時刻の呼び出し可能サービスでは、zSeries ホスト・データ形式の内部表現で使用することができます。呼び出し可能サービスに渡されるパラメーターはすべて、zSeries ホスト・データ形式にする必要があります。日付および時刻サービスに対する同一の呼び出しで、ネイティブ・データの内部表現とホスト・データの内部表現を混用することはできません。

浮動小数点のオーバーフロー例外

AIX ワークステーション、および zSeries ホスト上では、浮動小数点データ表現の制限に違いがあるため、FLOAT (HEX) が有効な場合は、2 つの形式間での変換中に、浮動小数点のオーバーフロー例外が発生する可能性があります。例えば、ホスト上で正常に実行できるプログラムをワークステーション上で実行すると、次のようなメッセージを受け取る可能性があります。

IWZ053S An overflow occurred on conversion to floating point

この問題を避けるため、どちらのプラットフォームでも、データ型ごとにサポートされる最大浮動小数点値に注意する必要があります。次の表に、それぞれの制限を示します。

表 75. 最大浮動小数点値

データ型	最大ワークステーション値	最大 zSeries ホスト値
COMP-1	$*(2^{128} - 2^4)$ (約 $*3.4028E+38$)	$*(16^{63} - 16^{57})$ (約 $*7.2370E+75$)

表 75. 最大浮動小数点値 (続き)

データ型	最大ワークステーション値	最大 zSeries ホスト値
COMP-2	$*(2^{1024} - 2^{971})$ (約 $*1.7977E+308$)	$*(16^{63} - 16^{49})$ (約 $*7.2370E+75$)
* 値は正数でも負数でも構いません。		

上記のとおり、ホストの COMP-1 値はワークステーションよりも大きくすることができ、ワークステーションの COMP-2 値はホストよりも大きくすることができます。

DB2

zSeries ホスト・データ形式のコンパイラー・オプションは、DB2 プログラムで使用することができます。

Java とのインターオペラビリティのためのオブジェクト指向構文

浮動小数点データ項目の zSeries 形式は、INVOKE ステートメントの引数または戻り項目として指定する必要はありません。

分散コンピューティング環境アプリケーション

zSeries ホスト・データ形式のコンパイラー・オプションは、分散コンピューティング環境プログラムで使用しないでください。

ファイル・データ

- EBCDIC データおよび 16 進数/2 進数データは、任意の順次ファイル、相対ファイル、索引付きファイルで読み取りおよび書き込みを行うことができます。自動変換は行われません。
- ホスト・データを含むファイルにアクセスする場合は、コンパイラー・オプション COLLSEQ(EBCDIC)、CHAR(EBCDIC)、および FLOAT(S390) を使用して、これらのファイルから得られる EBCDIC 文字データ、および 16 進数浮動小数点データを処理します。

SORT

zSeries ホスト・データ形式 DBCS (USAGE DISPLAY-1 を除く) は、すべてソート・キーとして使用できます。

関連概念

45 ページの『数値データの形式』

関連タスク

483 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

263 ページの『第 14 章 コンパイラー・オプション』

付録 D. 中間結果および算術精度

コンパイラーは、算術ステートメントを、演算子優先順位に従って実行される一連の操作として処理し、それらの操作の結果を入れる中間フィールドをセットアップします。コンパイラーはいくつかのアルゴリズムを使用して、確保する整数および小数部の桁数を判別します。

中間結果は、次の場合に得ることができます。

- 動詞の直後に複数のオペランドが入った ADD または SUBTRACT ステートメント。
- 一連の算術演算または複数の結果フィールドを指定する COMPUTE ステートメント。
- 条件ステートメントまたは参照変更指定に含まれている算術式。
- GIVING オプションおよび複数の結果フィールドを使用する ADD、SUBTRACT、MULTIPLY、または DIVIDE ステートメント。
- 組み込み関数をオペランドとして使用するステートメント。

633 ページの『例: 中間結果の計算』

中間結果の精度は、デフォルト・オプション ARITH(COMPAT) (互換モード と呼ばれる) を使用してコンパイルするか、または以下に説明しているように ARITH(EXTEND) (拡張モード と呼ばれる) を使用してコンパイルするかによって異なります。

互換モードでの算術演算の計算は、IBM COBOL Set for AIX の場合と変わりません。

- 固定小数点の中間結果の場合は、最大 30 桁が使用されます。
- 浮動小数点組み込み関数は、長精度 (64 ビットの) 浮動小数点結果を戻します。
- 浮動小数点オペランド、分数指数、または浮動小数点組み込み関数を含む式は、浮動小数点でないすべてのオペランドが長精度浮動小数点に変換され、式の計算に浮動小数点演算が使用されるかのように評価されます。
- 浮動小数点リテラルおよび外部浮動小数点データ項目は、長精度浮動小数点に変換されてから処理されます。

拡張モードでの算術演算の計算には、次のような特性があります。

- 固定小数点の中間結果の場合は、最大 31 桁が使用されます。
- 浮動小数点組み込み関数は、拡張精度 (128 ビットの) 浮動小数点結果を戻します。
- 浮動小数点オペランド、分数指数、または浮動小数点組み込み関数を含む式は、浮動小数点でないすべてのオペランドが拡張精度浮動小数点に変換され、式の計算に浮動小数点演算が使用されるかのように評価されます。
- 浮動小数点リテラルおよび外部浮動小数点データ項目は、拡張精度浮動小数点に変換されてから処理されます。

関連概念

45 ページの『数値データの形式』

60 ページの『固定小数点演算と浮動小数点演算の対比』

関連参照

633 ページの『固定小数点データと中間結果』

639 ページの『浮動小数点データと中間結果』

640 ページの『非算術ステートメントの算術式』

268 ページの『ARITH』

中間結果用の用語

中間結果に関するこの情報を理解するには、以下の用語を理解する必要があります。

- i*** 中間結果用に保持された整数の桁数。(ROUNDED 句を使用している場合は、正確さを得るために必要なら、整数がもう 1 桁余分に保持します。)
- d*** 中間結果用に保持された小数部の桁数。(ROUNDED 句を使用している場合は、正確さを得るために必要なら、小数部がもう 1 桁余分に保持します。)
- dmax*** 特定のステートメントで、次の項目のうち最も大きいもの。
- 最終結果フィールド (1 つ以上) に必要な小数部の桁数。
 - 除数または指数を除き、オペランドに対して定義された小数部の最大桁数。
 - 関数オペランドの *outer-dmax*。
- inner-dmax***
関数への参照で、次の項目のうち最も大きいもの。
- その基本引数に対して定義された小数部の桁数。
 - いずれかの算術式の引数の *dmax*。
 - そのいずれかの組み込み関数の *outer-dmax*。
- outer-dmax***
関数結果がそれ自体の計算の外で演算に寄与する小数部の桁数 (例えば、その関数が算術式中のオペランドであるか、または別の関数への引数である場合)。
- op1*** 生成される算術ステートメントの第 1 オペランド (除算では除数)。
- op2*** 生成される算術ステートメントの第 2 オペランド (除算では被除数)。
- i1, i2*** それぞれ *op1* と *op2* 内の整数の数値。
- d1, d2*** それぞれ、*op1* および *op2* 内の小数部の桁数。
- ir*** 生成された算術ステートメントまたは演算が実行されたときの中間結果。(中間結果は、レジスターまたは保管場所のいずれかで生成されます。)
- ir1, ir2***
連続する中間結果。(連続する中間結果は同じ保管場所にすることができます。)

関連参照

ROUNDED 句 (COBOL for AIX 言語解説書)

例: 中間結果の計算

次の例は、コンパイラーが算術ステートメントを一連の操作として実行し、必要に応じて中間結果を保管する方法を示しています。

```
COMPUTE Y = A + B * C - D / E + F ** G
```

結果は、以下の順序で計算されます。

1. F を G 乗すると *ir1* が得られる。
2. B に C を掛けると *ir2* が得られる。
3. E を D で割ると *ir3* が得られる。
4. A を *ir2* に加えると *ir4* が得られる。
5. *ir3* を *ir4* から引くと *ir5* が得られる。
6. *ir5* を *ir1* に加えると Y が得られる。

関連タスク

55 ページの『算術式の使用』

関連参照

632 ページの『中間結果用の用語』

固定小数点データと中間結果

コンパイラーは、中間結果における整数および小数点以下の桁数を、決定します。

加算、減算、乗算、および除算

次の表は、加算、減算、乗算、または除算の結果として理論上可能な精度を示しています。

演算	整数桁	小数桁
+ または -	(<i>i1</i> または <i>i2</i>) + 1、どちらか大きい方	<i>d1</i> または <i>d2</i> 、どちらか大きい方
*	<i>i1</i> + <i>i2</i>	<i>d1</i> + <i>d2</i>
/	<i>i2</i> + <i>d1</i>	(<i>d2</i> - <i>d1</i>) または <i>dmax</i> 、どちらか大きい方

算術ステートメントのオペランドを十分な小数桁で定義して、最終結果の正確度が希望どおりになるようにしなければなりません。

次の表は、互換モード (つまり、デフォルトのコンパイラー・オプション ARITH(COMPAT) が有効である場合) で加算、減算、乗算、または除算を伴う算術演算の固定小数点中間結果においてコンパイラーが保持する桁数を示しています。

$i + d$ の値	d の値	$i + dmax$ の値	ir 用に保持される桁数
<30 または =30	任意の値	任意の値	i 整数桁数および d 小数桁数
>30	< $dmax$ または = $dmax$	任意の値	$30-d$ 整数桁数および d 小数桁数
	> $dmax$	<30 または =30	i 整数桁数および $30-i$ 小数桁数
		>30	$30-dmax$ 整数桁数および $dmax$ 小数桁数

次の表は、拡張モード (つまり、コンパイラ・オプション ARITH(EXTEND) が有効である場合) で加算、減算、乗算、または除算を伴う算術演算の固定小数点中間結果においてコンパイラが保持する桁数を示しています。

$i + d$ の値	d の値	$i + dmax$ の値	ir 用に保持される桁数
<31 または =31	任意の値	任意の値	i 整数桁数および d 小数桁数
>31	< $dmax$ または = $dmax$	任意の値	$31-d$ 整数桁数および d 小数桁数
	> $dmax$	<31 または =31	i 整数桁数および $31-i$ 小数桁数
		>31	$31-dmax$ 整数桁数および $dmax$ 小数桁数

指数

指数は、式 $op1 ** op2$ で表されます。 $op2$ の特性に基づいて、コンパイラは固定小数点数のべき乗計算を次の 3 つのいずれかの方法で処理します。

- $op2$ が小数部で表されている場合は、浮動小数点命令が使用されます。
- $op2$ が整数リテラルまたは定数である場合、値 d は次のように計算されます。

$$d = d1 * |op2|$$

また、値 i は、 $op1$ の特性に基づいて、次のように計算されます。

- $op1$ がデータ名または変数である場合は、次のように計算されます。

$$i = i1 * |op2|$$

- $op1$ がリテラルまたは定数である場合、 i は、 $op1 ** |op2|$ の値の整数の桁数に等しく設定されます。

互換モード (ARITH(COMPAT) を使用してコンパイルする場合) では、コンパイラは i および d を計算し終わると、次の表に示すアクションを取り、べき乗計算の中間結果 ir を処理します。

$i + d$ の値	その他の条件	取られるアクション
<30	任意	ir において、 i の整数桁数および d の小数桁数が保持される。

$i + d$ の値	その他の条件	取られるアクション
=30	$op1$ が奇数の桁数を持つ。	ir において、 i の整数桁数および d の小数桁数が保持される。
	$op1$ が偶数の桁数を持つ。	$op2$ が整数データ名または変数である場合と同じアクション (以下を参照)。例外: リテラル 1 の累乗に計算される 30 桁の整数の場合は、 ir において、 i の整数桁数および d の小数桁数が保持される。
>30	任意	$op2$ が整数データ名または変数である場合と同じアクション (以下を参照)

拡張モード (ARITH(EXTEND) を使用してコンパイルする場合) では、コンパイラーは i および d を計算し終わると、次の表に示すアクションを取り、指数計算の中間結果 ir を処理します。

$i + d$ の値	その他の条件	取られるアクション
<31	任意	ir において、 i の整数桁数および d の小数桁数が保持される。
=31 または >31	任意	$op2$ が整数データ名または変数である場合と同じアクション (以下を参照)。例外: リテラル 1 の累乗に計算される 31 桁の整数の場合は、 ir において、 i の整数桁数および d の小数桁数が保持される。

$op2$ が負である場合は、1 という値が予備計算によって作成された結果で除算されます。使用される i と d の値は、上記に示された固定小数点データの除算規則に従って計算されます。

- $op2$ が整数データ名または変数である場合は、 $dmax$ の小数桁数と、 $30-dmax$ (互換モード) または $31-dmax$ (拡張モード) 整数桁数が使用されます。 $op1$ はそれ自体によって $(lop2l - 1)$ 回、乗算されます ($op2$ がゼロ以外の場合)。

$op2$ が 0 であると、結果は 1 になります。0 による除算とべき乗計算の SIZE ERROR 条件が適用されます。

9 桁を超える有効数字を持つ固定小数点の指数部は、常に 9 桁に切り捨てられます。指数がリテラルまたは定数である場合、E レベルのコンパイラー診断メッセージが発行されます。それ以外の場合は、実行時に通知メッセージが発行されます。

636 ページの『例: 固定小数点の算術での指数』

関連参照

632 ページの『中間結果用の用語』

636 ページの『中間結果での切り捨て』

636 ページの『バイナリー・データと中間結果』

639 ページの『浮動小数点データと中間結果』

637 ページの『固定小数点算術で評価される組み込み関数』

268 ページの『ARITH』

SIZE ERROR 句 (COBOL for AIX 言語解説書)

例: 固定小数点の算術での指数

次の例は、コンパイラーが、必要に応じて中間結果を保管しながら、ゼロ以外の整数累乗へのべき乗計算を一連の乗算として実行する方法を示しています。

```
COMPUTE Y = A ** B
```

B が 4 であれば、結果は次のように計算されます。使用される i と d の値は、固定小数点データおよび中間結果に関する乗算規則に従って計算されます (以下を参照してください)。

1. A に A を掛けると $ir1$ が得られる。
2. $ir1$ に A を掛けると $ir2$ が得られる。
3. $ir2$ に A を掛けると $ir3$ が得られる。
4. $ir3$ を $ir4$ に移動する。

$ir4$ は $dmax$ の小数桁数を持っています。B は正であるので、 $ir4$ は Y に移動されます。しかし B が -4 であった場合は、さらに 5 番目のステップが実行されます。

5. 1 を $ir4$ で割ると $ir5$ が得られる。

$ir5$ は $dmax$ の小数桁数を持っており、Y に移動されます。

関連参照

632 ページの『中間結果用の用語』

633 ページの『固定小数点データと中間結果』

中間結果での切り捨て

中間結果において桁数が互換モードの 30、または拡張モードの 31 を超えるときは常に、コンパイラーは、30 (互換モード) または 31 (拡張モード) 桁までに切り捨て、警告を表示します。切り捨てが実行時に起こった場合は、メッセージが出され、プログラムは実行を続けます。

固定小数点計算で発生する可能性のある中間結果の切り捨てを回避したい場合には、代わりに浮動小数点オペランド (COMP-1 または COMP-2) を使用してください。

関連概念

45 ページの『数値データの形式』

関連参照

633 ページの『固定小数点データと中間結果』

268 ページの『ARITH』

バイナリー・データと中間結果

2 進法オペランドを伴う操作で 18 桁より多い中間結果が必要となる場合、コンパイラーはオペランドを内部 10 進数に変換してから操作を実行します。結果フィールドが 2 進数である場合、コンパイラーは、結果を内部 10 進数から 2 進数に変換します。

2 進数オペランドが最も効率的に使用されるのは、中間結果が 9 桁を超えない場合です。

関連参照

633 ページの『固定小数点データと中間結果』

268 ページの『ARITH』

固定小数点算術で評価される組み込み関数

コンパイラーは、組み込み関数の *inner-dmax* 値および *outer-dmax* 値を、関数の特性から決定します。

整数関数

整数組み込み関数は整数を戻します。したがって、この関数の *outer-dmax* は常に 0 です。引数がすべて整数でなければならない整数関数の場合は、*inner-dmax* も常に 0 になります。

次の表に、*inner-dmax* および関数結果の精度を要約します。

関数	<i>Inner-dmax</i>	関数結果の桁精度
DATE-OF-INTEGERS	0	8
DATE-TO-YYYYMMDD	0	8
DAY-OF-INTEGERS	0	7
DAY-TO-YYYYDDD	0	7
FACTORIAL	0	30 (互換モード)、31 (拡張モード)
INTEGER-OF-DATE	0	7
INTEGER-OF-DAY	0	7
LENGTH	n/a	9
MOD	0	$\min(i1\ i2)$
ORD	n/a	3
ORD-MAX		9
ORD-MIN		9
YEAR-TO-YYYY	0	4
INTEGER		固定小数点引数の場合: 引数より 1 桁大きくなります。浮動小数点引数の場合: 30 (互換モード)、31 (拡張モード)
INTEGER-PART		固定小数点引数の場合: 引数と同じ桁数になります。浮動小数点引数の場合: 30 (互換モード)、31 (拡張モード)

混合関数

混合 組み込み関数とは、結果の型がその引数の型に依存する関数です。引数がすべて数値で、どの引数も浮動小数点でない場合、その混合関数は固定小数点です。(混合関数のいずれかの引数が浮動小数点である場合、その関数は浮動小数点命令によ

って評価され、浮動小数点結果を戻します。) 混合関数が固定小数点算術演算で評価されたときは、引数がすべて整数であれば結果は整数になり、それ以外の場合は結果は固定小数点になります。

混合関数 MAX、MIN、RANGE、REM、および SUM の場合、*outer-dmax* は常に *inner-dmax* に等しくなります (したがって、引数がすべて整数であれば、両方とも 0 になります)。これらの関数について戻される結果の精度を判別するためには、固定小数点算術演算および中間結果に関する規則 (以下を参照) を、アルゴリズムの各ステップに適用してください。

MAX

1. 最初の引数を関数結果に割り当てる。
2. 残りの引数ごとに、以下のステップを実行する。
 - a. 関数結果の代数値を引数と比較する。
 - b. 2 つの値のうちの大きな方を関数結果に割り当てる。

MIN

1. 最初の引数を関数結果に割り当てる。
2. 残りの引数ごとに、以下のステップを実行する。
 - a. 関数結果の代数値を引数と比較する。
 - b. 2 つの値のうちの小さな方を関数結果に割り当てる。

RANGE

1. MAX 用のステップを使用して、最大の引数を選択する。
2. MIN 用のステップを使用して、最小の引数を選択する。
3. 最大の引数から最小の引数を引く。
4. その差を関数結果に割り当てる。

REM

1. 引数 1 を引数 2 で割る。
2. ステップ 1 の結果からすべての非整数桁を除去する。
3. ステップ 2 の結果に引数 2 を掛ける。
4. ステップ 3 の結果を引数 1 から引く。
5. その差を関数結果に割り当てる。

SUM

1. 値 0 を関数結果に割り当てる。
2. 引数ごとに、以下のステップを実行する。
 - a. その引数を関数結果に足す。
 - b. その和を関数結果に割り当てる。

関連参照

632 ページの『中間結果用の用語』

633 ページの『固定小数点データと中間結果』

639 ページの『浮動小数点データと中間結果』

268 ページの『ARITH』

浮動小数点データと中間結果

算術式の演算が浮動小数点で計算される場合は、すべてのオペランドが浮動小数点に変換され、しかも浮動小数点命令を使用して演算が行われたかのように、式全体が計算されます。

算術式に関して以下の条件のいずれかが真である場合、浮動小数点命令を使用して算術式が計算されます。

- 受け取り側またはオペランドが COMP-1、COMP-2、外部浮動小数点、または浮動小数点リテラルである。
- 指数に小数部の桁が含まれている。
- 指数が、べき乗計算または除算演算子を含む式であり、かつ d_{max} が 0 より大きい。
- 組み込み関数が浮動小数点関数である。

互換モードでは、式が浮動小数点算術演算で計算される場合、算術演算を評価するために使用される精度は、次のように決まります。

- 受け取り側およびオペランドがすべて COMP-1 データ項目で、式に乗算またはべき乗演算が含まれていない場合には、単精度が使用されます。
- これ以外の場合には、長精度が使用されます。

長精度浮動小数点が算術式の 1 つの演算で使用された場合は、その式のすべての演算は、長精度浮動小数点命令が使用されたかのように計算されます。

拡張モードでは、式が浮動小数点算術演算で計算される場合、算術演算を評価するために使用される精度は、次のように決まります。

- 受け取り側およびオペランドがすべて COMP-1 データ項目で、式に乗算またはべき乗演算が含まれていない場合には、単精度が使用されます。
- 受け取り側およびオペランドがすべて COMP-1 または COMP-2 データ項目で、受け取り側またはオペランドの少なくとも 1 つが COMP-2 データ項目であり、かつ式に乗算またはべき乗演算が含まれていない場合には、長精度が使用されます。
- これ以外の場合には、拡張精度が使用されます。

拡張精度浮動小数点が算術式の 1 つの演算で使用された場合は、その式のすべての演算は、拡張精度浮動小数点命令が使用されたかのように計算されます。

注意: 浮動小数点演算で、指数オーバーフローが発生した中間結果フィールドがあると、ジョブは異常終了します。

浮動小数点演算で評価される指数

互換モードでは、浮動小数点の指数は、常に長浮動小数点演算を使用して評価されます。拡張モードでは、浮動小数点べき乗計算は常に、拡張精度浮動小数点算術演算を使用して評価されます。

COBOL では、負の数を小数で累乗した値は定義されていません。例えば、 $(-2) ** 3$ は -8 ですが、 $(-2) ** (3.000001)$ は定義されていません。べき乗計算が浮動小数

点で行われ、結果が未定義である可能性がある場合には、実行時に指数の値が評価され、それが実際に整数値を持つかどうか判別されます。整数値を持っていない場合には、診断メッセージが出されます。

浮動小数点演算で評価される組み込み関数

互換モードでは、浮動小数点組み込み関数は常に長精度 (64 ビット) 浮動小数点値を戻します。拡張モードでは、浮動小数点組み込み関数は常に、拡張精度 (128 ビット) の浮動小数点値を戻します。

少なくとも 1 つの浮動小数点引数を持つ混合関数は、浮動小数点算術演算を使用して評価されます。

関連参照

632 ページの『中間結果用の用語』

268 ページの『ARITH』

非算術ステートメントの算術式

算術式は、算術ステートメント以外のコンテキストでも使用できます。例えば、IF または EVALUATE ステートメントを持つ算術式を使用することができます。

このようなステートメントでは、固定小数点データを持つ中間結果および浮動小数点データを持つ中間結果に関する規則が適用されます。ただし、次のような変更があります。

- 省略された IF ステートメントは、省略されていないかのように処理されます。
- 被比較数の少なくとも 1 つが算術式であるような明示比較条件では、*dmax* は、いずれかの被比較数の任意のオペランド (除数と指数を除く) 用に定義された小数部の最大小数桁数になります。以下のいずれかの条件が真である場合は、浮動小数点算術演算の規則が適用されます。
 - いずれかの被比較数のオペランドが COMP-1、COMP-2、外部浮動小数点、または浮動小数点リテラルである。
 - 指数に小数部の桁が含まれている。
 - 指数が、べき乗計算または除算演算子を含む式であり、かつ *dmax* が 0 より大きい。

以下に例を示します。

```
IF operand-1 = expression-1 THEN . . .
```

operand-1 が COMP-2 と定義されるデータ名である場合は、*expression-1* には、たとえ固定小数点オペランドしか含まれていなくても、浮動小数点算術演算の規則が適用されます。というのは、これは固定小数点オペランドと比較されるためです。

- ある算術式と別のデータ項目または算術式との間の比較で、関係演算子が使用されない場合 (すなわち、明示的な比較条件がない場合) は、その算術式は、被比較数の属性を考慮に入れずに評価されます。以下に、その例を示します。


```
EVALUATE expression-1  
  WHEN expression-2 THRU expression-3  
  WHEN expression-4  
  . . .  
END-EVALUATE
```

上記のステートメントでは、それぞれの算術式は、その特性に応じて、固定小数点または浮動小数点算術で評価されます。

関連概念

60 ページの『固定小数点演算と浮動小数点演算の対比』

関連参照

632 ページの『中間結果用の用語』

633 ページの『固定小数点データと中間結果』

639 ページの『浮動小数点データと中間結果』

IF ステートメント (*COBOL for AIX* 言語解説書)

EVALUATE ステートメント (*COBOL for AIX* 言語解説書)

条件式 (*COBOL for AIX* 言語解説書)

付録 E. 複合 OCCURS DEPENDING ON

複合 OCCURS DEPENDING ON (複合 ODO) にはいくつかのタイプがあります。複合 ODO は、標準 COBOL 85 の拡張としてサポートされます。

コンパイラによって認められる複合 ODO の基本形式は、以下のとおりです。

- 可変位置項目またはグループ: DEPENDING ON 句を指定した OCCURS 節で記述されたデータ項目の後に、非従属基本データ項目またはグループ・データ項目が続きます。
- 可変位置テーブル: DEPENDING ON 句を指定した OCCURS 節によって記述されたデータ項目の後に、OCCURS 節によって記述された非従属データ項目が続きます。
- 可変長エレメントを持つテーブル: OCCURS 節によって記述されたデータ項目に、OCCURS 節に DEPENDING ON 句を指定して記述された従属データ項目が含まれています。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

『例: 複合 ODO』

関連タスク

645 ページの『ODO オブジェクト値を変更する際の指標エラーを防止する』

646 ページの『エレメントを可変テーブルに追加する際のオーバーレイを防止する』

関連参照

644 ページの『ODO オブジェクト値の変更の影響』

OCCURS DEPENDING ON 節 (COBOL for AIX 言語解説書)

例: 複合 ODO

次の例は、複合 ODO が現れる場合の可能なタイプを示しています。

```
01 FIELD-A.
   02 COUNTER-1                PIC S99.
   02 COUNTER-2                PIC S99.
   02 TABLE-1.
     03 RECORD-1 OCCURS 1 TO 5 TIMES
       DEPENDING ON COUNTER-1  PIC X(3).
   02 EMPLOYEE-NUMBER          PIC X(5). (1)
   02 TABLE-2 OCCURS 5 TIMES  (2) (3)
     INDEXED BY INDX.          (4)
   03 TABLE-ITEM              PIC 99.  (5)
     03 RECORD-2 OCCURS 1 TO 3 TIMES
       DEPENDING ON COUNTER-2.
   04 DATA-NUM                PIC S99.
```

定義: この例では、COUNTER-1 は ODO オブジェクトです。つまり、RECORD-1 の DEPENDING ON 節のオブジェクトです。RECORD-1 は ODO サブジェクトであると言われます。同様に、COUNTER-2 は、対応する ODO サブジェクトである RECORD-2 の ODO オブジェクトです。

上記の例で現れている複合 ODO のタイプは次のとおりです。

- (1) 可変位置項目: EMPLOYEE-NUMBER は、同じレベル 01 レコード内の可変長テーブルに続いている (ただし、従属してはいない) データ項目です。
- (2) 可変位置テーブル: TABLE-2 は、同じレベル 01 レコード内の可変長テーブルに続いている (ただし、従属してはいない) テーブルです。
- (3) 可変長エレメントを持つテーブル: TABLE-2 は、従属データ項目 RECORD-2 を含んでいるテーブルであり、この従属データ項目の出現回数は ODO オブジェクトの内容によって異なります。
- (4) 可変長エレメントを持つテーブルの指標名 INDX。
- (5) 可変長エレメントを持つテーブルのエレメント TABLE-ITEM。

長さの計算方法

各レコードの可変部分の長さは、その ODO オブジェクトとその ODO サブジェクトの長さとの積です。例えば、上記に示された複合 ODO 項目の 1 つに参照が行われるたびに、使用される際の実際の長さは、次のように計算されます。

- TABLE-1 の長さは、COUNTER-1 の内容 (RECORD-1 のオカレンスの回数) に 3 (RECORD-1 の長さ) を掛けることによって計算されます。
- TABLE-2 の長さは、COUNTER-2 の内容 (RECORD-2 のオカレンスの回数) に 2 (RECORD-2 の長さ) を掛け、TABLE-ITEM の長さを加算することによって計算されます。
- FIELD-A の長さは、COUNTER-1、COUNTER-2、TABLE-1、EMPLOYEE-NUMBER、および TABLE-2 の長さに 5 を掛けたものを加算することによって計算されます。

ODO オブジェクトの値の設定

グループ内の複合 ODO 項目を参照するには、グループ項目内のすべての ODO オブジェクトを設定しておく必要があります。例えば、上記のコードの EMPLOYEE-NUMBER を参照するには、その前に、COUNTER-1 と COUNTER-2 を設定しておかなければなりません。ただし、EMPLOYEE-NUMBER は ODO オブジェクトに直接依存して値を得るわけではありません。

制約事項: ODO オブジェクトは可變的に配置することはできません。

ODO オブジェクト値の変更の影響

DEPENDING ON 句を指定した OCCURS 節で記述されたデータ項目の後に、同じグループ内で 1 つ以上の非従属データ項目 (複合 ODO 形式) が続いている場合、ODO オブジェクトの値を変更すると、レコード内の複合 ODO 項目への後続の参照が影響を受けます。

以下に例を示します。

- 関係のある ODO 節を含んでいるグループのサイズは、ODO オブジェクトの新しい値を反映します。

- ODO オブジェクトの新しい値に基づいて、ODO サブジェクトを含んでいるグループへの移動 (MOVE) が行われます。
- ODO 節で記述された項目に続いている非従属項目の位置は、ODO オブジェクトの新しい値の影響を受けます。(非従属項目の内容を保持するためには、ODO オブジェクトの値が変更される前に、非従属項目を作業域に移動しておき、後でそれらを戻してください。)

ODO オブジェクトの値は、データをその ODO オブジェクトに移動するか、その ODO オブジェクトが含まれているグループに移動すると、変更される可能性があります。また、ODO オブジェクトが READ ステートメントのターゲットであるレコードに含まれている場合にも、その値が変更されることがあります。

関連タスク

『ODO オブジェクト値を変更する際の指標エラーを防止する』

646 ページの『エレメントを可変テーブルに追加する際のオーバーレイを防止する』

ODO オブジェクト値を変更する際の指標エラーを防止する

テーブル内の従属データ項目の ODO オブジェクトの値を変更した後に、複合 ODO 指標名 (つまり、可変長エレメントを持つテーブルの指標名) を参照する場合には、注意してください。

ODO オブジェクトの値を変更すると、テーブルの長さが変わるため、関連する複合 ODO 指標のバイト・オフセットはもう有効ではありません。ですから次のような指標名への参照をコーディングした場合、予防措置を講じなければ、予想しない結果が生じることになります。

- テーブルのエレメントへの参照
- SET *integer-data-item* TO *index-name* 形式の SET ステートメント (形式 1)
- SET *index-name* UP|DOWN BY *integer* 形式の SET ステートメント (形式 2)

この種のエラーを回避するためには、次の手順を行ってください。

1. 指標を整数データ項目に保存する。(これを行うと、暗黙の変換が行われます。整数項目は、指標のオフセットに対応するテーブル・エレメント出現番号を受け取ります。)
2. ODO オブジェクトの値を変更する。
3. ただちに整数データ項目から指標を復元する。(これを行うと、暗黙の変換が行われます。指標名は、整数項目でのテーブル・エレメント出現番号に対応するオフセットを受け取ります。オフセットは、その時点で有効なテーブルの長さに従って計算されます。)

次のコードは、ODO オブジェクト COUNTER-2 が変更される場合の指標名の保存方法と復元方法を示しています (643 ページの『例: 複合 ODO』を参照)。

```

77 INTEGER-DATA-ITEM-1      PIC 99.
. . .
  SET INDX TO 5.
*      INDX is valid at this point.
  SET INTEGER-DATA-ITEM-1 TO INDX.
*      INTEGER-DATA-ITEM-1 now has the
*      occurrence number that corresponds to INDX.
  MOVE NEW-VALUE TO COUNTER-2.

```

```

*      INDX is not valid at this point.
SET INDX TO INTEGER-DATA-ITEM-1.
*      INDX is now valid, containing the offset
*      that corresponds to INTEGER-DATA-ITEM-1, and
*      can be used with the expected results.

```

関連参照

SET ステートメント (COBOL for AIX 言語解説書)

エレメントを可変テーブルに追加する際のオーバーレイを防止する

同じグループ内で 1 つ以上の非従属データ項目が後に続いている可変オカレンス・テーブル内のエレメントの数を増やす場合には、注意してください。ODO オブジェクトの値を増分し、テーブルにエレメントを追加すると、テーブルの後に続く可変位置データ項目を誤ってオーバーレイする可能性があります。

この種のエラーを回避するためには、次の手順を行ってください。

1. テーブルの後に続く可変位置データ項目を別のデータ域に保管する。
2. ODO オブジェクトの値を増分する。
3. データを新しいテーブル・エレメントに移動する (必要な場合)。
4. 可変位置データ項目を、それらを保管したデータ域から復元する。

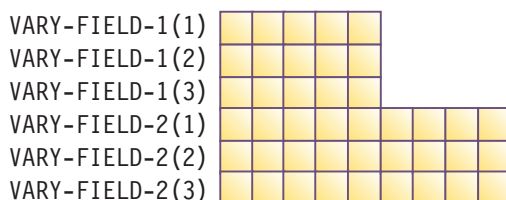
次の例では、テーブル VARY-FIELD-1 にエレメントを追加しますが、このテーブルのエレメントの数は ODO オブジェクト CONTROL-1 に左右されます。VARY-FIELD-1 の後には、非従属可変位置データ項目である GROUP-ITEM-1 が続いており、このエレメントがオーバーレイされる可能性があります。

```

WORKING-STORAGE SECTION.
01 VARIABLE-REC.
   05 FIELD-1                PIC X(10).
   05 CONTROL-1              PIC $99.
   05 CONTROL-2              PIC $99.
   05 VARY-FIELD-1 OCCURS 1 TO 10 TIMES
      DEPENDING ON CONTROL-1 PIC X(5).
   05 GROUP-ITEM-1.
      10 VARY-FIELD-2
         OCCURS 1 TO 10 TIMES
         DEPENDING ON CONTROL-2 PIC X(9).
01 STORE-VARY-FIELD-2.
   05 GROUP-ITEM-2.
      10 VARY-FLD-2
         OCCURS 1 TO 10 TIMES
         DEPENDING ON CONTROL-2 PIC X(9).

```

VARY-FIELD-1 の各エレメントは 5 バイトで、VARY-FIELD-2 の各エレメントは 9 バイトです。CONTROL-1 と CONTROL-2 の両方に値 3 が含まれている場合は、VARY-FIELD-1 および VARY-FIELD-2 のストレージは次のように示すことができます。



付録 F. 日時呼び出し可能サービス

日付および時刻の呼び出し可能サービスを使用すると、現行の現地時間および日付をいくつかの形式で入手し、日時の変換を行うことができます。

使用可能な日付および時刻呼び出し可能サービスを以下に示します。2つのサービス CEEQCEN と CEESCEN は、2桁の年号 (例えば、1991 を表す 91、2008 を表す 08 など) を扱う予測可能な方法を提供します。

表 76. 日時呼び出し可能サービス

呼び出し可能サービス	説明
651 ページの『CEECBLDY: 日付から COBOL 整数形式への変換』	文字日付値を COBOL 整数日付形式に変換します。1日目は 1601 年 1 月 1 日で、その後 1 日ごとに値が 1 ずつ増えます。
655 ページの『CEEDATE: リリアン日付から文字形式への変換』	リリアン形式の日付を文字値に変換します。
659 ページの『CEEDATM: 秒から文字タイム・スタンプへの変換』	秒数を文字タイム・スタンプに変換します。
663 ページの『CEEDAYS: 日付からリリアン形式への変換』	文字日付値をリリアン形式に変換します。1日目は 1582 年 10 月 15 日で、その後 1 日ごとに値が 1 ずつ増えます。
666 ページの『CEEDYWK: リリアン日付からの曜日の計算』	曜日計算を行います。
669 ページの『CEEGMT: 現在のグリニッジ標準時の取得』	現在のグリニッジ標準時 (日時) を取得します。
671 ページの『CEEGMTO: グリニッジ標準時から現地時間までのオフセットの取得』	グリニッジ標準時と現地時間の時差を取得します。
673 ページの『CEEISEC: 整数から秒への変換』	2進数の年、月、日、時、分、秒、ミリ秒を、1582 年 10 月 15 日の 00:00:00 から数えた秒数を表す数値に変換します。
676 ページの『CEELOCT: 現在の現地日時の取得』	現在の日時を取得します。
678 ページの『CEEQCEN: 世紀ウィンドウの照会』	呼び出し可能サービスの世紀ウィンドウを照会します。
679 ページの『CEESCEN: 世紀ウィンドウの設定』	呼び出し可能サービスの世紀ウィンドウを設定します。
681 ページの『CEESECI: 秒から整数への変換』	1582 年 10 月 15 日の 00:00:00 から数えた秒数を表す数値を、年、月、日、時、分、秒、ミリ秒を表す 7 つの 2 進整数に変換します。
685 ページの『CEESECS: タイム・スタンプの秒への変換』	文字タイム・スタンプ (日時) を、1582 年 10 月 15 日の 00:00:00 から数えた秒数に変換します。
689 ページの『CEEUTC: 協定世界時の取得』	CEEGMT と同じ。

表 76. 日時呼び出し可能サービス (続き)

呼び出し可能サービス	説明
689 ページの『IGZEDT4: 現在日付の取得』	4 桁年号を使用した現在日付を YYYYMMDD 形式で戻します。

これらすべての日付および時刻呼び出し可能サービスで、Enterprise COBOL for z/OS とのソース・コード互換性があります。ただし、条件の処理方法には大きな違いがあります。

日付および時刻の呼び出し可能サービスは、以下に表す日付/時刻組み込み関数への追加です。

表 77. 日時組み込み関数

組み込み関数	説明
CURRENT-DATE	現在の日時とグリニッジ標準時からの時間差
DATE-OF-INTEGERS ¹	整数で表された日付に相当する標準フォーマットの日付 (YYYYMMDD)
DATE-TO-YYYYMMDD ¹	指定された 100 年間隔に従ったウィンドウ化西暦年を使用した、整数表現の日付に相当する標準フォーマットの日付 (YYYYMMDD)
DATEVAL ¹	整数または英数字で表された日付に相当する日付フィールド
DAY-OF-INTEGERS ¹	整数で表された日付に相当する年間通算日フォーマットの日付 (YYYYDDD)
DAY-TO-YYYYDDD ¹	指定された 100 年間隔に従ったウィンドウ化西暦年を使用した、整数表現の日付に相当するユリウス日付 (YYYYMMDD)
INTEGER-OF-DATE	標準フォーマットの日付 (YYYYMMDD) に相当する整数で表された日付
INTEGER-OF-DAY	年間通算日 (YYYYDDD) に相当する整数で表された日付
UNDATE ¹	整数または英数字で表された日付フィールドに相当する非日付
YEAR-TO-YYYY ¹	指定された 100 年間隔に従ったウィンドウ化西暦年に相当する拡張西暦年 (YYYY)
YEARWINDOW ¹	YEARWINDOW コンパイラー・オプションで指定された世紀ウィンドウの開始年
1. DATEPROC コンパイラー・オプションの設定によって動作が異なります。	

605 ページの『例: 出力用の日付形式』

関連参照

606 ページの『フィードバック・トークン』

CALL ステートメント (COBOL for AIX 言語解説書)

関数定義 (COBOL for AIX 言語解説書)

CEECBLDY: 日付から COBOL 整数形式への変換

CEECBLDY は、日付を表す文字列を 1600 年 12 月 31 日から数えた日数に変換します。日付および時刻の呼び出し可能サービスの世紀ウィンドウにアクセスする場合や、COBOL 組み込み関数を使用して日付計算を実行する場合には、CEECBLDY を使用します。

このサービスは CEEDAYS と似ていますが、COBOL 組み込み関数と互換性のある COBOL 整数形式で文字列を戻す点が異なります。

CALL CEECBLDY の構文

```
▶▶—CALL—"CEECBLDY"—USING—input_char_date,—picture_string,————▶▶
▶—output_Integer_date,—fc.————▶▶
```

input_char_date (入力)

picture_string の指定に準拠した形式で日付またはタイム・スタンプを表す、ハーフワード長の接頭部の付いた文字列。

文字列に含まれる文字数は 5 から 255 文字です。

input_char_date には、先行または末尾空白を含めることができます。日付の構文解析は、最初の非空白文字から始まります (ピクチャー・文字列自体に先行空白が含まれる場合は、CEECBLDY がその位置を正確にスキップした後、構文解析が始まります)。

CEECBLDY は、*picture_string* で指定された日付形式によって判別される有効な日付を解析したら、残りの文字をすべて無視します。有効な日付範囲は、1601 年 1 月 1 日から 9999 年 12 月 31 日です。

picture_string (入力)

input_char_date で指定された日付の形式を示す、ハーフワード長の接頭部の付いた文字列。

picture_string 内の各文字は、*input_char_date* 内の文字に対応します。例えば、MMDDYY を *picture_string* として指定すると、CEECBLDY は *input_char_date* の値 060288 を 1988 年 6 月 2 日として読み取ります。

スラッシュ (/) などの区切り文字がピクチャー・文字列内にある場合は、先行ゼロを省略することができます。例えば、次の CEECBLDY の呼び出しは、同じ値 141502 (1988 年 6 月 2 日) をそれぞれ COBINTDTE に割り当てます。

```
MOVE '6/2/88' TO DATEVAL-STRING.
MOVE 6 TO DATEVAL-LENGTH.
MOVE 'MM/DD/YY' TO PICSTR-STRING.
MOVE 8 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.
```

```

MOVE '06/02/88' TO DATEVAL-STRING.
MOVE 8 TO DATEVAL-LENGTH.
MOVE 'MM/DD/YY' TO PICSTR-STRING.
MOVE 8 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.

MOVE '060288' TO DATEVAL-STRING.
MOVE 6 TO DATEVAL-LENGTH.
MOVE 'MMDDYY' TO PICSTR-STRING.
MOVE 6 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.

MOVE '88154' TO DATEVAL-STRING.
MOVE 5 TO DATEVAL-LENGTH.
MOVE 'YYDDD' TO PICSTR-STRING.
MOVE 5 TO PICSTR-LENGTH.
CALL CEECBLDY USING DATEVAL, PICSTR, COBINTDTE, FC.

```

picture_string にコロンやスラッシュなどの文字 (例: HH:MI:SS YY/MM/DD) が含まれる場合はブレースホルダーと見なされますが、それ以外の場合は無視されます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*input_char_date* の YY の位置に、日本元号での年号が入ります。例えば、1988 年は日本の昭和 63 年に相当します。

output_Integer_date (出力)

COBOL 整数日付 (1600 年 12 月 31 日から数えた日数) を表す 32 ビットの 2 進整数。例えば、1988 年 5 月 16 日は、日数 141485 に相当します。

input_char_date に有効な日付が含まれていない場合は、*output_Integer_date* が 0 に設定され、CEECBLDY が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_Integer_date は整数なので、*output_Integer_date* では日付計算を容易に行うことができます。うるう年や年末偏差は計算に影響しません。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 78. CEECBLDY のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EB	3	2507	CEEDAYS または CEESECS に渡されたデータが不十分です。リリアン日付の値は計算されませんでした。
CEE2EC	3	2508	CEEDAYS または CEESECS に渡された日付値が無効です。
CEE2ED	3	2509	CEEDAYS または CEESECS に渡された元号が認識されませんでした。

表 78. CEECBLDY のシンボリック条件 (続き)

シンボリック・フィールドバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EO	3	2520	CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。
CEE2EP	3	2521	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。

使用上の注意

- CEECBLDY の呼び出しは必ず、戻り値を COBOL 組み込み関数への入力として使用する COBOL プログラムから行ってください。CEEDAYS とは異なり、CEECEBLDY の逆関数はありません。CEECEBLDY は、COBOL ユーザーが日付および時刻の世紀ウィンドウ・サービスを COBOL 組み込み関数とともに使用して、日付計算を行う場合にのみ使用されます。CEECEBLDY の逆は、DATE-OF-INTEGGER および DAY-OF-INTEGGER 組み込み関数によって実現されます。
- 1601 年 1 月 1 日よりも前の日付に対して計算を実行するには、各日付の年号に 4000 を加算し、その日付を COBOL 整数形式に変換してから計算します。計算結果が日数ではなく日付になる場合は、計算結果を日付ストリングに変換し、年号から 4000 を減算します。
- デフォルトでは、2 桁の年号は、システム日付より 80 年前から始まる 100 年間にあります。したがって、2008 年の場合、2 桁の年号はすべて 1928 から 2027 年の範囲内の日付を表します。このデフォルトの範囲を変更するには、CEECEBEN 呼び出し可能サービスを使用します。

例

CBL LIB

```
*****
**
** Function: Invoke CEECBLDY callable service
** to convert date to COBOL integer format.
**
** This service is used when using the
** Century Window feature of the 日付および時刻
** callable services mixed with COBOL
** intrinsic functions.
**
**
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDY.
*
DATA DIVISION.
```

```

WORKING-STORAGE SECTION.
01  CHRDATE.
    02  Vstring-length      PIC S9(4) BINARY.
    02  Vstring-text.
        03  Vstring-char    PIC X
            OCCURS 0 TO 256 TIMES
            DEPENDING ON Vstring-length
            of CHRDATE.

01  PICSTR.
    02  Vstring-length      PIC S9(4) BINARY.
    02  Vstring-text.
        03  Vstring-char    PIC X
            OCCURS 0 TO 256 TIMES
            DEPENDING ON Vstring-length
            of PICSTR.

01  INTEGER                  PIC S9(9) BINARY.
01  NEWDATE                  PIC 9(8).
01  FC.
    02  Condition-Token-Value.
    COPY CEEIGZCT.
        03  Case-1-Condition-ID.
            04  Severity      PIC S9(4) COMP.
            04  Msg-No        PIC S9(4) COMP.
        03  Case-2-Condition-ID
            REDEFINES Case-1-Condition-ID.
            04  Class-Code    PIC S9(4) COMP.
            04  Cause-Code    PIC S9(4) COMP.
        03  Case-Sev-Ctl      PIC X.
        03  Facility-ID       PIC XXX.
    02  I-S-Info              PIC S9(9) COMP.

```

```

*
PROCEDURE DIVISION.
PARA-CBLDAYS.
*****
** Specify input date and length          **
*****
    MOVE 25 TO Vstring-length of CHRDATE.
    MOVE '1 January 00'
      to Vstring-text of CHRDATE.
*****
** Specify a picture string that describes **
** input date, and set the string's length. **
*****
    MOVE 23 TO Vstring-length of PICSTR.
    MOVE 'ZD Mmmmmmmmmmmz YY'
      TO Vstring-text of PICSTR.
*****
** Call CEECBLDY to convert input date to a **
** COBOL integer date                       **
*****
    CALL 'CEECBLDY' USING CHRDATE, PICSTR,
      INTEGER, FC.
*****
** If CEECBLDY runs successfully, then compute **
** the date of the 90th day after the          **
** input date using Intrinsic Functions      **
*****
    IF CEE000 of FC THEN
        COMPUTE INTEGER = INTEGER + 90
        COMPUTE NEWDATE = FUNCTION
          DATE-OF-INTEGGER (INTEGER)
        DISPLAY NEWDATE
          ' is Lilian day: ' INTEGER
    ELSE
        DISPLAY 'CEEBLDY failed with msg '
          Msg-No of FC UPON CONSOLE

```

```
        STOP RUN
      END-IF.
*
      GOBACK.
```

関連参照

607 ページの『ピクチャー文字項およびストリング』

CEEDATE: リリアン日付から文字形式への変換

CEEDATE は、リリアン日付を表す数値を文字形式の日付に変換します。出力は、2008/04/23 などの文字ストリングになります。

CALL CEEDATE の構文

```
▶▶—CALL—"CEEDATE"—USING—input_Lilian_date,—picture_string,————▶▶
▶—output_char_date,—fc.————▶▶
```

input_Lilian_date (入力)

リリアン日付を表す 32 ビットの整数。このリリアン日付は、1582 年 10 月 14 日から数えた日数です。例えば、1988 年 5 月 16 日は、リリアン日数 148138 に相当します。有効なリリアン日付の範囲は 1 から 3,074,324 (1582 年 10 月 15 日から 9999 年 12 月 31 日) です。

picture_string (入力)

output_char_date の必要な形式 (例: MM/DD/YY) を表す、ハーフワード長の接頭部の付いた文字ストリング。*picture_string* 内の各文字は、*output_char_date* 内の文字を表します。スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、現状のまま *output_char_date* にコピーされます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*output_char_date* の YY の位置に、日本元号での年号が入ります。例えば、1988 年は日本の昭和 63 年に相当します。

output_char_date (出力)

input_Lilian_date を *picture_string* で指定された形式に変換した結果として生成される、固定長の 80 文字のストリング。*input_Lilian_date* が無効な場合は、*output_char_date* がすべてブランクに設定され、CEEDATE が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 79. CEEDATE のシンボリック条件

シンボリック・フィールドバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EG	3	2512	CEEDATE または CEEDYWK への呼び出しで渡されたリリアン日付値が、対応範囲内にありませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EQ	3	2522	CEEDATE に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、または <CCCCCCCC>) が使用されていましたが、リリアン日付値が対応範囲内にありませんでした。元号を判別できませんでした。
CEE2EU	2	2526	CEEDATE によって戻された日付ストリングが切り捨てられました。
CEE2F6	1	2534	CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。出力はブランクに設定されました。

使用上の注意: CEEDATE の逆は CEEDAYS です。CEEDAYS は文字日付をリリアン形式に変換します。

例

CBL LIB

```

*****
**                                     **
** Function: CEEDATE - convert Lilian date to **
**                                     character format **
**                                     **
** In this example, a call is made to CEEDATE **
** to convert a Lilian date (the number of **
** days since 14 October 1582) to a character **
** format (such as 6/22/98). The result is **
** displayed. The Lilian date is obtained **
** via a call to CEEDAYS. **
**                                     **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDATE.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LILIAN                PIC S9(9) BINARY.
01 CHRDATE              PIC X(80).
01 IN-DATE.
   02 Vstring-length    PIC S9(4) BINARY.
   02 Vstring-text.
   03 Vstring-char      PIC X
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Vstring-length
                        of IN-DATE.
01 PICSTR.
   02 Vstring-length    PIC S9(4) BINARY.
   02 Vstring-text.

```



```

          03 Vstring-char    PIC X
              OCCURS 0 TO 256 TIMES
              DEPENDING ON Vstring-length
              of PICSTR.
01 FC.
  02 Condition-Token-Value.
  COPY CEEIGZCT.
    03 Case-1-Condition-ID.
      04 Severity    PIC S9(4) COMP.
      04 Msg-No      PIC S9(4) COMP.
    03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) COMP.
      04 Cause-Code PIC S9(4) COMP.
    03 Case-Sev-Ctl  PIC X.
    03 Facility-ID   PIC XXX.
  02 I-S-Info        PIC S9(9) COMP.
*
PROCEDURE DIVISION.
PARA-CBLDAYS.
*****
** Call CEEDAYS to convert date of 6/2/98 to **
** Lilian representation **
*****
MOVE 6 TO Vstring-length of IN-DATE.
MOVE '6/2/98' TO Vstring-text of IN-DATE(1:6).
MOVE 8 TO Vstring-length of PICSTR.
MOVE 'MM/DD/YY' TO Vstring-text of PICSTR(1:8).
CALL 'CEEDAYS' USING IN-DATE, PICSTR,
LILIAN, FC.

*****
** If CEEDAYS runs successfully, display result**
*****
IF CEE000 of FC THEN
  DISPLAY Vstring-text of IN-DATE
  ' is Lilian day: ' LILIAN
ELSE
  DISPLAY 'CEEDAYS failed with msg '
  Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

*****
** Specify picture string that describes the **
** desired format of the output from CEEDATE, **
** and the picture string's length. **
*****
MOVE 23 TO Vstring-length OF PICSTR.
MOVE 'ZD Mmmmmmmmmmmmmz YYYY' TO
Vstring-text OF PICSTR(1:23).

*****
** Call CEEDATE to convert the Lilian date **
** to a picture string. **
*****
CALL 'CEEDATE' USING LILIAN, PICSTR,
CHRDATE, FC.

*****
** If CEEDATE runs successfully, display result**
*****
IF CEE000 of FC THEN
  DISPLAY 'Input Lilian date of ' LILIAN
  ' corresponds to: ' CHRDATE
ELSE
  DISPLAY 'CEEDATE failed with msg '

```

```

Msg-No of FC UPON CONSOLE
STOP RUN
END-IF.

GOBACK.

```

次の表に、CEEDATE からの出力例を示します。

input_Lilian_date	picture_string	output_char_date
148138	YY YYMM YY-MM YYMMDD YYYYMMDD YYYY-MM-DD YYYY-ZM-ZD <JJJJ> YY.MM.DD	98 9805 98-05 980516 19980516 1998-05-16 1998-5-16 昭和 63.05.16 (DBCS ストリング)
148139	MM MMDD MM/DD MMDDYY MM/DD/YYYY ZM/DD/YYYY	05 0517 05/17 051798 05/17/1998 5/17/1998
148140	DD DDMM DDMMYY DD.MM.YY DD.MM.YYYY DD Mmm YYYY	18 1805 180598 18.05.98 18.05.1998 18 May 1998
148141	DDD YYDDD YY.DDD YYYY.DDD	140 98140 98.140 1998.140
148142	YY/MM/DD HH:MI:SS.99 YYYY/ZM/ZD ZH:MI AP	98/05/20 00:00:00.00 1998/5/20 0:00 AM
148143	WWW., MMM DD, YYYY Www., Mmm DD, YYYY Wwwwwwwwww, Mmmmmmmmm DD, YYYY Wwwwwwwwwz, Mmmmmmmmmz DD, YYYY	SAT., MAY 21, 1998 Sat., May 21, 1998 Saturday, May 21, 1998 Saturday, May 21, 1998

609 ページの『例: 日時のピクチャー・ストリング』

関連参照

607 ページの『ピクチャー文字項およびストリング』

CEEDATM: 秒から文字タイム・スタンプへの変換

CEEDATM は、1582 年 10 月 14 日の 00:00:00 から数えた秒数を表す数値を、文字ストリングに変換します。出力は、1988/07/26 20:37:00 などの文字ストリングのタイム・スタンプになります。

CALL CEEDATM の構文

```
▶▶—CALL—“CEEDATM”—USING—input_seconds,—picture_string,—————▶▶  
▶—output_timestamp,—fc.—————▶▶
```

input_seconds (入力)

1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 ($24 \times 60 \times 60 + 01$) に相当します。*input_seconds* の有効範囲は 86,400 から 265,621,679,999.999 (9999 年 12 月 31 日の 23:59:59.999) です。

picture_string (入力)

output_timestamp の必要な形式 (例: MM/DD/YY HH:MI AP) を表す、ハーフワード長の接頭部の付いた文字ストリング。

picture_string 内の各文字は、*output_timestamp* 内の文字を表します。スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、現状のまま *output_timestamp* にコピーされます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*output_timestamp* の YY の位置に、日本元号での年号が入ります。

output_timestamp (出力)

input_seconds を *picture_string* で指定された形式に変換した結果として生成される、固定長の 80 文字のストリング。

必要に応じて、出力が *output_timestamp* の長さまで切り詰められます。

input_seconds が無効な場合は、*output_timestamp* がすべてブランクに設定され、CEEDATM が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 80. CEEDATM のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。

表 80. CEEDATM のシンボリック条件 (続き)

シンボリック・フィールドバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE2E9	3	2505	CEEDATM または CEESECI への呼び出し内の input_seconds 値が、対応範囲内にありませんでした。
CEE2EA	3	2506	CEEDATM に渡されたピクチャー・ストリング内に元号 (<JJJ>、<CCCC>、または <CCCCCCCC>) が使用されていましたが、入力された秒数値が対応範囲内にありませんでした。元号を判別できませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EV	2	2527	CEEDATM によって戻されたタイム・スタンプ・ストリングが切り捨てられました。
CEE2F6	1	2534	CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。出力はブランクに設定されました。

使用上の注意: CEEDATM の逆は CEESECS です。CEESECS は、タイム・スタンプを秒数に変換します。

例

CBL LIB

```

*****
**
** Function: CEEDATM - convert seconds to
**                   character timestamp
**
** In this example, a call is made to CEEDATM
** to convert a date represented in Lilian
** seconds (the number of seconds since
** 00:00:00 14 October 1582) to a character
** format (such as 06/02/88 10:23:45). The
** result is displayed.
**
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDATM.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DEST          PIC S9(9) BINARY VALUE 2.
01 SECONDS      COMP-2.
01 IN-DATE.
   02 Vstring-length PIC S9(4) BINARY.
   02 Vstring-text.
   03 Vstring-char  PIC X
                     OCCURS 0 TO 256 TIMES
                     DEPENDING ON Vstring-length
                     of IN-DATE.
01 PICSTR.
   02 Vstring-length PIC S9(4) BINARY.
   02 Vstring-text.
   03 Vstring-char  PIC X
                     OCCURS 0 TO 256 TIMES

```

```

                                DEPENDING ON Vstring-length
                                of PICSTR.
01 TIMESTP                      PIC X(80).
01 FC.
  02 Condition-Token-Value.
  COPY CEEIGZCT.
    03 Case-1-Condition-ID.
      04 Severity    PIC S9(4) COMP.
      04 Msg-No      PIC S9(4) COMP.
    03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) COMP.
      04 Cause-Code PIC S9(4) COMP.
    03 Case-Sev-Ctl  PIC X.
    03 Facility-ID   PIC XXX.
  02 I-S-Info        PIC S9(9) COMP.
*
PROCEDURE DIVISION.
PARA-CBLDATM.
*****
** Call CEESECS to convert timestamp of 6/2/88 **
**   at 10:23:45 AM to Lilian representation **
*****
MOVE 20 TO Vstring-length of IN-DATE.
MOVE '06/02/88 10:23:45 AM'
  TO Vstring-text of IN-DATE.
MOVE 20 TO Vstring-length of PICSTR.
MOVE 'MM/DD/YY HH:MI:SS AP'
  TO Vstring-text of PICSTR.
CALL 'CEESECS' USING IN-DATE, PICSTR,
  SECONDS, FC.

*****
** If CEESECS runs successfully, display result**
*****
IF CEE000 of FC THEN
  DISPLAY Vstring-text of IN-DATE
    ' is Lilian second: ' SECONDS
ELSE
  DISPLAY 'CEESECS failed with msg '
    Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

*****
** Specify desired format of the output.      **
*****
MOVE 35 TO Vstring-length OF PICSTR.
MOVE 'ZD Mmmmmmmmmmmmmmmz YYYY at HH:MI:SS'
  TO Vstring-text OF PICSTR.

*****
** Call CEEDATM to convert Lilian seconds to **
**   a character timestamp                  **
*****
CALL 'CEEDATM' USING SECONDS, PICSTR,
  TIMESTP, FC.

*****
** If CEEDATM runs successfully, display result**
*****
IF CEE000 of FC THEN
  DISPLAY 'Input seconds of ' SECONDS
    ' corresponds to: ' TIMESTP
ELSE
  DISPLAY 'CEEDATM failed with msg '
    Msg-No of FC UPON CONSOLE

```

```

STOP RUN
END-IF.

GOBACK.

```

次の表に、CEEDATM からの出力例を示します。

input_seconds	picture_string	output_timestamp
12,799,191,601.000	YYMMDD HH:MI:SS YY-MM-DD YYMMDDHHMISS YY-MM-DD HH:MI:SS YYYY-MM-DD HH:MI:SS AP	880516 19:00:01 88-05-16 880516190001 88-05-16 19:00:01 1988-05-16 07:00:01 PM
12,799,191,661.986	DD Mmm YY DD MMM YY HH:MM WWW, MMM DD, YYYY ZH:MI AP Wwwwwwwwwz, ZM/ZD/YY HH:MI:SS.99	16 May 88 16 MAY 88 19:01 MON, MAY 16, 1988 7:01 PM Monday, 5/16/88 19:01:01.98
12,799,191,662.009	YYYY YY Y MM ZM RRRR MMM Mmm Mmmmmmmmm Mmmmmmmz DD ZD DDD HH ZH MI SS 99 999 AP WWW Www Wwwwwwwww Wwwwwwwwwz	1988 88 8 05 5 V MAY May May May May 16 16 137 19 19 01 02 00 009 PM MON Mon Monday Monday

609 ページの『例: 日時のピクチャー・ストリング』

関連参照

607 ページの『ピクチャー文字項およびストリング』

CEEDAYS: 日付からリリアン形式への変換

CEEDAYS は、日付を表す文字列をリリアン形式に変換します。リリアン形式では、グレゴリオ暦の開始日 (1582 年 10 月 14 日 金曜日) から数えた日数として日付を表します。

CEEDAYS を COBOL 組み込み関数と併用しないでください。CEECBLDY は、組み込み関数を使用するプログラムに使用します。

CALL CEEDAYS の構文

```
▶▶—CALL—"CEEDAYS"—USING—input_char_date,—picture_string,—————▶▶
▶—output_Lilian_date,—fc.—————▶▶
```

input_char_date (入力)

picture_string の指定に準拠した形式で日付またはタイム・スタンプを表す、ハーフワード長の接頭部の付いた文字列。

文字列に含まれる文字数は 5 から 255 文字です。

input_char_date には、先行または末尾空白を含めることができます。日付の構文解析は、最初の非空白文字から始まります (ピクチャー・ストリング自体に先行空白が含まれる場合は、CEEDAYS がその位置を正確にスキップした後、構文解析が始まります)。

CEEDAYS は、*picture_string* で指定された日付形式によって判別される有効な日付を解析したら、残りの文字をすべて無視します。有効な日付範囲は、1582 年 10 月 15 日から 9999 年 12 月 31 日です。

picture_string (入力)

input_char_date で指定された日付の形式を示す、ハーフワード長の接頭部の付いた文字列。

picture_string 内の各文字は、*input_char_date* 内の文字に対応します。例えば、MMDDYY を *picture_string* として指定すると、CEEDAYS は *input_char_date* の値 060288 を 1988 年 6 月 2 日として読み取ります。

スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、先行ゼロを省略することができます。例えば、次の CEEDAYS の呼び出しは、同じ値 148155 (1988 年 6 月 2 日) をそれぞれ *lildate* に割り当てます。

```
CALL CEEDAYS USING '6/2/88' , 'MM/DD/YY' , lildate , fc.
CALL CEEDAYS USING '06/02/88' , 'MM/DD/YY' , lildate , fc.
CALL CEEDAYS USING '060288' , 'MMDDYY' , lildate , fc.
CALL CEEDAYS USING '88154' , 'YYDDD' , lildate , fc.
```

picture_string にコロンやスラッシュなどの文字 (例: HH:MI:SS YY/MM/DD) が含まれる場合はプレースホルダーと見なされますが、それ以外の場合は無視されます。

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*input_char_date* の YY の位置に、日本元号での年号が入ります。例えば、1988 年は日本の昭和 63 年に相当します。

output_Lilian_date (出力)

リリアン日付 (1582 年 10 月 14 日から数えた日数) を表す 32 ビットの 2 進整数。例えば、1988 年 5 月 16 日は、日数 148138 に相当します。

input_char_date に有効な日付が含まれていない場合は、*output_Lilian_date* が 0 に設定され、CEEDAYS が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_Lilian_date は整数なので、日付計算を容易に行うことができます。うるう年や年末偏差は計算に影響しません。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 81. CEEDAYS のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EB	3	2507	CEEDAYS または CEESECS に渡されたデータが不十分です。リリアン日付の値は計算されませんでした。
CEE2EC	3	2508	CEEDAYS または CEESECS に渡された日付値が無効です。
CEE2ED	3	2509	CEEDAYS または CEESECS に渡された元号が認識されませんでした。
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EO	3	2520	CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。
CEE2EP	3	2521	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。

使用上の注意

- CEEDAYS の逆は CEEDATE です。CEEDATE は、*output_Lilian_date* をリリアン形式から文字形式に変換します。

- 1582 年 10 月 15 日よりも前の日付に対して計算を実行するには、各日付の年号に 4000 を加算し、その日付をリリアン形式に変換してから計算します。計算結果が日数ではなく日付になる場合は、計算結果を日付ストリングに変換し、年号から 4000 を減算します。
- デフォルトでは、2 桁の年号は、システム日付より 80 年前から始まる 100 年間にあります。したがって、2008 年の場合、2 桁の年号はすべて 1928 から 2027 年の範囲内の日付を表します。このデフォルトの範囲を変更するには、CEESCEN 呼び出し可能サービスを使用します。
- *output_Lilian_date* は整数なので、日付計算を容易に行うことができます。うるう年や年末偏差は回避されます。

例

```

CBL LIB
*****
**                                     **
** Function: CEEDAYS - convert date to **
**                   Lilian format   **
**                                     **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDAYS.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CHRDATE.
    02  Vstring-length      PIC S9(4) BINARY.
    02  Vstring-text.
        03  Vstring-char   PIC X
            OCCURS 0 TO 256 TIMES
            DEPENDING ON Vstring-length
            of CHRDATE.
01  PICSTR.
    02  Vstring-length      PIC S9(4) BINARY.
    02  Vstring-text.
        03  Vstring-char   PIC X
            OCCURS 0 TO 256 TIMES
            DEPENDING ON Vstring-length
            of PICSTR.
01  LILIAN                  PIC S9(9) BINARY.
01  FC.
    02  Condition-Token-Value.
COPY CEEIGZCT.
    03  Case-1-Condition-ID.
        04  Severity      PIC S9(4) COMP.
        04  Msg-No       PIC S9(4) COMP.
    03  Case-2-Condition-ID
        REDEFINES Case-1-Condition-ID.
        04  Class-Code   PIC S9(4) COMP.
        04  Cause-Code  PIC S9(4) COMP.
    03  Case-Sev-Ctl     PIC X.
    03  Facility-ID     PIC XXX.
    02  I-S-Info        PIC S9(9) COMP.
*
PROCEDURE DIVISION.
PARA-CBLDAYS.
*****
** Specify input date and length      **
*****
        MOVE 16 TO Vstring-length of CHRDATE.
        MOVE '1 January 2005'
          TO Vstring-text of CHRDATE.
*****

```

```

** Specify a picture string that describes      **
** input date, and the picture string's length.**
*****
MOVE 25 TO Vstring-length of PICSTR.
MOVE 'ZD Mmmmmmmmmmmmmz YYYY'
TO Vstring-text of PICSTR.

*****
** Call CEEDAYS to convert input date to a      **
** Lilian date                                 **
*****
CALL 'CEEDAYS' USING CHRDATE, PICSTR,
LILIAN, FC.

*****
** If CEEDAYS runs successfully, display result**
*****
IF CEE000 of FC THEN
    DISPLAY Vstring-text of CHRDATE
    ' is Lilian day: ' LILIAN
ELSE
    DISPLAY 'CEEDAYS failed with msg '
    Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.

GOBACK.

```

609 ページの『例: 日時のピクチャー・ストリング』

関連参照

607 ページの『ピクチャー文字項およびストリング』

CEEDYWK: リリアン日付からの曜日の計算

CEEDYWK は、リリアン日付の曜日を 1 から 7 までの数字として計算します。

CEEDYWK から戻される数値から、曜日を計算することができます。

CALL CEEDYWK の構文

```
▶▶—CALL—"CEEDYWK"—USING—input_Lilian_date,—output_day_no,—fc.—▶▶
```

input_Lilian_date (入力)

リリアン日付 (1582 年 10 月 14 日から数えた日数) を表す 32 ビットの 2 進整数。

例えば、1988 年 5 月 16 日は、日数 148138 に相当します。

input_Lilian_date の有効範囲は 1 から 3,074,324 (1582 年 10 月 15 日から 9999 年 12 月 31 日) です。

output_day_no (出力)

input_Lilian_date の曜日を表す 32 ビットの 2 進整数 (1 = 日曜日、2 = 月曜日、... 7 = 土曜日)。

input_Lilian_date が無効な場合は、*output_day_no* が 0 に設定され、*CEEDYWK* が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 82. *CEEDYWK* のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EG	3	2512	CEEDATE または CEEDYWK への呼び出しで渡されたリリアン日付値が、対応範囲内にありませんでした。

例

CBL LIB

```

*****
**                                     **
** Function: Call CEEDYWK to calculate the **
**           day of the week from Lilian date **
**                                     **
** In this example, a call is made to CEEDYWK **
** to return the day of the week on which a **
** Lilian date falls. (A Lilian date is the **
** number of days since 14 October 1582) **
**                                     **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLDYWK.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LILIAN                PIC S9(9) BINARY.
01 DAYNUM                PIC S9(9) BINARY.
01 IN-DATE.
   02 Vstring-length    PIC S9(4) BINARY.
   02 Vstring-text.
   03 Vstring-char      PIC X,
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Vstring-length
                        of IN-DATE.
01 PICSTR.
   02 Vstring-length    PIC S9(4) BINARY.
   02 Vstring-text.
   03 Vstring-char      PIC X,
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Vstring-length
                        of PICSTR.
01 FC.
   02 Condition-Token-Value.
   COPY CEEIGZCT.
   03 Case-1-Condition-ID.
   04 Severity          PIC S9(4) COMP.
   04 Msg-No            PIC S9(4) COMP.
   03 Case-2-Condition-ID
                        REDEFINES Case-1-Condition-ID.
   04 Class-Code        PIC S9(4) COMP.

```

```

          04 Cause-Code PIC S9(4) COMP.
          03 Case-Sev-Ctl PIC X.
          03 Facility-ID PIC XXX.
02 I-S-Info          PIC S9(9) COMP.

```

PROCEDURE DIVISION.

PARA-CBLDAYS.

```

** Call CEEDAYS to convert date of 6/2/88 to
** Lilian representation
MOVE 6 TO Vstring-length of IN-DATE.
MOVE '6/2/88' TO Vstring-text of IN-DATE(1:6).
MOVE 8 TO Vstring-length of PICSTR.
MOVE 'MM/DD/YY' TO Vstring-text of PICSTR(1:8).
CALL 'CEEDAYS' USING IN-DATE, PICSTR,
LILIAN, FC.

```

```

** If CEEDAYS runs successfully, display result.
IF CEE000 of FC THEN
    DISPLAY Vstring-text of IN-DATE
    ' is Lilian day: ' LILIAN
ELSE
    DISPLAY 'CEEDAYS failed with msg '
    Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.

```

PARA-CBLDYWK.

```

** Call CEEDYWK to return the day of the week on
** which the Lilian date falls
CALL 'CEEDYWK' USING LILIAN , DAYNUM , FC.

```

```

** If CEEDYWK runs successfully, print results
IF CEE000 of FC THEN
    DISPLAY 'Lilian day ' LILIAN
    ' falls on day ' DAYNUM
    ' of the week, which is a:'

```

```

** Select DAYNUM to display the name of the day
** of the week.

```

```

EVALUATE DAYNUM
    WHEN 1
        DISPLAY 'Sunday.'
    WHEN 2
        DISPLAY 'Monday.'
    WHEN 3
        DISPLAY 'Tuesday'
    WHEN 4
        DISPLAY 'Wednesday.'
    WHEN 5
        DISPLAY 'Thursday.'
    WHEN 6
        DISPLAY 'Friday.'
    WHEN 7
        DISPLAY 'Saturday.'
END-EVALUATE ELSE
    DISPLAY 'CEEDYWK failed with msg '
    Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.

```

GOBACK.

CEEGMT: 現在のグリニッジ標準時の取得

CEEGMT は、現在のグリニッジ標準時 (GMT) をリリアン日、および 1582 年 10 月 14 日 00:00:00 以降の秒数として戻します。戻り値は、他の日付および時刻の呼び出し可能サービスによって生成および使用される値と互換性があります。

ALL CEEGMT の構文

```
▶▶—CALL—"CEEGMT"—USING—output_GMT_Lilian,—output_GMT_seconds,—fc.—▶▶
```

output_GMT_Lilian (出力)

グリニッジ (イングランド) の現在の日付をリリアン形式 (1582 年 10 月 14 日から数えた日数) で表す 32 ビットの 2 進整数。

例えば、1988 年 5 月 16 日は、日数 148138 に相当します。システムから GMT を使用できない場合は、*output_GMT_Lilian* が 0 に設定され、CEEGMT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_GMT_seconds (出力)

グリニッジ (イングランド) の現在の日時を 1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数で表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 ($24 \times 60 \times 60 + 01$) に相当します。1988 年 5 月 16 日の 19:00:01.078 は、秒数 12,799,191,601.078 に相当します。システムから GMT を使用できない場合は、*output_GMT_seconds* が 0 に設定され、CEEGMT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 83. CEEGMT のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2E6	3	2502	システムから UTC/GMT を使用できませんでした。

使用上の注意

- CEEDATE は *output_GMT_Lilian* を文字日付に変換し、CEEDATM は *output_GMT_seconds* を文字タイム・スタンプに変換します。

- このサービスから意味のある結果を得るには、システムのクロックを現地時間に設定し、環境変数 TZ を正しく設定する必要があります。
- CEEGMT から戻される値から、経過時間を計算することができます。例えば、CEEGMT への呼び出しが行われてから、次に同じ呼び出しが行われるまでの経過時間を計算するには、2 つの戻り値の差を計算します。
- CEEUTC はこのサービスと同じです。

例

```

CBL LIB
*****
**                                     **
** Function: Call CEEGMT to get current **
**           Greenwich Mean Time      **
**                                     **
** In this example, a call is made to CEEGMT **
** to return the current GMT as a Lilian date **
** and as Lilian seconds. The results are **
** displayed.                          **
**                                     **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. IGZTGMT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LILIAN          PIC S9(9) BINARY.
01 SECS           COMP-2.
01 FC.
02 Condition-Token-Value.
COPY CEEIGZCT.
03 Case-1-Condition-ID.
04 Severity      PIC S9(4) COMP.
04 Msg-No       PIC S9(4) COMP.
03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
04 Class-Code   PIC S9(4) COMP.
04 Cause-Code  PIC S9(4) COMP.
03 Case-Sev-Ctl PIC X.
03 Facility-ID  PIC XXX.
02 I-S-Info     PIC S9(9) COMP.
PROCEDURE DIVISION.
PARA-CBLGMT.
    CALL 'CEEGMT' USING LILIAN , SECS , FC.

    IF CEE000 of FC THEN
        DISPLAY 'The current GMT is also '
            'known as Lilian day: ' LILIAN
        DISPLAY 'The current GMT in Lilian '
            'seconds is: ' SECS
    ELSE
        DISPLAY 'CEEGMT failed with msg '
            Msg-No of FC UPON CONSOLE
        STOP RUN
    END-IF.

GOBACK.

```

関連タスク

225 ページの『環境変数の設定』

CEEGMTO: グリニッジ標準時から現地時間までのオフセットの取得

CEEGMTO は、ローカル・システムの時刻とグリニッジ標準時 (GMT) の差を表す値を呼び出しルーチンに戻します。

CALL CEEGMTO の構文

```
▶▶CALL "CEEGMTO" USING offset_hours, offset_minutes, offset_seconds, ▶▶
▶fc.▶▶
```

offset_hours (出力)

GMT から現地時間までのオフセットを時間単位で表す 32 ビットの 2 進整数。

例えば太平洋標準時の場合、*offset_hours* は -8 に相当します。

offset_hours の範囲は -12 から +13 (+13 = +12 の時間帯における夏時間調整) です。

現地時間のオフセットを使用できない場合は、*offset_hours* が 0 になり、CEEGMTO が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

offset_minutes (出力)

現地時間が GMT よりも何分進んでいるか、または何分遅れているかを表す、32 ビットの 2 進整数。

offset_minutes の範囲は 0 から 59 です。

現地時間のオフセットを使用できない場合は、*offset_minutes* が 0 になり、CEEGMTO が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

offset_seconds (出力)

GMT から現地時間までのオフセットを秒単位で表す 64 ビット長の浮動小数点数。

例えば、太平洋標準時は GMT よりも 8 時間遅れています。現地時間が標準時で太平洋標準時間帯に属する場合、CEEGMTO は -28,800 (-8 * 60 * 60) を戻します。*offset_seconds* の範囲は -43,200 から +46,800 です。

offset_seconds を CEEGMT で使用すると、現地日時を計算することができます。

システムから現地時間のオフセットを使用できない場合は、*offset_seconds* が 0 に設定され、CEEGMTO が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 84. CEEGMTO のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2E7	3	2503	UTC/GMT から現地時間までのオフセットをシステムから使用できませんでした。

使用上の注意

- CEEDATM は *offset_seconds* を文字タイム・スタンプに変換します。
- このサービスから意味のある結果を得るには、システムのクロックを現地時間に設定し、環境変数 TZ を正しく設定する必要があります。

例

```

CBL LIB
*****
**                                                                    **
** Function: Call CEEGMTO to get offset from Greenwich Mean Time to local time **
**                                                                    **
** In this example, a call is made to CEEGMTO to return the offset from GMT to local time **
** as separate binary integers representing offset hours, minutes, and seconds. The **
** results are displayed. **
**                                                                    **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. IGTGMTO.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 HOURS          PIC S9(9) BINARY.
01 MINUTES        PIC S9(9) BINARY.
01 SECONDS COMP-2.
01 FC.
02 Condition-Token-Value.
COPY CEEIGZCT.
03 Case-1-Condition-ID.
04 Severity      PIC S9(4) COMP.
04 Msg-No        PIC S9(4) COMP.
03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
04 Class-Code    PIC S9(4) COMP.
04 Cause-Code    PIC S9(4) COMP.
03 Case-Sev-Ctl  PIC X.
03 Facility-ID   PIC XXX.
02 I-S-Info      PIC S9(9) COMP.
PROCEDURE DIVISION.
PARA-CBLGMTO.
CALL 'CEEGMTO' USING HOURS , MINUTES ,
SECONDS , FC.

IF CEE000 of FC THEN
    DISPLAY 'Local time differs from GMT '
           'by: ' HOURS ' hours, '
           MINUTES ' minutes, OR '
           SECONDS ' seconds. '

```



```

ELSE
  DISPLAY 'CEEGMT0 failed with msg '
    Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

GOBACK.

```

関連タスク

225 ページの『環境変数の設定』

関連参照

669 ページの『CEEGMT: 現在のグリニッジ標準時の取得』

228 ページの『ランタイム環境変数』

CEEISEC: 整数から秒への変換

CEEISEC は、年、月、日、時、分、秒、ミリ秒を表す 2 進整数を、1582 年 10 月 14 日の 00:00:00 から数えた秒数を表す数値に変換します。

CALL CEEISEC の構文

```

▶▶—CALL—"CEEISEC"—USING—input_year,—input_months,—input_day,————▶
▶—input_hours,—input_minutes,—input_seconds,—input_milliseconds,————▶
▶—output_seconds,—fc.————▶▶

```

input_year (入力)

年を表す 32 ビットの 2 進整数。

input_year の有効な値範囲は 1582 から 9999 です。

input_month (入力)

月を表す 32 ビットの 2 進整数。

input_month の有効な値範囲は 1 から 12 です。

input_day (入力)

日を表す 32 ビットの 2 進整数。

input_day の有効な値範囲は 1 から 31 です。

input_hours (入力)

時を表す 32 ビットの 2 進整数。

input_hours の有効な値範囲は 0 から 23 です。

input_minutes (入力)

分を表す 32 ビットの 2 進整数。

input_minutes の有効な値範囲は 0 から 59 です。

input_seconds (入力)

秒を表す 32 ビットの 2 進整数。

input_seconds の有効な値範囲は 0 から 59 です。

input_milliseconds (入力)

ミリ秒を表す 32 ビットの 2 進整数。

input_milliseconds の有効な値範囲は 0 から 999 です。

output_seconds (出力)

1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 ($24 \times 60 \times 60 + 01$) に相当します。*output_seconds* の有効範囲は 86,400 から 265,621,679,999.999 (9999 年 12 月 31 日の 23:59:59.999) です。

入力値が無効な場合は、*output_seconds* が 0 に設定されます。

output_seconds をリリアン日数に変換するには、`divide output_seconds` を 86,400 (1 日分の秒数) で除算します。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 85. CEEISEC のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EE	3	2510	CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。
CEE2EF	3	2511	CEEISEC の呼び出しで渡された日のパラメーターが、指定された年および月に対して無効です。
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EI	3	2514	CEEISEC の呼び出しで渡された年の値が、対応範囲内にありませんでした。
CEE2EJ	3	2515	CEEISEC 呼び出し内のミリ秒の値が認識されませんでした。
CEE2EK	3	2516	CEEISEC 呼び出し内の分の値が認識されませんでした。
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EN	3	2519	CEEISEC 呼び出し内の秒の値が認識されませんでした。

使用上の注意: CEEISEC の逆は CEESECI です。CEESECI は、秒数を整数の年、月、日、時、分、秒、ミリ秒に変換します。

例

CBL LIB

```

*****
**                                                                 **
** Function: Call CEEISEC to convert integers **
** to seconds                                                                 **
**                                                                 **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLISEC.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 YEAR PIC S9(9) BINARY.
01 MONTH PIC S9(9) BINARY.
01 DAYS PIC S9(9) BINARY.
01 HOURS PIC S9(9) BINARY.
01 MINUTES PIC S9(9) BINARY.
01 SECONDS PIC S9(9) BINARY.
01 MILLSEC PIC S9(9) BINARY.
01 OUTSECS COMP-2.
01 FC.
02 Condition-Token-Value.
COPY CEEIGZCT.
03 Case-1-Condition-ID.
04 Severity PIC S9(4) COMP.
04 Msg-No PIC S9(4) COMP.
03 Case-2-Condition-ID
REDEFINES Case-1-Condition-ID.
04 Class-Code PIC S9(4) COMP.
04 Cause-Code PIC S9(4) COMP.
03 Case-Sev-Ctl PIC X.
03 Facility-ID PIC XXX.
02 I-S-Info PIC S9(9) COMP.
PROCEDURE DIVISION.
PARA-CBLISEC.
*****
** Specify seven binary integers representing **
** the date and time as input to be converted **
** to Lilian seconds **
*****
MOVE 2000 TO YEAR.
MOVE 1 TO MONTH.
MOVE 1 TO DAYS.
MOVE 0 TO HOURS.
MOVE 0 TO MINUTES.
MOVE 0 TO SECONDS.
MOVE 0 TO MILLSEC.
*****
** Call CEEISEC to convert the integers **
** to seconds **
*****
CALL 'CEEISEC' USING YEAR, MONTH, DAYS,
HOURS, MINUTES, SECONDS,
MILLSEC, OUTSECS , FC.
*****
** If CEEISEC runs successfully, display result**
*****
IF CEE000 of FC THEN
DISPLAY MONTH '/' DAYS '/' YEAR
' AT ' HOURS ':' MINUTES ':' SECONDS
' is equivalent to ' OUTSECS ' seconds'
ELSE
DISPLAY 'CEEISEC failed with msg '
Msg-No of FC UPON CONSOLE

```

```
STOP RUN
END-IF.

GOBACK.
```

CEELOCT: 現在の現地日時の取得

CEELOCT は、現在の現地日時をリリアン日付 (1582 年 10 月 14 日から数えた日数)、リリアン秒数 (1582 年 10 月 14 日の 00:00:00 から数えた秒数)、グレゴリオ文字ストリング (YYYYMMDDHHMISS999) としてそれぞれ返します。

これらの値は、他の日付および時刻の呼び出し可能サービスや、既存の組み込み関数と互換性があります。

CEELOCT は、CEEGMT、CEEGMT0、および CEEDATM の各サービスを個別に呼び出すのと同じ機能を実行します。ただし、CEELOCT の呼び出しの方が、はるかに高速です。

CALL CELOCT の構文

```
▶▶CALL"CEELOCT"USINGoutput_Lilian,output_seconds,
▶output_Gregorian,fc.▶▶
```

output_Lilian (出力)

現在の現地日付をリリアン形式 (1 日目は 1582 年 10 月 15 日、148,887 日目は 1990 年 6 月 4 日) で表す、32 ビットの 2 進整数。

システムから現地時間を使用できない場合は、*output_Lilian* が 0 に設定され、CEELOCT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_seconds (出力)

現在の現地日時を 1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数で表す、64 ビット長の浮動小数点数。例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 ($24 \times 60 \times 60 + 01$) に相当します。1990 年 6 月 4 日の 19:00:01.078 は、秒数 12,863,905,201.078 に相当します。

システムから現地時間を使用できない場合は、*output_seconds* が 0 に設定され、CEELOCT が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_Gregorian (出力)

現地の年、月、日、時、分、秒、ミリ秒を表す、YYYYMMDDHHMISS999 形式の 17 バイト固定長文字ストリング。

output_Gregorian の形式が必要な形式と合わない場合は、CEEDATM 呼び出し可能サービスを使用して、*output_seconds* を別の形式に変換することができます。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 86. CEELCOT のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2F3	3	2531	システムから現地時間を使用できませんでした。

使用上の注意

- CEEGMT 呼び出し可能サービスを使用して、グリニッジ標準時 (GMT) を判断することができます。
- CEEGMT0 呼び出し可能サービスを使用して、GMT から現地時間までのオフセットを取得することができます。
- CEELCOT により戻される文字値は、既存の組み込み関数から生成される文字値と一致するようになっています。戻される数値を使用して、日付計算を単純化することができます。

例

CBL LIB

```
*****
**
** Function: Call CEELCOT to get current
**          local time
**
** In this example, a call is made to CEELCOT
** to return the current local time in Lillian
** days (the number of days since 14 October
** 1582), Lillian seconds (the number of
** seconds since 00:00:00 14 October 1582),
** and a Gregorian string (in the form
** YYYYMMDDMISS999). The Gregorian character
** string is then displayed.
**
**
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLCOT.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 LILIAN          PIC S9(9) BINARY.
01 SECONDS        COMP-2.
01 GREGORN        PIC X(17).
01 FC.
02 Condition-Token-Value.
COPY CEEIGZCT.
03 Case-1-Condition-ID.
04 Severity      PIC S9(4) COMP.
04 Msg-No       PIC S9(4) COMP.
03 Case-2-Condition-ID
   REDEFINES Case-1-Condition-ID.
04 Class-Code   PIC S9(4) COMP.
04 Cause-Code  PIC S9(4) COMP.
03 Case-Sev-Ctl PIC X.
03 Facility-ID  PIC XXX.
```

```

02 I-S-Info          PIC S9(9) COMP.
PROCEDURE DIVISION.
PARA-CBLLOCT.
    CALL 'CEELOCT' USING LILIAN, SECONDS,
                        GREGORN, FC.
*****
** If CEELOCT runs successfully, display    **
**   Gregorian character string            **
*****
    IF CEE000 of FC THEN
        DISPLAY 'Local Time is ' GREGORN
    ELSE
        DISPLAY 'CEELOCT failed with msg '
                Msg-No of FC UPON CONSOLE
        STOP RUN
    END-IF.

GOBACK.

```

CEEQCEN: 世紀ウィンドウの照会

CEEQCEN は、2 桁の年号値の世紀ウィンドウを照会します。

世紀ウィンドウを変更する場合は、CEEQCEN を使用して設定を取得した後、CEESCEN を使用して現行の設定を保存して復元します。

CALL CEEQCEN の構文

```

▶▶—CALL—"CEEQCEN"—USING—century_start,—fc.—▶▶

```

century_start (出力)

世紀ウィンドウの基になる年を表す、0 から 100 の整数。

例えば、日付および時刻の呼び出し可能サービスのデフォルトが有効な場合、2 桁の年号はすべて、システム日付より 80 年前から始まる 100 年間に属します。この後、CEEQCEN が値 80 を戻します。例えば、2008 年の場合、80 は、すべての 2 桁年号が 100 年間 (1928 年から 2027 年まで) にあることを示します。

fc (出力)

このサービスの結果を示す 12 バイトのフィールドバック・コード (オプション)。

表 87. CEEQCEN のシンボリック条件

シンボリック・フィールドバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。

例

```

CBL LIB
*****
**
** Function: Call CEEQCEN to query the      **
**         日付および時刻 callable services **
**         century window                  **
**
** In this example, CEEQCEN is called to query **
** the date at which the century window starts **
** The century window is the 100-year window **
** within which the 日付および時刻 callable **
** services assume all two-digit years lie.   **
**
**
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLQCEN.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 STARTCW          PIC S9(9) BINARY.
01 FC.
02 Condition-Token-Value.
COPY CEEIGZCT.
03 Case-1-Condition-ID.
04 Severity        PIC S9(4) COMP.
04 Msg-No          PIC S9(4) COMP.
03 Case-2-Condition-ID
    REDEFINES Case-1-Condition-ID.
04 Class-Code     PIC S9(4) COMP.
04 Cause-Code     PIC S9(4) COMP.
03 Case-Sev-Ctl   PIC X.
03 Facility-ID    PIC XXX.
02 I-S-Info       PIC S9(9) COMP.
PROCEDURE DIVISION.

PARA-CBLQCEN.
*****
** Call CEEQCEN to return the start of the  **
**         century window                    **
*****

    CALL 'CEEQCEN' USING STARTCW, FC.
*****
** CEEQCEN has no nonzero feedback codes to  **
** check, so just display result.            **
**
*****
IF CEE000 of FC THEN
    DISPLAY 'The start of the century '
           'window is: ' STARTCW
ELSE
    DISPLAY 'CEEQCEN failed with msg '
           'Msg-No of FC UPON CONSOLE'
    STOP RUN
END-IF.

GOBACK.

```

CEESCEN: 世紀ウィンドウの設定

CEESCEN は、世紀ウィンドウを、他の日付および時刻の呼び出し可能サービスが使用できる 2 桁年号値に設定します。

次のような場合は、CEEDAYS または CEESECS と組み合わせて CEESCEN を使用します。

- 2桁の年号を含む日付値を処理する場合 (YYMMDD 形式など)
- デフォルトの世紀間隔が特定のアプリケーションの要件に合わない場合

世紀ウィンドウを照会するには、CEEQCEN を使用します。

CALL CEESCEN の構文

```
▶▶—CALL—“CEESCEN”—USING—century_start,—fc.—▶▶
```

century_start

世紀ウィンドウを設定する、0 から 100 の整数。

例えば、値 80 の場合は、すべての 2 桁年桁が、システム日付より 80 年前から始まる 100 年間に属します。したがって、2008 年の場合、2 桁の年号はすべて 1928 から 2027 年の範囲内の日付を表すものと想定されます。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 88. CEESCEN のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2E6	3	2502	システムから UTC/GMT を使用できませんでした。
CEE2F5	3	2533	CEESCEN に渡された値が 0 から 100 の範囲内にありませんでした。

例

CBL LIB

```
*****
**
** Function: Call CEESCEN to set the          **
**   日付および時刻 callable services      **
**   century window                         **
**
** In this example, CEESCEN is called to change **
** the start of the century window to 30 years **
** before the system date. CEEQCEN is then   **
** called to query that the change made. A    **
** message that this has been done is then   **
** displayed.                                **
**
**
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLSCEN.
```



```

DATA DIVISION.
WORKING-STORAGE SECTION.
01 STARTCW          PIC S9(9) BINARY.
01 FC.
  02 Condition-Token-Value.
  COPY CEEIGZCT.
    03 Case-1-Condition-ID.
    04 Severity     PIC S9(4) COMP.
    04 Msg-No       PIC S9(4) COMP.
    03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
    04 Class-Code   PIC S9(4) COMP.
    04 Cause-Code   PIC S9(4) COMP.
    03 Case-Sev-Ctl PIC X.
    03 Facility-ID  PIC XXX.
  02 I-S-Info       PIC S9(9) COMP.
PROCEDURE DIVISION.
PARA-CBLSCEN.
*****
** Specify 30 as century start, and two-digit
**   years will be assumed to lie in the
**   100-year window starting 30 years before
**   the system date.
*****
      MOVE 30 TO STARTCW.

*****
** Call CEEScen to change the start of the century
**   window.
*****
      CALL 'CEEScen' USING STARTCW, FC.
      IF NOT CEE000 of FC THEN
        DISPLAY 'CEEScen failed with msg '
          Msg-No of FC UPON CONSOLE
      STOP RUN
      END-IF.

PARA-CBLQCEN.
*****
** Call CEEQCEN to return the start of the century
**   window
*****
      CALL 'CEEQCEN' USING STARTCW, FC.

*****
** CEEQCEN has no nonzero feedback codes to
**   check, so just display result.
*****
      DISPLAY 'The start of the century '
        'window is: ' STARTCW
      GOBACK.

```

CEESECI: 秒から整数への変換

CEESECI は、1582 年 10 月 14 日の 00:00:00 から数えた秒数を表す数値を、年、月、日、時、分、秒、ミリ秒を表す 2 進整数に変換します。

文字形式ではなく数値形式の出力が必要な場合は、CEEDATM ではなく CEESECI を使用します。

CALL CEESECI の構文

```
▶▶—CALL—“CEESECI”—USING—input_seconds,—output_year,—output_month,—————▶  
▶—output_day,—output_hours,—output_minutes,—output_seconds,—————▶  
▶—output_milliseconds,—fc.—————▶▶
```

input_seconds

1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。

例えば、1582 年 10 月 15 日の 00:00:01 は秒数 86,401 ($24*60*60 + 01$) に相当します。*input_seconds* の有効な値範囲は 86,400 から 265,621,679,999.999 (9999 年 12 月 31 日の 23:59:59.999) です。

input_seconds が無効な場合は、フィードバック・コードを除くすべての出力パラメーターが 0 に設定されます。

output_year (出力)

年を表す 32 ビットの 2 進整数。

output_year の有効な値範囲は 1582 から 9999 です。

output_month (出力)

月を表す 32 ビットの 2 進整数。

output_month の有効な値範囲は 1 から 12 です。

output_day (出力)

日を表す 32 ビットの 2 進整数。

output_day の有効な値範囲は 1 から 31 です。

output_hours (出力)

時を表す 32 ビットの 2 進整数。

output_hours の有効な値範囲は 0 から 23 です。

output_minutes (出力)

分を表す 32 ビットの 2 進整数。

output_minutes の有効な値範囲は 0 から 59 です。

output_seconds (出力)

秒を表す 32 ビットの 2 進整数。

output_seconds の有効な値範囲は 0 から 59 です。

output_milliseconds (出力)

ミリ秒を表す 32 ビットの 2 進整数。

output_milliseconds の有効な値範囲は 0 から 999 です。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 89. CEESECI のシンボリック条件

シンボリック・フィールドバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2E9	3	2505	CEEDATM または CEESECI への呼び出し内の input_seconds 値が、対応範囲内にありませんでした。

使用上の注意

- CEESECI の逆は CEEISEC です。CEEISEC は、年、月、日、時、分、秒、ミリ秒を表す個々の 2 進整数を秒数に変換します。
- 入力値が秒ではなくリリアン日付の場合は、リリアン日付に 86,400 (1 日分の秒数) を乗算してから、新しい値を CEESECI に渡します。

例

CBL LIB

```

*****
**
** Function: Call CEESECI to convert seconds
**           to integers
**
** In this example a call is made to CEESECI
** to convert a number representing the number
** of seconds since 00:00:00 14 October 1582
** to seven binary integers representing year,
** month, day, hour, minute, second, and
** millisecond. The results are displayed in
** this example.
**
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLSECI.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 INSECS                COMP-2.
01 YEAR                 PIC S9(9) BINARY.
01 MONTH                PIC S9(9) BINARY.
01 DAYS                 PIC S9(9) BINARY.
01 HOURS                PIC S9(9) BINARY.
01 MINUTES              PIC S9(9) BINARY.
01 SECONDS              PIC S9(9) BINARY.
01 MILLSEC              PIC S9(9) BINARY.
01 IN-DATE.
   02 Vstring-length     PIC S9(4) BINARY.
   02 Vstring-text.
     03 Vstring-char     PIC X,
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Vstring-length
                        of IN-DATE.
01 PICSTR.
   02 Vstring-length     PIC S9(4) BINARY.
   02 Vstring-text.
     03 Vstring-char     PIC X,
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Vstring-length
                        of PICSTR.

```

```

01 FC.
  02 Condition-Token-Value.
  COPY CEEIGZCT.
    03 Case-1-Condition-ID.
      04 Severity PIC S9(4) COMP.
      04 Msg-No PIC S9(4) COMP.
    03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code PIC S9(4) COMP.
      04 Cause-Code PIC S9(4) COMP.
    03 Case-Sev-Ctl PIC X.
    03 Facility-ID PIC XXX.
  02 I-S-Info PIC S9(9) COMP.
PROCEDURE DIVISION.
PARA-CBLSECS.
*****
** Call CEESECS to convert timestamp of 6/2/88
** at 10:23:45 AM to Lilian representation
*****
MOVE 20 TO Vstring-length of IN-DATE.
MOVE '06/02/88 10:23:45 AM'
    TO Vstring-text of IN-DATE.
MOVE 20 TO Vstring-length of PICSTR.
MOVE 'MM/DD/YY HH:MI:SS AP'
    TO Vstring-text of PICSTR.
CALL 'CEESECS' USING IN-DATE, PICSTR,
    INSECS, FC.
IF NOT CEE000 of FC THEN
    DISPLAY 'CEESECS failed with msg '
        Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.

PARA-CBLSECI.
*****
** Call CEESECI to convert seconds to integers
*****
CALL 'CEESECI' USING INSECS, YEAR, MONTH,
    DAYS, HOURS, MINUTES,
    SECONDS, MILLSEC, FC.
*****
** If CEESECI runs successfully, display results
*****
IF CEE000 of FC THEN
    DISPLAY 'Input seconds of ' INSECS
        ' represents:'
        DISPLAY ' Year..... ' YEAR
    DISPLAY ' Month..... ' MONTH
    DISPLAY ' Day..... ' DAYS
    DISPLAY ' Hour..... ' HOURS
    DISPLAY ' Minute..... ' MINUTES
    DISPLAY ' Second..... ' SECONDS
    DISPLAY ' Millisecond.. ' MILLSEC
ELSE
    DISPLAY 'CEESECI failed with msg '
        Msg-No of FC UPON CONSOLE
    STOP RUN
END-IF.

GOBACK.

```

CEESECS: タイム・スタンプの秒への変換

CEESECS は、タイム・スタンプを表すストリングをリリアン秒 (1582 年 10 月 14 日の 00:00:00 から数えた秒数) に変換します。このサービスを使用すると、2 つのタイム・スタンプ間の経過時間を計算するなどの時間演算が容易になります。

CALL CEESECS の構文

```
▶▶—CALL—“CEESECS”—USING—input_timestamp,—picture_string,————▶▶  
▶—output_seconds,—fc.————▶▶
```

input_timestamp (入力)

picture_string の指定と一致した形式で日付またはタイム・スタンプを表す、ハーフワード長の接頭部の付いた文字ストリング。

文字ストリングに含まれる文字数は 5 から 80 ピクチャー文字です。
input_timestamp には、先行または末尾ブランクを含めることができます。構文解析は、最初の非ブランク文字から始まります (ピクチャー・ストリング自体に先行ブランクが含まれる場合は、CEESECS がその位置を正確にスキップした後、構文解析が始まります)。

picture_string で指定された日付形式によって判別される有効な日付を解析したら、CEESECS は残りの文字をすべて無視します。有効な日付範囲は、1582 年 10 月 15 日から 9999 年 12 月 31 日です。完全な日付を指定する必要があります。有効な時刻範囲は 00:00:00.000 から 23:59:59.999 です。

時刻値の一部または全部を省略すると、残りの値には 0 が代入されます。以下に例を示します。

```
1992-05-17-19:02 is equivalent to 1992-05-17-19:02:00  
1992-05-17      is equivalent to 1992-05-17-00:00:00
```

picture_string (入力)

input_timestamp で指定された日付またはタイム・スタンプ値の形式を示す、ハーフワード長の接頭部の付いた文字ストリング。

picture_string 内の各文字は、*input_timestamp* 内の文字を表します。例えば、MMDDYY HH.MI.SS を *picture_string* として指定すると、CEESECS は *input_char_date* の値 060288 15.35.02 を 1988 年 6 月 2 日の 3:35:02 PM として読み取ります。スラッシュ (/) などの区切り文字がピクチャー・ストリング内にある場合は、先行ゼロを省略することができます。例えば、次に示す CEESECS への呼び出しはすべて、同じ値をデータ項目 *secs* に割り当てます。

```
CALL CEESECS USING '92/06/03 15.35.03',  
                  'YY/MM/DD HH.MI.SS', secs, fc.  
CALL CEESECS USING '92/6/3 15.35.03',  
                  'YY/MM/DD HH.MI.SS', secs, fc.  
CALL CEESECS USING '92/6/3 3.35.03 PM',
```

```
'YY/MM/DD HH.MI.SS AP', secs, fc.
CALL CEESECS USING '92.155 3.35.03 pm',
'YY.DDD HH.MI.SS AP', secs, fc.
```

picture_string に日本元号のシンボル <JJJJ> が含まれる場合は、*input_timestamp* の YY の位置に、日本元号での年号が入ります。例えば、1988 年は日本の昭和 63 年に相当します。

output_seconds (出力)

1582 年 10 月 14 日の 00:00:00 から数えた (うるう秒は数えない) 秒数を表す、64 ビット長の浮動小数点数。例えば、1582 年 10 月 15 日の 00:00:01 は、リリアン形式の秒数 86,401 (24*60*60 + 01) に相当します。1988 年 5 月 16 日の 19:00:01.12 は、秒数 12,799,191,601.12 に相当します。

表現される最大値は 9999 年 12 月 31 日の 23:59:59.999 です。これは、リリアン形式では秒数 265,621,679,999.999 に相当します。

64 ビット長の浮動小数点値は、精度を失うことなく約 16 桁の有効小数桁数を正確に表現することができます。このため、最も近いミリ秒 (15 桁の小数桁数) を正確に使用することができます。

input_timestamp に有効な日付またはタイム・スタンプが含まれていない場合は、*output_seconds* が 0 に設定され、CEESECS が終了して非 CEE000 シンボリック・フィードバック・コードが戻されます。

output_seconds は経過時間を表すので、経過時間の計算を容易に行うことができます。うるう年や年末偏差は計算に影響しません。

fc (出力)

このサービスの結果を示す 12 バイトのフィードバック・コード (オプション)。

表 90. CEESECS のシンボリック条件

シンボリック・フィードバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE000	0	--	サービスが正しく完了しました。
CEE2EB	3	2507	CEEDAYS または CEESECS に渡されたデータが不十分です。リリアン日付の値は計算されませんでした。
CEE2EC	3	2508	CEEDAYS または CEESECS に渡された日付値が無効です。
CEE2ED	3	2509	CEEDAYS または CEESECS に渡された元号が認識されませんでした。
CEE2EE	3	2510	CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。
CEE2EH	3	2513	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
CEE2EK	3	2516	CEEISEC 呼び出し内の分の値が認識されませんでした。

表 90. CEESECS のシンボリック条件 (続き)

シンボリック・フィールドバック・コード	重大度	メッセージ番号	メッセージ・テキスト
CEE2EL	3	2517	CEEISEC 呼び出し内の月の値が認識されませんでした。
CEE2EM	3	2518	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
CEE2EN	3	2519	CEEISEC 呼び出し内の秒の値が認識されませんでした。
CEE2EP	3	2521	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。
CEE2ET	3	2525	CEESECS が数値フィールド内に非数値データを検出したか、あるいはタイム・スタンプ・ストリングとピクチャー・ストリングが一致しませんでした。

使用上の注意

- CEESECS の逆は CEEDATM です。CEEDATM は、*output_seconds* を文字形式に変換します。
- デフォルトでは、2 桁の年号は、システム日付より 80 年前から始まる 100 年間にあります。したがって、2008 年の場合、2 桁の年号はすべて 1928 から 2027 年の範囲内の日付を表します。この範囲を変更するには、CEESCEN 呼び出し可能サービスを使用します。

例

```

CBL LIB
*****
**
** Function: Call CEESECS to convert          **
**          timestamp to number of seconds  **
**
** In this example, calls are made to CEESECS **
** to convert two timestamps to the number of **
** seconds since 00:00:00 14 October 1582.   **
** The Lilian seconds for the earlier        **
** timestamp are then subtracted from the    **
** Lilian seconds for the later timestamp   **
** to determine the number of between the   **
** two. This result is displayed.          **
**
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLSECS.

DATA DIVISION.
WORKING-STORAGE SECTION.
01 SECOND1          COMP-2.
01 SECOND2          COMP-2.
01 TIMESTP.
02 Vstring-length  PIC S9(4) BINARY.
02 Vstring-text.
03 Vstring-char    PIC X,
                   OCCURS 0 TO 256 TIMES

```

```

                                DEPENDING ON Vstring-length
                                of TIMESTP.
01 TIMESTP2.
  02 Vstring-length      PIC S9(4) BINARY.
  02 Vstring-text.
    03 Vstring-char      PIC X,
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Vstring-length
                        of TIMESTP2.
01 PICSTR.
  02 Vstring-length      PIC S9(4) BINARY.
  02 Vstring-text.
    03 Vstring-char      PIC X,
                        OCCURS 0 TO 256 TIMES
                        DEPENDING ON Vstring-length
                        of PICSTR.
01 FC.
  02 Condition-Token-Value.
  COPY CEEIGZCT.
    03 Case-1-Condition-ID.
      04 Severity      PIC S9(4) COMP.
      04 Msg-No        PIC S9(4) COMP.
    03 Case-2-Condition-ID
      REDEFINES Case-1-Condition-ID.
      04 Class-Code    PIC S9(4) COMP.
      04 Cause-Code    PIC S9(4) COMP.
    03 Case-Sev-Ctl     PIC X.
    03 Facility-ID      PIC XXX.
  02 I-S-Info           PIC S9(9) COMP.
PROCEDURE DIVISION.

```

PARA-SECS1.

```

*****
** Specify first timestamp and a picture string
**   describing the format of the timestamp
**   as input to CEESECS
*****
MOVE 25 TO Vstring-length of TIMESTP.
MOVE '1969-05-07 12:01:00.000'
    TO Vstring-text of TIMESTP.
MOVE 25 TO Vstring-length of PICSTR.
MOVE 'YYYY-MM-DD HH:MI:SS.999'
    TO Vstring-text of PICSTR.

*****
** Call CEESECS to convert the first timestamp
**   to Lilian seconds
*****
CALL 'CEESECS' USING TIMESTP, PICSTR,
                    SECOND1, FC.
IF NOT CEE000 of FC THEN
  DISPLAY 'CEESECS failed with msg '
    Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

```

PARA-SECS2.

```

*****
** Specify second timestamp and a picture string
**   describing the format of the timestamp as
**   input to CEESECS.
*****
MOVE 25 TO Vstring-length of TIMESTP2.
MOVE '2004-01-01 00:00:01.000'
    TO Vstring-text of TIMESTP2.
MOVE 25 TO Vstring-length of PICSTR.
MOVE 'YYYY-MM-DD HH:MI:SS.999'

```


TO Vstring-text of PICSTR.

```
*****
** Call CEESECS to convert the second timestamp
** to Lillian seconds
*****
CALL 'CEESECS' USING TIMESTP2, PICSTR,
      SECOND2, FC.
IF NOT CEE000 of FC THEN
  DISPLAY 'CEESECS failed with msg '
      Msg-No of FC UPON CONSOLE
  STOP RUN
END-IF.

PARA-SECS2.
*****
** Subtract SECOND2 from SECOND1 to determine the
** number of seconds between the two timestamps
*****
SUBTRACT SECOND1 FROM SECOND2.
DISPLAY 'The number of seconds between '
      Vstring-text OF TIMESTP ' and '
      Vstring-text OF TIMESTP2 ' is: ' SECOND2.

GOBACK.
```

609 ページの『例: 日時のピクチャー・ストリング』

関連参照

607 ページの『ピクチャー文字項およびストリング』

CEEUTC: 協定世界時の取得

CEEUTC は CEEGMT と同じです。

関連参照

669 ページの『CEEGMT: 現在のグリニッジ標準時の取得』

IGZEDT4: 現在日付の取得

IGZEDT4 は、4桁年号を使用した現在日付を YYYYMMDD 形式で戻します。

CALL IGZEDT4 の構文

```
▶▶—CALL—"IGZEDT4"—USING—output_char_date.—▶▶
```

output_char_date (出力)

現在の年、月、日を表す、YYYYMMDD 形式の 8 バイト固定長文字ストリング。

使用上の注意: IGZEDT4 は、CICS ではサポートされません。

例

```

CBL LIB
*****
** Function: IGZEDT4 - get current date in the **
**                format YYYYMMDD.          **
*****
IDENTIFICATION DIVISION.
PROGRAM-ID. CBLEDT4.

. . .
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CHRDATE                PIC S9(8) USAGE DISPLAY.

. . .
PROCEDURE DIVISION.
PARA-CBLEDT4.
*****
** Call IGZEDT4.
*****
CALL 'IGZEDT4' USING BY REFERENCE CHRDATE.
*****
** IGZEDT4 has no nonzero return code to
**   check, so just display result.
*****
DISPLAY 'The current date is: '
CHRDATE
GOBACK.

```

付録 G. XML 参照資料

ここでは、XML パーサーおよび XML GENERATE ステートメントが特殊レジスター XML-CODE で返す XML 例外コードについて説明します。また、パーサーが検査する、XML specification の整形形式性制約条件に関する資料も記載します。

関連参照

『継続を許可する XML PARSE 例外』
697 ページの『継続を許可しない XML PARSE 例外』
700 ページの『XML 準拠』
702 ページの『XML GENERATE 例外』
XML specification

継続を許可する XML PARSE 例外

例外イベントが発生すると、パーサーは特殊レジスター XML-CODE を例外を識別する値に設定します。XML-CODE の値によっては、パーサーが処理を続行できる場合があります。

関連する例外コードが以下のいずれかの範囲内であれば、パーサーは例外イベント後も処理を続行できます。

- 1-99
- 100,001 から 165,535
- 200,001-265,535

次の表には、それぞれの例外と、例外発生後の続行要求時にパーサーが実行するアクションを記述しています。記述の中には、以下の用語を使用しているものがあります。

- 実際の文書エンコード
- 文書エンコード宣言
- 外部 ASCII コード・ページ
- 外部 EBCDIC コード・ページ

用語の定義については、XML 入力文書エンコードに関する下記の関連概念を参照してください。

表 91. 続行可能な XML PARSE 例外

コード	説明	継続されるパーサーのアクション
1	パーサーで、エレメントの内容に含まれない空白文字を走査中に、無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

表 91. 続行可能な XML PARSE 例外 (続き)

コード	説明	継続されるパーサーのアクション
2	パーサーで、エレメント内容に含まれない、処理命令、エレメント、コメント、または文書タイプ宣言の無効な開始が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
3	パーサーで、重複する属性名が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
4	パーサーで、属性値にマークアップ文字 '<' が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
5	エレメントの開始および終了タグ名が一致しません。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
6	パーサーで、エレメント内容に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
7	パーサーで、エレメント内容に、エレメント、コメント、処理命令、または CDATA セクションの無効な開始が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

表 91. 続行可能な XML PARSE 例外 (続き)

コード	説明	継続されるパーサーのアクション
8	パーサーで、エレメント内容に、一致する開始文字シーケンス ' <code><![CDATA[</code> ' のない、CDATA 終了文字シーケンス ' <code>]]></code> ' が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
9	パーサーで、コメント内に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
10	パーサーで、コメント内に、後にパーサーで、コメント内に、後に ' <code>></code> ' が付いていない文字シーケンス ' <code>--</code> ' (2 つのハイフン) が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
11	パーサーで、処理命令データ・セグメント内に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
12	XML 宣言が文書の先頭にありませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
13	パーサーで、16 進文字参照 (形式 <code>&#xddd;</code> の) 内に無効な数字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

|
|

表 91. 続行可能な XML PARSE 例外 (続き)

コード	説明	継続されるパーサーのアクション
14	パーサーで、10 進数文字参照 (形式 &#ddd; の) 内に無効な数字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
15	XML 宣言内のエンコード宣言値が小文字または大文字の A から Z で始まっていませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
16	文字参照が適切な XML 文字を参照しませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
17	パーサーで、エンティティー参照名に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
18	パーサーで、属性値に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
70	実際の文書エンコードは EBCDIC であり、外部 EBCDIC コード・ページがサポートされていますが、文書エンコード宣言ではサポートされる EBCDIC コード・ページが指定されていませんでした。	パーサーは、外部 EBCDIC コード・ページで指定されたエンコードを使用します。
71	実際の文書エンコードは EBCDIC であり、文書エンコード宣言ではサポートされる EBCDIC エンコードが指定されていましたが、外部 EBCDIC コード・ページがサポートされていません。	パーサーは、文書エンコード宣言で指定されたエンコードを使用します。

表 91. 続行可能な XML PARSE 例外 (続き)

コード	説明	継続されるパーサーのアクション
72	実際の文書エンコードは EBCDIC ですが、外部 EBCDIC コード・ページがサポートされておらず、文書はエンコード宣言を含んでいませんでした。	パーサーは、EBCDIC コード・ページ 1140 (USA、カナダ、... ユーロ国別拡張コード・ページ) を使用します。
73	実際の文書エンコードは EBCDIC ですが、外部 EBCDIC コード・ページおよび文書エンコード宣言のどちらにもサポートされる EBCDIC コード・ページが指定されていませんでした。	パーサーは、EBCDIC コード・ページ 1140 (USA、カナダ、... ユーロ国別拡張コード・ページ) を使用します。
80	実際の文書エンコードは ASCII であり、外部 ASCII コード・ページがサポートされていますが、文書エンコード宣言ではサポートされる ASCII コード・ページが指定されていませんでした。	パーサーは、外部 ASCII コード・ページで指定されたエンコードを使用します。
81	実際の文書エンコードは ASCII であり、文書エンコード宣言ではサポートされる ASCII エンコードが指定されていましたが、外部 ASCII コード・ページがサポートされていません。	パーサーは、文書エンコード宣言で指定されたエンコードを使用します。
82	実際の文書エンコードは ASCII でしたが、外部 ASCII コード・ページがサポートされておらず、文書にエンコード宣言が含まれていませんでした。	パーサーは、ASCII コード・ページ 819 (ISO-8859-1 Latin 1/オープン・システム) を使用します。
83	実際の文書エンコードは ASCII でしたが、外部 ASCII コード・ページと文書エンコード宣言の両方がサポートされる ASCII コード・ページを指定していませんでした。	パーサーは、ASCII コード・ページ 819 (ISO-8859-1 Latin 1/オープン・システム) を使用します。
84	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページには UTF-8 が指定されていて、文書にエンコード宣言が含まれていませんでした。	パーサーは UTF-8 を使用します。
85	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページには UTF-8 が指定されていて、文書エンコード宣言にはサポート対象の ASCII コード・ページも UTF-8 も指定されていませんでした。	パーサーは UTF-8 を使用します。
86	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページにはサポート対象の ASCII コード・ページが指定されていて、文書エンコード宣言には UTF-8 が指定されていました。	パーサーは UTF-8 を使用します。

表 91. 続行可能な XML PARSE 例外 (続き)

コード	説明	継続されるパーサーのアクション
87	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページおよび文書エンコード宣言の両方に UTF-8 が指定されていました。	パーサーは UTF-8 を使用します。
88	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページにはサポート対象の ASCII コード・ページも UTF-8 も指定されていなく、文書エンコード宣言には UTF-8 が指定されていました。	パーサーは UTF-8 を使用します。
89	実際の文書エンコードは ASCII でしたが無効な UTF-8 であり、外部コード・ページには UTF-8 が指定されていて、文書エンコード宣言にはサポート対象の ASCII コード・ページが指定されていました。	パーサーは UTF-8 を使用します。
92	文書データ項目は英数字でしたが、実際の文書エンコードは Unicode UTF-16 でした。	パーサーはコード・ページ 1200 (Unicode UTF-16) を使用します。
100,001 から 165,535	外部 EBCDIC コード・ページおよび文書エンコード宣言に指定された、サポートされる EBCDIC コード・ページがそれぞれ異なっていました。XML-CODE には、エンコード宣言に 100,000 をプラスするためのコード・ページ CCSID が含まれています。	EXCEPTION イベントから戻る前に、XML-CODE をゼロに設定した場合、パーサーでは、外部 EBCDIC コード・ページによって指定したエンコードが使用されます。文書エンコード宣言に対して (100,000 を減算して) XML-CODE を CCSID に設定した場合、パーサーではこのエンコードが使用されます。
200,001 - 265,535	外部 ASCII コード・ページおよび文書エンコード宣言に指定された、サポートされる ASCII コード・ページがそれぞれ異なっていました。XML-CODE には、エンコード宣言に 200,000 をプラスするための CCSID が含まれています。	EXCEPTION イベントから戻る前に、XML-CODE をゼロに設定した場合、パーサーでは、外部 ASCII コード・ページによって指定したエンコードが使用されます。文書エンコード宣言に対して (200,000 を減算して) XML-CODE を CCSID に設定した場合、パーサーではこのエンコードが使用されます。

関連概念

393 ページの『XML-CODE』

398 ページの『XML 入力文書エンコード』

関連タスク

401 ページの『XML PARSE の例外処理』

継続を許可しない XML PARSE 例外

例外イベントが発生すると、パーサーは特殊レジスター XML-CODE を例外を識別する値に設定します。パーサーが処理を続行できなくなる例外について以下で説明します。

XML-CODE をゼロに設定していて、例外の処理後に制御をパーサーに返しても、下記の表に示された例外については、これ以上のイベントはパーサーから返されません。パーサーは、ON EXCEPTION 句で指定したステートメント、または XML PARSE ステートメントの最後 (ON EXCEPTION 句をコーディングしていない場合) に制御を渡します。

表 92. 継続を許可しない XML PARSE 例外

コード	説明
100	パーサーで、XML 宣言の開始を走査中に、文書の末尾に達しました。
101	パーサーで、XML 宣言の末尾を走査中に、文書の末尾に達しました。
102	パーサーで、ルート・エレメントを走査中に、文書の末尾に達しました。
103	パーサーで、XML 宣言内のバージョン情報を走査中に、文書の末尾に達しました。
104	パーサーで、XML 宣言内のバージョン情報値を走査中に、文書の末尾に達しました。
106	パーサーで、XML 宣言内のエンコード宣言値を走査中に、文書の末尾に達しました。
108	パーサーで、XML 宣言内の standalone 宣言値を走査中に、文書の末尾に達しました。
109	パーサーで、属性名を走査中に、文書の末尾に達しました。
110	パーサーで、属性値を走査中に、文書の末尾に達しました。
111	パーサーで、属性値内の文字参照またはエンティティー参照を走査中に、文書の末尾に達しました。
112	パーサーで、空のエレメント・タグを走査中に、文書の末尾に達しました。
113	パーサーで、ルート・エレメント名を走査中に、文書の末尾に達しました。
114	パーサーで、エレメント名を走査中に、文書の末尾に達しました。
115	パーサーで、エレメント内容の文字データを走査中に、文書の末尾に達しました。
116	パーサーで、エレメント内容の処理命令を走査中に、文書の末尾に達しました。
117	パーサーで、エレメント内容のコメントまたは CDATA セクションを走査中に文書の末尾に達しました。
118	パーサーで、エレメント内容のコメントを走査中に文書の末尾に達しました。
119	パーサーで、エレメント内容の CDATA セクションを走査中に文書の末尾に達しました。
120	パーサーで、エレメント内容の文字参照またはエンティティー参照を走査中に文書の末尾に達しました。
121	パーサーで、ルート・エレメントの末尾を走査中に、文書の末尾に達しました。
122	パーサーで、文書タイプ宣言の無効の可能性のある開始が見つかりました。

表 92. 継続を許可しない XML PARSE 例外 (続き)

コード	説明
123	パーサーで、2 つ目の文書タイプ宣言が見つかりました。
124	ルート・エレメントの先頭文字が、文字、'_'、または ':' ではありませんでした。
125	エレメントの先頭の属性名の先頭文字が、文字、'_'、または ':' ではありませんでした。
126	パーサーで、エレメント名内に、またはエレメント名の後のいずれかに無効文字が見つかりました。
127	パーサーで、属性名の後に '=' 以外の文字が見つかりました。
128	パーサーで、無効な属性値区切り文字が見つかりました。
130	属性名の先頭文字が、文字、'_'、または ':' ではありませんでした。
131	パーサーで、属性名内に、または属性名の後のいずれかに無効文字が見つかりました。
132	空のエレメント・タグが、'/' の後に続く '>' で終了しませんでした。
133	エレメント終了タグ名の先頭文字が、文字、'_'、または ':' ではありませんでした。
134	エレメント終了タグ名が '>' で終了しませんでした。
135	エレメント名の先頭文字が、文字、'_'、または ':' ではありませんでした。
136	パーサーで、エレメント内容に、コメントまたは CDATA セクションの無効な開始が見つかりました。
137	パーサーで、コメントの無効な開始が見つかりました。
138	処理命令ターゲット名の先頭文字が、文字、'_'、または ':' ではありませんでした。
139	パーサーで、処理命令ターゲット名内に、または処理命令ターゲット名の後のいずれかに無効文字が見つかりました。
140	処理命令が終了文字シーケンス '?>' で終了しませんでした。
141	パーサーで、文字参照またはエンティティー参照内の '&' の後に無効文字が見つかりました。
142	バージョン情報が XML 宣言にありませんでした。
143	XML 宣言内の 'version' の後に '=' がありませんでした。
144	XML 宣言内のバージョン宣言値が欠落しているか、または不適切に区切られています。
145	XML 宣言内のバージョン情報値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
146	パーサーで、XML 宣言内のバージョン情報値の終了区切り文字の後に無効文字が見つかりました。
147	パーサーで、XML 宣言にオプションのエンコード宣言ではない、無効な属性が見つかりました。
148	XML 宣言内の 'encoding' の後に '=' がありませんでした。
149	XML 宣言内のエンコード宣言値が欠落しているか、または不適切に区切られています。
150	XML 宣言内のエンコード宣言値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。

表 92. 継続を許可しない XML PARSE 例外 (続き)

コード	説明
151	パーサーで、XML 宣言内のエンコード宣言値の終了区切り文字の後に無効文字が見つかりました。
152	パーサーで、XML 宣言にオプションの standalone 宣言ではない、無効な属性が見つかりました。
153	XML 宣言内の standalone の後に = がありませんでした。
154	XML 宣言内の standalone 宣言値が欠落しているか、または不適切に区切られています。
155	standalone 宣言値が 'yes' または 'no' 以外の値になっていました。
156	XML 宣言内の standalone 宣言値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
157	パーサーで、XML 宣言内の standalone 宣言値の終了区切り文字の後に無効文字が見つかりました。
158	XML 宣言が正しい文字シーケンス '?>' で終了しなかったか、無効属性が含まれていました。
159	パーサーで、ルート・エレメントの末尾の後に文書タイプ宣言の開始が見つかりました。
160	パーサーで、ルート・エレメントの末尾の後にエレメントの開始が見つかりました。
161	パーサーは、無効な UTF-8 バイト・シーケンスを検出しました。
162	パーサーは、x'FFFF' より大きな Unicode スカラー値を持つ UTF-8 文字を検出しました。
315	実際の文書エンコードは、UTF-16 リトル・エンディアンですが、パーサーはこのプラットフォームではリトル・エンディアンをサポートしません。
316	実際の文書エンコードは UCS4 ですが、パーサーは UCS4 をサポートしません。
317	パーサーで、文書エンコードを判別できません。文書は破損している可能性があります。
318	実際の文書エンコードは UTF-8 ですが、パーサーは UTF-8 をサポートしません。
320	文書データ項目は国別でしたが、実際の文書エンコードは EBCDIC でした。
321	文書データ項目は国別でしたが、実際の文書エンコードは ASCII でした。
322	文書データ項目はネイティブ英数字データ項目でしたが、実際の文書エンコードは EBCDIC でした。
323	文書データ項目はホスト英数字データ項目でしたが、実際の文書エンコードは ASCII でした。
324	文書データ項目は国別でしたが、実際の文書エンコードは UTF-8 でした。
325	文書データ項目はホスト英数字データ項目でしたが、実際の文書エンコードは UTF-8 でした。
500-599	内部エラーこのエラーをサービス担当者に報告してください。

関連概念

393 ページの『XML-CODE』

関連タスク

401 ページの『XML PARSE の例外処理』

XML 準拠

COBOL for AIX に含まれる組み込み COBOL XML パーサーは、*XML specification* の定義によると、XML 準拠のプロセッサではありません。パーサーは、構文解析する XML 文書の妥当性検査を行いません。XML パーサーは、各種の整形形式性エラーの検査は行いますが、妥当性検証をしない XML プロセッサに必要とされるアクションのすべてを行うわけではありません。

特に、XML パーサーは、内部文書タイプ定義 (DTD 内部サブセット) を処理しません。したがって、XML パーサーは、デフォルト属性値の提供、属性値の正規化、および事前定義エンティティを除く、内部エンティティの置換テキストの組み込みを行いません。ただし、XML パーサーは、文書タイプ宣言全体を DOCUMENT-TYPE-DECLARATION XML イベントの XML-TEXT または XML-NTEXT の内容として渡します。したがって、アプリケーションは、必要に応じて、これらのアクションを実行することができます。

オプションとして、パーサーを使用すると、プログラムはエラー後に XML 文書の処理を続行することができます。処理の続行を可能にする目的は、XML 文書および処理プロシーチャーのデバッグを容易にすることです。

XML specification での定義を要約すると、以下のような、テキスト・オブジェクトは整形形式の XML 文書です。

- 概して言えば、XML 文書の文法に準拠している。
- *XML specification* にリストされた明示的な整形形式制約をすべて満たす。
- 文書内で直接的または間接的に参照する、構文解析したエンティティ (テキストのセグメント) がそれぞれ、整形形式である。

COBOL XML パーサーでは、文書タイプ宣言を除き、文書が XML 文法に準拠しているかを検査します。文書タイプ宣言は、チェックされない状態でそのままアプリケーションに渡されます。

次の情報は、*XML specification* からの注釈です。オリジナル URL (www.w3.org/TR/REC-xml) に存在しない内容については、W3C は責任を負いません。注釈はすべて、仕様には含まれておらず、イタリック体で記述されています。

Copyright © 1994-2001 W3C® (マサチューセッツ工科大学、フランス国立情報学自動制御研究所、慶応義塾大学)、All Rights Reserved. W3C の責任、商標、文書使用、およびソフトウェア・ライセンスの規則が適用されます。 (www.w3.org/Consortium/Legal/ipr-notice-20000612)

また、*XML specification* には、12 事項の明示的な整形形式性制約が記述されています。COBOL XML パーサーが部分的または全面的に検査する制約事項は太字で記述されています。

1. 内部サブセット内のパラメーター・エンティティ (PE): 「内部 DTD サブセットの中では、パラメーター・エンティティ参照は、マークアップ宣言の内

部ではなく、マークアップ宣言が発生できる場所でのみ発生できない。(この制約は、外部パラメーター・エンティティの中で発生する参照や外部サブセットには適用されない。)

パーサーは内部 DTD サブセットを処理しないので、この制約を強制しません。

2. 外部サブセット: 「外部サブセットが存在している場合は、そのプロダクションを extSubset に一致させなくてはならない。」

パーサーは外部サブセットを処理しないので、この制約を強制しません。

3. 宣言と宣言の間のパラメーター・エンティティ: 「DeclSep のパラメーター・エンティティ参照のテキストの置換はそのプロダクションを extSubsetDecl に一致させなくてはならない。」

パーサーは内部 DTD サブセットを処理しないので、この制約を強制しません。

4. エレメント・タイプの突き合わせ: 「エレメントの終了タグの名前は、開始タグのエレメント・タイプと一致させなければならない。」

パーサーで、この規制は強制されます。

5. 属性指定の一意性: 「同じ開始タグまたは空エレメント・タグの中に 1 回を超えて現れてよい属性名はない。」

パーサーで、一意性について、指定したエレメント内の最大 10 までの属性名が検査され、この制約は部分的にサポートされます。アプリケーションでは、この制限を超えて、すべての属性名を検査することができます。

6. 外部エンティティ参照の禁止: 「属性値は、外部エンティティへの直接的または間接的エンティティ参照を含むことができない。」

パーサーはこの制約を強制しません。

7. 属性値内の '<' の禁止: 「属性値の中で直接的または間接的に参照するエンティティの置換テキストは、'<' を含んではならない。」

パーサーはこの制約を強制しません。

8. 正しい文字: 「文字参照の使用に言及する文字は、そのプロダクションを Char に一致させなくてはならない。」

パーサーで、この規制は強制されます。

9. エンティティの宣言: 「DTD のない文書、パラメーター・エンティティ参照が含まれない内部 DTD サブセットだけしかない文書、または standalone='yes' の文書においては、外部サブセットまたはパラメーター・エンティティ内で発生しないエンティティ参照において与えられている Name が、外部サブセット、またはパラメーター・エンティティ内で発生しないエンティティ宣言の中のものとは一致しなければならない。ただし、整形形式文書は、次のエンティティ amp、lt、gt、apos、quot を宣言する必要はない。一般的エンティティの宣言は、属性リスト宣言内のデフォルト値に現れる、それに対するどの参照よりも先行しなければならない。」

エンティティが外部サブセットまたは外部パラメーター・エンティティで宣言されている場合、妥当性検査をしないプロセッサは、それらの宣言を読み取って処理するよう強制されないことに注意してください。そのような文書の場合、エンティティを宣言しなければならないという規則は、standalone='yes' の場合にのみ整形形式制約です。

パーサーはこの制約を強制しません。

10. 解析済みエンティティ: 「エンティティ参照は、解析対象外エンティティの名前を含んではならない。解析対象外エンティティを参照してもよいのは、ENTITY 型または ENTITIES 型として宣言した属性値の中だけである。」

パーサーはこの制約を強制しません。

11. 再帰の禁止: 「解析されるエンティティは、直接的または間接的を問わず、それ自身への再帰的参照を含めてはならない。」

パーサーはこの制約を強制しません。

12. DTD 内: 「パラメーター・エンティティ参照が出現してよいのは、DTD の中だけである。」

このエラーは発生する可能性がないため、パーサーで、この規制は強制されません。

この資料は、XML specification からの注釈です。オリジナル URL (www.w3.org/TR/REC-xml) に存在しない内容については、W3C は責任を負いません。上記の注釈はすべて、仕様には含まれていません。本文書は、W3C メンバーや他の関係者による校閲を受け、ディレクターによって W3C 推奨文書として承認されています。本文書は継続的に提供される文書であり、参照資料として使用されたり、別文書で仕様に含まれる資料として引用される可能性があります。仕様の正規版は英語バージョンで、W3C のサイトで確認することができます。翻訳文書については、翻訳上の誤りが含まれる可能性があります。

関連概念

388 ページの『COBOL での XML パーサー』

関連参照

Extensible Markup Language (XML)

XML specification (Prolog and document type declaration)

XML GENERATE 例外

XML の生成時に、いずれかの例外コードが XML-CODE 特殊レジスターで戻されることがあります。このような例外が発生すると、ON EXCEPTION 句で指定されたステートメント、または、ON EXCEPTION 句をコーディングしていない場合には、XML GENERATE ステートメントの末尾に制御が渡されます。

表 93. XML GENERATE 例外

コード	説明
400	受信は小さすぎて、生成された XML 文書を入れられませんでした。指定されていれば、COUNT IN データ項目に、実際に生成された文字位置のカウントが格納されています。

表 93. XML GENERATE 例外 (続き)

コード	説明
401	マルチバイト・データ名は、Unicode への変換時に XML エlementまたは属性名では無効な文字を含んでいました。
402	Unicode への変換時に、マルチバイト・データ名の先頭文字は、XML Elementまたは属性名の先頭文字としては無効なものでした。
403	OCCURS DEPENDING ON 変数の値が 16,777,215 を超えました。
410	XML 文書に対して指定されたエンコードは、Unicode への変換に関してはサポートされません。
411	受け取り側はホスト英数字でしたが、文書に対して指定されたエンコードは、サポート対象の 1 バイト EBCDIC コード・ページではありません。
412	受け取り側はネイティブ英数字でしたが、文書に対して指定されたエンコードは UTF-8 でもサポート対象の 1 バイト ASCII コード・ページでもありませんでした。
413	受け取り側は英数字でしたが、ランタイム・ロケールがコンパイル時ロケールと一致していませんでした。
414	XML 文書に指定されたエンコードが無効だったか、またはサポートされるコード・ページではありませんでした。
415	受け取り側は国別でしたが、文書に対して指定されたエンコードは UTF-16 ではありませんでした。
416	XML 名前空間 ID に無効な XML 文字が含まれていました。
417	Element文字コンテンツまたは属性値に XML コンテンツでは正しくない文字が含まれていました。文書内の「hex」が接頭部のElement・タグ名または属性名および元のデータ値を 16 進表記して、XML の生成を続行しました。
418	置換文字がエンコード変換で生成されました。
419	XML 名前空間接頭部が無効でした。
420	ソース・データ項目にはマルチバイトの名前または内容が含まれていて、受け取り側はネイティブ英数字でしたが、文書に対して指定されたエンコードは UTF-8 ではありませんでした。
600-699	内部エラーエラーをサービス担当者に報告してください。

関連タスク

412 ページの『XML GENERATE 例外の処理』

関連参照

616 ページの『XML 生成』

付録 H. JNI.cpy

このリストはコピーブック JNI.cpy を示しています。これを使用すると、COBOL プログラムから Java Native Interface (JNI) サービスにアクセスすることができます。

JNI.cpy には、Java JNI タイプに対応するサンプル COBOL データ定義と、JNI 呼び出し可能サービスにアクセスするための関数ポインターを含む JNI 環境構造である JNINativeInterface が入っています。

JNI.cpy は、COBOL インストール・ディレクトリーの include サブディレクトリー (通常は /usr/lpp/cobol/include) にあります。JNI.cpy は、C プログラマーが JNI にアクセスするために使用するヘッダー・ファイル jni.h に類似しています。

```
*****
* COBOL declarations for Java native method interoperation      *
*                                                                 *
* To use the Java Native Interface callable services from a     *
* COBOL program:                                               *
* 1) Use a COPY statement to include this file into the        *
*    the Linkage Section of the program, e.g.                  *
*    Linkage Section.                                          *
*    Copy JNI                                                  *
* 2) Code the following statements at the beginning of the     *
*    Procedure Division:                                       *
*    Set address of JNIEnv to JNIEnvPtr                        *
*    Set address of JNINativeInterface to JNIEnv              *
*****
*
* Sample JNI type definitions in COBOL
*
*01 jboolean1 pic X.
* 88 jboolean1-true  value X'01' through X'FF'.
* 88 jboolean1-false value X'00'.
*
*01 jbyte1 pic X.
*
*01 jchar1 pic N usage national.
*
*01 jshort1 pic s9(4)  comp-5.
*01 jint1  pic s9(9)  comp-5.
*01 jlong1 pic s9(18) comp-5.
*
*01 jfloat1 comp-1.
*01 jdouble1 comp-2.
*
*01 jobject1 object reference.
*01 jclass1 object reference.
*01 jstring1 object reference jstring.
*01 jarray1  object reference jarray.
*
*01 jbooleanArray1 object reference jbooleanArray.
*01 jbyteArray1   object reference jbyteArray.
*01 jcharArray1   object reference jcharArray.
*01 jshortArray1  object reference jshortArray.
*01 jintArray1    object reference jintArray.
*01 jlongArray1   object reference jlongArray.
*01 floatArray1   object reference floatArray.
*01 jdoubleArray1 object reference jdoubleArray.
*01 jobjectArray1 object reference jobjectArray.
```

```

* Possible return values for JNI functions.
01 JNI-RC pic S9(9) comp-5.
* success
  88 JNI-OK          value 0.
* unknown error
  88 JNI-ERR        value -1.
* thread detached from the VM
  88 JNI-EDETACHED value -2.
* JNI version error
  88 JNI-EVERSION  value -3.
* not enough memory
  88 JNI-ENOMEM    value -4.
* VM already created
  88 JNI-EEXIST    value -5.
* invalid arguments
  88 JNI-EINVAL    value -6.

* Used in ReleaseScalarArrayElements
01 releaseMode pic s9(9) comp-5.
  88 JNI-COMMIT value 1.
  88 JNI-ABORT  value 2.

01 JNIenv pointer.

* JNI Native Method Interface - environment structure.
01 JNINativeInterface.
  02 pointer.
  02 pointer.
  02 pointer.
  02 pointer.
  02 GetVersion          function-pointer.
  02 DefineClass         function-pointer.
  02 FindClass           function-pointer.
  02 FromReflectedMethod function-pointer.
  02 FromReflectedField function-pointer.
  02 ToReflectedMethod  function-pointer.
  02 GetSuperclass       function-pointer.
  02 IsAssignableFrom    function-pointer.
  02 ToReflectedField   function-pointer.
  02 Throw                function-pointer.
  02 ThrowNew            function-pointer.
  02 ExceptionOccurred   function-pointer.
  02 ExceptionDescribe   function-pointer.
  02 ExceptionClear      function-pointer.
  02 FatalError          function-pointer.
  02 PushLocalFrame      function-pointer.
  02 PopLocalFrame       function-pointer.
  02 NewGlobalRef         function-pointer.
  02 DeleteGlobalRef     function-pointer.
  02 DeleteLocalRef     function-pointer.
  02 IsSameObject        function-pointer.
  02 NewLocalRef         function-pointer.
  02 EnsureLocalCapacity function-pointer.
  02 AllocObject         function-pointer.
  02 NewObject           function-pointer.
  02 NewObjectV          function-pointer.
  02 NewObjectA          function-pointer.
  02 GetObjectClass      function-pointer.
  02 IsInstanceOf        function-pointer.
  02 GetMethodID         function-pointer.
  02 CallObjectMethod    function-pointer.
  02 CallObjectMethodV  function-pointer.
  02 CallObjectMethodA  function-pointer.
  02 CallBooleanMethod  function-pointer.
  02 CallBooleanMethodV function-pointer.

```

02 CallBooleanMethodA	function-pointer.
02 CallByteMethod	function-pointer.
02 CallByteMethodV	function-pointer.
02 CallByteMethodA	function-pointer.
02 CallCharMethod	function-pointer.
02 CallCharMethodV	function-pointer.
02 CallCharMethodA	function-pointer.
02 CallShortMethod	function-pointer.
02 CallShortMethodV	function-pointer.
02 CallShortMethodA	function-pointer.
02 CallIntMethod	function-pointer.
02 CallIntMethodV	function-pointer.
02 CallIntMethodA	function-pointer.
02 CallLongMethod	function-pointer.
02 CallLongMethodV	function-pointer.
02 CallLongMethodA	function-pointer.
02 CallFloatMethod	function-pointer.
02 CallFloatMethodV	function-pointer.
02 CallFloatMethodA	function-pointer.
02 CallDoubleMethod	function-pointer.
02 CallDoubleMethodV	function-pointer.
02 CallDoubleMethodA	function-pointer.
02 CallVoidMethod	function-pointer.
02 CallVoidMethodV	function-pointer.
02 CallVoidMethodA	function-pointer.
02 CallNonvirtualObjectMethod	function-pointer.
02 CallNonvirtualObjectMethodV	function-pointer.
02 CallNonvirtualObjectMethodA	function-pointer.
02 CallNonvirtualBooleanMethod	function-pointer.
02 CallNonvirtualBooleanMethodV	function-pointer.
02 CallNonvirtualBooleanMethodA	function-pointer.
02 CallNonvirtualByteMethod	function-pointer.
02 CallNonvirtualByteMethodV	function-pointer.
02 CallNonvirtualByteMethodA	function-pointer.
02 CallNonvirtualCharMethod	function-pointer.
02 CallNonvirtualCharMethodV	function-pointer.
02 CallNonvirtualCharMethodA	function-pointer.
02 CallNonvirtualShortMethod	function-pointer.
02 CallNonvirtualShortMethodV	function-pointer.
02 CallNonvirtualShortMethodA	function-pointer.
02 CallNonvirtualIntMethod	function-pointer.
02 CallNonvirtualIntMethodV	function-pointer.
02 CallNonvirtualIntMethodA	function-pointer.
02 CallNonvirtualLongMethod	function-pointer.
02 CallNonvirtualLongMethodV	function-pointer.
02 CallNonvirtualLongMethodA	function-pointer.
02 CallNonvirtualFloatMethod	function-pointer.
02 CallNonvirtualFloatMethodV	function-pointer.
02 CallNonvirtualFloatMethodA	function-pointer.
02 CallNonvirtualDoubleMethod	function-pointer.
02 CallNonvirtualDoubleMethodV	function-pointer.
02 CallNonvirtualDoubleMethodA	function-pointer.
02 CallNonvirtualVoidMethod	function-pointer.
02 CallNonvirtualVoidMethodV	function-pointer.
02 CallNonvirtualVoidMethodA	function-pointer.
02 GetFieldID	function-pointer.
02 GetObjectField	function-pointer.
02 GetBooleanField	function-pointer.
02 GetByteField	function-pointer.
02 GetCharField	function-pointer.
02 GetShortField	function-pointer.
02 GetIntField	function-pointer.
02 GetLongField	function-pointer.
02 GetFloatField	function-pointer.
02 GetDoubleField	function-pointer.
02 SetObjectField	function-pointer.
02 SetBooleanField	function-pointer.

02 SetByteField	function-pointer.
02 SetCharField	function-pointer.
02 SetShortField	function-pointer.
02 SetIntField	function-pointer.
02 SetLongField	function-pointer.
02 SetFloatField	function-pointer.
02 SetDoubleField	function-pointer.
02 GetStaticMethodID	function-pointer.
02 CallStaticObjectMethod	function-pointer.
02 CallStaticObjectMethodV	function-pointer.
02 CallStaticObjectMethodA	function-pointer.
02 CallStaticBooleanMethod	function-pointer.
02 CallStaticBooleanMethodV	function-pointer.
02 CallStaticBooleanMethodA	function-pointer.
02 CallStaticByteMethod	function-pointer.
02 CallStaticByteMethodV	function-pointer.
02 CallStaticByteMethodA	function-pointer.
02 CallStaticCharMethod	function-pointer.
02 CallStaticCharMethodV	function-pointer.
02 CallStaticCharMethodA	function-pointer.
02 CallStaticShortMethod	function-pointer.
02 CallStaticShortMethodV	function-pointer.
02 CallStaticShortMethodA	function-pointer.
02 CallStaticIntMethod	function-pointer.
02 CallStaticIntMethodV	function-pointer.
02 CallStaticIntMethodA	function-pointer.
02 CallStaticLongMethod	function-pointer.
02 CallStaticLongMethodV	function-pointer.
02 CallStaticLongMethodA	function-pointer.
02 CallStaticFloatMethod	function-pointer.
02 CallStaticFloatMethodV	function-pointer.
02 CallStaticFloatMethodA	function-pointer.
02 CallStaticDoubleMethod	function-pointer.
02 CallStaticDoubleMethodV	function-pointer.
02 CallStaticDoubleMethodA	function-pointer.
02 CallStaticVoidMethod	function-pointer.
02 CallStaticVoidMethodV	function-pointer.
02 CallStaticVoidMethodA	function-pointer.
02 GetStaticFieldID	function-pointer.
02 GetStaticObjectField	function-pointer.
02 GetStaticBooleanField	function-pointer.
02 GetStaticByteField	function-pointer.
02 GetStaticCharField	function-pointer.
02 GetStaticShortField	function-pointer.
02 GetStaticIntField	function-pointer.
02 GetStaticLongField	function-pointer.
02 GetStaticFloatField	function-pointer.
02 GetStaticDoubleField	function-pointer.
02 SetStaticObjectField	function-pointer.
02 SetStaticBooleanField	function-pointer.
02 SetStaticByteField	function-pointer.
02 SetStaticCharField	function-pointer.
02 SetStaticShortField	function-pointer.
02 SetStaticIntField	function-pointer.
02 SetStaticLongField	function-pointer.
02 SetStaticFloatField	function-pointer.
02 SetStaticDoubleField	function-pointer.
02 NewString	function-pointer.
02 GetStringLength	function-pointer.
02 GetStringChars	function-pointer.
02 ReleaseStringChars	function-pointer.
02 NewStringUTF	function-pointer.
02 GetStringUTFLength	function-pointer.
02 GetStringUTFChars	function-pointer.
02 ReleaseStringUTFChars	function-pointer.
02 GetArrayLength	function-pointer.
02 NewObjectArray	function-pointer.

02	GetObjectArrayElement	function-pointer.
02	SetObjectArrayElement	function-pointer.
02	NewBooleanArray	function-pointer.
02	NewByteArray	function-pointer.
02	NewCharArray	function-pointer.
02	NewShortArray	function-pointer.
02	NewIntArray	function-pointer.
02	NewLongArray	function-pointer.
02	NewFloatArray	function-pointer.
02	NewDoubleArray	function-pointer.
02	GetBooleanArrayElements	function-pointer.
02	GetByteArrayElements	function-pointer.
02	GetCharArrayElements	function-pointer.
02	GetShortArrayElements	function-pointer.
02	GetIntArrayElements	function-pointer.
02	GetLongArrayElements	function-pointer.
02	GetFloatArrayElements	function-pointer.
02	GetDoubleArrayElements	function-pointer.
02	ReleaseBooleanArrayElements	function-pointer.
02	ReleaseByteArrayElements	function-pointer.
02	ReleaseCharArrayElements	function-pointer.
02	ReleaseShortArrayElements	function-pointer.
02	ReleaseIntArrayElements	function-pointer.
02	ReleaseLongArrayElements	function-pointer.
02	ReleaseFloatArrayElements	function-pointer.
02	ReleaseDoubleArrayElements	function-pointer.
02	GetBooleanArrayRegion	function-pointer.
02	GetByteArrayRegion	function-pointer.
02	GetCharArrayRegion	function-pointer.
02	GetShortArrayRegion	function-pointer.
02	GetIntArrayRegion	function-pointer.
02	GetLongArrayRegion	function-pointer.
02	GetFloatArrayRegion	function-pointer.
02	GetDoubleArrayRegion	function-pointer.
02	SetBooleanArrayRegion	function-pointer.
02	SetByteArrayRegion	function-pointer.
02	SetCharArrayRegion	function-pointer.
02	SetShortArrayRegion	function-pointer.
02	SetIntArrayRegion	function-pointer.
02	SetLongArrayRegion	function-pointer.
02	SetFloatArrayRegion	function-pointer.
02	SetDoubleArrayRegion	function-pointer.
02	RegisterNatives	function-pointer.
02	UnregisterNatives	function-pointer.
02	MonitorEnter	function-pointer.
02	MonitorExit	function-pointer.
02	GetJavaVM	function-pointer.
02	GetStringRegion	function-pointer.
02	GetStringUTFRegion	function-pointer.
02	GetPrimitiveArrayCritical	function-pointer.
02	ReleasePrimitiveArrayCritical	function-pointer.
02	GetStringCritical	function-pointer.
02	ReleaseStringCritical	function-pointer.
02	NewWeakGlobalRef	function-pointer.
02	DeleteWeakGlobalRef	function-pointer.
02	ExceptionCheck	function-pointer.

関連タスク

257 ページの『オブジェクト指向アプリケーションのコンパイル』

477 ページの『JNI サービスへのアクセス』

付録 I. COBOL SYSADATA ファイルの内容

ADATA コンパイラー・オプションを使用すると、コンパイラーは、プログラム・データを含んだファイルを生成します。コンパイラー・リストではなく、このファイルを使用して、プログラムに関する情報を取り出すことができます。例えば、シンボリック・デバッグ・ツールや相互参照ツールに対応したプログラムに関する情報を取り出すことができます。

713 ページの『例: SYSADATA』

関連参照

266 ページの『ADATA』

『SYSADATA ファイルに影響する既存のコンパイラー・オプション』

712 ページの『SYSADATA レコード・タイプ』

714 ページの『SYSADATA レコード記述』

SYSADATA ファイルに影響する既存のコンパイラー・オプション

いくつかのコンパイラー・オプションは、SYSADATA ファイルの内容に影響を与えることがあります。

COMPILE

NOCOMPILE(W|E|S) を使用すると、コンパイルが実行途中で停止され、その結果、特定のメッセージが失われる可能性があります。

EXIT INEXIT はコンパイル・ソース・ファイルの識別を禁止します。

TEST TEST を使用すると、SYSADATA ファイルの内容にも影響を与える、追加のオブジェクト・テキスト・レコードが作成されます。

NUM NUM を使用すると、コンパイラーは、生成されたシーケンス番号ではなく、ソース・レコードのカラム 1 から 6 の内容を行番号に使用します。無効 (非数値)、または順不同の番号は、直前のレコードより 1 だけ大きな数値で置き換えられます。

以下の SYSADATA フィールドには、NUM|NONUM 設定によってその内容が異なる、行番号が含まれています。

タイプ	フィールド	レコード
0020	AE_LINE	外部シンボル・レコード
0030	ATOK_LINE	トークン・レコード
0032	AF_STMT	ソース・エラー・レコード
0038	AS_STMT	ソース・レコード
0039	AS_REP_EXP_SLIN	COPY REPLACING レコード
0039	AS_REP_EXP_ELIN	COPY REPLACING レコード
0042	ASY_STMT	記号レコード
0044	AX_DEFN	記号相互参照レコード

タイプ	フィールド	レコード
0044	AX_STMT	記号相互参照レコード
0046	AN_STMT	ネストされたプログラム・レコード

タイプ 0038 ソース・レコードには、行番号とレコード番号に関連する 2 つのフィールドが含まれています。

- AS_STMT には、NUM および NONUM の両方に、コンパイラ行番号が含まれています。
- AS_CUR_REC# には、物理ソース・レコード番号が含まれています。

上記の 2 つのフィールドは常に、上記フィールドのすべてにおいて使われるコンパイラ行番号と物理ソース・レコード番号を相関させるために使用されます。

残りのコンパイラ・オプションは、SYSADATA ファイルに直接的な影響は与えませんが、FLAGSAA、FLAGSTD、SSRANGE など、特定のオプションに関連付けられた、別のエラー・メッセージの生成をトリガーする可能性があります。

713 ページの『例: SYSADATA』

関連参照

『SYSADATA レコード・タイプ』

276 ページの『COMPILE』

281 ページの『EXIT』

297 ページの『NUMBER』

310 ページの『TEST』

SYSADATA レコード・タイプ

SYSADATA ファイルは、種々のレコード・タイプに分類されるレコードを含みます。各レコード・タイプには、コンパイルされる COBOL プログラムに関する情報が提供されます。

各レコードは、以下の 2 つのセクションで構成されます。

- 全レコード・タイプに対して同一の構造を有し、レコード・タイプを識別するレコード・コードを含む、12 バイトのヘッダー・セクション
- レコード・タイプによって異なる、可変長データ・セクション

表 94. SYSADATA レコード・タイプ

レコード・タイプ	アクション
717 ページの『ジョブ識別レコード: X'0000'』	ソース・データの処理に使用する環境に関する情報を記述します
717 ページの『ADATA 識別レコード: X'0001'』	SYSADATA ファイルのレコードに関する共通情報を記述します
718 ページの『コンパイル単位の開始終了レコード: X'0002'』	ソース・ファイル内のコンパイル単位の開始と終了のマーク付けを行います
718 ページの『オプション・レコード: X'0010'』	コンパイルに使用するコンパイラ・オプションを記述します

表 94. SYSADATA レコード・タイプ (続き)

レコード・タイプ	アクション
729 ページの『外部シンボル・レコード: X'0020'』	プログラム内のすべての外部名、定義、および参照を記述します
730 ページの『構文解析ツリー・レコード: X'0024'』	プログラムの構文解析ツリーにノードを定義します
744 ページの『トークン・レコード: X'0030'』	ソース・トークンを定義します
758 ページの『ソース・エラー・レコード: X'0032'』	ソース・プログラム・ステートメントのエラーを記述します
758 ページの『ソース・レコード: X'0038'』	単一のソース行を記述します
759 ページの『COPY REPLACING レコード: X'0039'』	コピーブック内のテキストとの、 COPY.?.?.REPLACING <i>operand-1</i> の突き合わせの結果として、テキスト置換のインスタンスを記述します
759 ページの『記号レコード: X'0042'』	プログラムに定義される、単一の記号を記述します。プログラムに定義される、それぞれの記号ごとに 1 つの記号レコードがあります。
772 ページの『記号相互参照レコード: X'0044'』	単一の記号への参照を記述します
773 ページの『ネストされたプログラム・レコード: X'0046'』	プログラムの名前とネスト・レベルを記述します
774 ページの『ライブラリー・レコード: X'0060'』	各ライブラリーで使用されるライブラリー・ファイルとメンバーを記述します
775 ページの『統計レコード: X'0090'』	コンパイルに関する統計を記述します
775 ページの『EVENTS レコード: X'0120'』	EVENTS レコードは、COBOL/370™ との互換性を提供します。レコード形式は、COBOL/370 と同一ですが、レコードの先頭に置かれる標準 ADATA ヘッダー、および EVENTS レコード・データの長さを示すフィールドが追加されます。

例: SYSADATA

以下の例は、COBOL プログラムのリストの一部を示しています。この COBOL プログラムを ADATA オプションを使用してコンパイルした場合には、関連データ・ファイルに生成されるレコードは、次の表に示す順序で記述されます。

000001	IDENTIFICATION DIVISION.	AD000020
000002	PROGRAM-ID. AD04202.	AD000030
000003	ENVIRONMENT DIVISION.	AD000040
000004	DATA DIVISION.	AD000050
000005	WORKING-STORAGE SECTION.	AD000060
000006	77 COMP3-FLD2 pic S9(3)v9.	AD000070
000007	PROCEDURE DIVISION.	AD000080
000008	STOP RUN.	

タイプ	説明
X'0120'	EVENTS タイム・スタンプ・レコード
X'0120'	EVENTS プロセッサ・レコード
X'0120'	EVENTS ファイル ID レコード
X'0120'	EVENTS プログラム・レコード
X'0001'	ADATA 識別レコード
X'0000'	ジョブ識別レコード
X'0010'	オプション・レコード
X'0038'	ステートメント 1 のソース・レコード
X'0038'	ステートメント 2 のソース・レコード
X'0038'	ステートメント 3 のソース・レコード
X'0038'	ステートメント 4 のソース・レコード
X'0038'	ステートメント 5 のソース・レコード
X'0038'	ステートメント 6 のソース・レコード
X'0038'	ステートメント 7 のソース・レコード
X'0038'	ステートメント 8 のソース・レコード
X'0020'	AD04202 の外部シンボル・レコード
X'0044'	STOP の記号相互参照レコード
X'0044'	COMP3-FLD2 の記号相互参照レコード
X'0044'	AD04202 の記号相互参照レコード
X'0042'	AD04202 の記号レコード
X'0042'	COMP3-FLD2 の記号レコード
X'0090'	統計レコード
X'0120'	EVENTS ファイル終わりレコード

関連参照

『SYSADATA レコード記述』

SYSADATA レコード記述

関連データ・ファイルに書き込まれるレコードのフォーマットは、以下の関連参照に示されています。

それぞれのレコード・タイプに記述されたフィールドは、以下のシンボルで表されます。

- C** 文字 (EBCDIC または ASCII) データを表す
- H** 2 バイトの 2 進整数データを表す
- F** 4 バイトの 2 進整数データを表す
- A** 4 バイトの 2 進整数アドレスとオフセット・データを表す
- X** 16 進数 (ビット) データまたは 1 バイトの 2 進整数データを表す

データ型には、境界合わせは一切含まれていません。したがって、上記の暗黙の長さは、長さ指標 (Ln) を含めることで変更される可能性があります。すべての整数データは、ヘッダー・フラグ・バイトの指標ビットによって、ビッグ・エンディアン形式、またはリトル・エンディアン形式になっています。ビッグ・エンディアン形式では、ビット 0 が常に最上位ビットで、ビット n が最下位ビットであることを意味します。リトル・エンディアンとは、Intel® プロセッサで見られる「逆順バイト」整数のことです。

未定義フィールドおよび未使用値はすべて、予約済みです。

関連参照

『共通ヘッダー・セクション』

717 ページの『ジョブ識別レコード: X'0000'』

717 ページの『ADATA 識別レコード: X'0001'』

718 ページの『コンパイル単位の開始終了レコード: X'0002'』

718 ページの『オプション・レコード: X'0010'』

729 ページの『外部シンボル・レコード: X'0020'』

730 ページの『構文解析ツリー・レコード: X'0024'』

744 ページの『トークン・レコード: X'0030'』

758 ページの『ソース・エラー・レコード: X'0032'』

758 ページの『ソース・レコード: X'0038'』

759 ページの『COPY REPLACING レコード: X'0039'』

759 ページの『記号レコード: X'0042'』

772 ページの『記号相互参照レコード: X'0044'』

773 ページの『ネストされたプログラム・レコード: X'0046'』

774 ページの『ライブラリー・レコード: X'0060'』

775 ページの『統計レコード: X'0090'』

775 ページの『EVENTS レコード: X'0120'』

共通ヘッダー・セクション

次の表は、すべてのレコード・タイプに共通するヘッダー・セクションの形式を示しています。MVS™ および VSE の場合、各レコードの前に 4 バイトの RDW (レコード記述子ワード) が置かれます。この RDW は通常、アクセス方式によってのみ使用され、ダウンロード・ユーティリティによって、ストリップされます。

表 95. SYSADATA 共通ヘッダー・セクション

フィールド	サイズ	説明
言語コード	XL1	16 高水準アセンブラー
		17 すべてのプラットフォームの COBOL
		40 サポートされるプラットフォームの PL/I

表 95. SYSADATA 共通ヘッダー・セクション (続き)

フィールド	サイズ	説明
レコード・タイプ	HL2	以下のいずれかのレコード・タイプ。 X'0000' ジョブ識別レコード ¹ X'0001' ADATA 識別レコード X'0002' コンパイル単位の開始/終了レコード X'0010' オプション・レコード ¹ X'0020' 外部シンボル・レコード X'0024' 構文解析ツリー・レコード X'0030' トークン・レコード X'0032' ソース・エラー・レコード X'0038' ソース・レコード X'0039' COPY REPLACING レコード X'0042' 記号レコード X'0044' 記号相互参照レコード X'0046' ネストされたプログラム・レコード X'0060' ライブラリー・レコード X'0090' 統計レコード ¹ X'0120' EVENTS レコード
関連データ・アーキテクチャー・レベル	XL1	3 ヘッダー構造の定義レベル
フラグ	XL11. ADATA レコード整数は、リトル・エンディアン (Intel) 形式です1 このレコードは、次のレコードに続行されます 1111 11.. 将来の利用のために予約済み
関連データ・レコード・エディション・レベル	XL1	特定のレコード・タイプの新規形式を表すために使用され、通常は 0 です
予約済み	CL4	将来の利用のために予約済み
関連データ・フィールド長	HL2	ヘッダーの後に置かれるデータの長さ (バイト単位)
1. バッチ・コンパイル (一連のプログラム) が ADATA オプションで実行されるときには、それぞれのコンパイルごとに、複数ジョブ識別、オプション、および統計レコードがあります。		

12 バイト・ヘッダーのマッピングには、MVS および VSE のアクセス方式が必要とする、可変長レコード記述子ワードに使用される領域は組み込まれません。

ジョブ識別レコード: X'0000'

次の表に、ジョブ識別レコードの内容を示します。

表 96. SYSADATA ジョブ識別レコード

フィールド	サイズ	説明
日付	CL8	YYYYMMDD 形式のコンパイルの日付
時刻	CL4	HHMM 形式のコンパイルの時刻
プロダクト番号	CL8	関連データ・ファイルを生成したコンパイラーのプロダクト番号
プロダクト・バージョン	CL8	V.R.M 形式の、関連データ・ファイルを生成したプロダクトのバージョン番号
PTF レベル	CL8	関連データ・ファイルを生成したプロダクトの PTF レベル番号 (PTF 番号が利用不可の場合、このフィールドはブランクです。)
システム ID	CL24	コンパイルが実行されたシステムのシステム識別
ジョブ名	CL8	コンパイル・ジョブの MVS ジョブ名
ステップ名	CL8	コンパイル・ステップの MVS ステップ名
PROC ステップ	CL8	コンパイル・プロシージャの MVS プロシージャ・ステップ名
入力ファイル数 ¹	HL2	このレコードに記録した入力ファイルの数 以下の、7 つのフィールドのグループでは、このフィールドの値に従って、 <i>n</i> 回発生します。
...入力ファイル番号	HL2	ファイルの割り当てシーケンス番号
...入力ファイル名長	HL2	次の入力ファイル名の長さ
...ボリューム通し番号長	HL2	ボリューム通し番号の長さ
...メンバー名長	HL2	メンバー名の長さ
...入力ファイル名	CL(<i>n</i>)	コンパイルの入力ファイルの名前
...ボリューム通し番号	CL(<i>n</i>)	入力ファイルが常駐する (最初の) ボリュームのボリューム通し番号
...メンバー名	CL(<i>n</i>)	該当する場合には、入力ファイル内のメンバーの名前

1. 入力ファイル数が、関連データ・ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。入力ファイルの現行数 (そのレコードの) はレコードに保管され、レコードは関連データ・ファイルに書き込まれます。入力ファイルの残りは次のレコードに入れられます。入力ファイルの数のカウントは、現行レコードのカウントです。

ADATA 識別レコード: X'0001'

次の表に、ADATA 識別レコードの内容を示します。

表 97. ADATA 識別レコード

フィールド	サイズ	説明
時刻 (2 進数)	XL8	世界時 (UT) は、真夜中のグリニッジ標準時を基準としたマイクロ秒数を表す 2 進数で、下位ビットは 1 マイクロ秒を表します。この時刻は、時間帯から独立したタイム・スタンプとして使用することができます。 Windows および AIX システムでは、フィールドのバイト 5 から 8 のみが、時刻を入れるフルワード・バイナリー・フィールドとして使用されます。
CCSID ¹	XL2	コード化文字セット ID
文字セット・フラグ	XL1	X'80' EBCDIC (IBM-037) X'40' ASCII (IBM-1252)
コード・ページ名長	XL2	後に続く、コード・ページ名の長さ
コード・ページ名	CL(n)	コード・ページの名前
1. 適切な CCS フラグが常に設定されます。CCSID がゼロ以外に設定されると、コード・ページ名の長さはゼロとなります。CCSID がゼロに設定されると、コード・ページ名の長さはゼロ以外の値となり、コード・ページ名が表示されます。		

コンパイル単位の開始|終了レコード: X'0002'

次の表に、コンパイル単位の開始|終了レコードの内容を示します。

表 98. SYSADATA コンパイル単位の開始|終了レコード

フィールド	サイズ	説明
タイプ	HL2	コンパイル単位タイプ。以下のどちらかが使用できます。 X'0000' 開始コンパイル単位 X'0001' 終了コンパイル単位
予約済み	CL2	将来の利用のために予約済み
予約済み	FL4	将来の利用のために予約済み

オプション・レコード: X'0010'

次の表に、オプション・レコードの内容を示します。

表 99. SYSADATA オプション・レコード

フィールド	サイズ	説明
オプション・バイト 0	XL1	1111 1111 将来の利用のために予約済み

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 1	XL1	<p>1... ビット 1 = DECK、ビット 0 = NODECK</p> <p>.1.. ビット 1 = ADATA、ビット 0 = NOADATA</p> <p>..1. ビット 1 = COLLSEQ(EBCDIC)、ビット 0 = COLLSEQ(LOCALE BINARY) (Windows および AIX のみ)</p> <p>...1 ビット 1 = SEPOBJ、ビット 0 = NOSEPOBJ (Windows および AIX のみ)</p> <p>.... 1... ビット 1 = NAME、ビット 0 = NONAME</p> <p>.... .1.. ビット 1 = OBJECT、ビット 0 = NOOBJECT</p> <p>.... ..1. ビット 1 = SQL、ビット 0 = NOSQL</p> <p>.... ...1 ビット 1 = CICS、ビット 0 = NOCICS</p>
オプション・バイト 2	XL1	<p>1... ビット 1 = OFFSET、ビット 0 = NOOFFSET</p> <p>.1.. ビット 1 = MAP、ビット 0 = NOMAP</p> <p>..1. ビット 1 = LIST、ビット 0 = NOLIST</p> <p>...1 ビット 1 = DBCSXREF、ビット 0 = NODBCSXREF</p> <p>.... 1... ビット 1 = XREF(SHORT)、ビット 0 = XREF(SHORT) ではない。このフラグは、ビット 7 のフラグと組み合わせて使用します。このフラグによって、XREF(FULL) はオフ状態で示され、ビット 7 のフラグはオン状態で示されます。</p> <p>.... .1.. ビット 1 = SOURCE、ビット 0 = NOSOURCE</p> <p>.... ..1. ビット 1 = VBREF、ビット 0 = NOVBREF</p> <p>.... ...1 ビット 1 = XREF、ビット 0 = XREF ではない。ビット 4 以降のフラグも参照。</p>

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 3	XL1	<p>1... .. ビット 1 = 指定される FLAG 組み込み診断レベル (FLAG(x,y) のように値 y が指定されます)</p> <p>.1.. ビット 1 = FLAGSTD、ビット 0 = NOFLAGSTD</p> <p>..1. ビット 1 = NUM、ビット 0 = NONUM</p> <p>...1 ビット 1 = SEQUENCE、ビット 0 = NOSEQUENCE</p> <p>.... 1... ビット 1 = SOSI、ビット 0 = NOSOSI (Windows および AIX のみ)</p> <p>.... .1.. ビット 1 = NSYMBOL(NATIONAL)、ビット 0 = NSYMBOL(DBCS)</p> <p>.... ..1. ビット 1 = PROFILE、ビット 0 = NOPROFILE (AIX のみ)</p> <p>.... ...1 ビット 1 = WORD、ビット 0 = NOWORD</p>
オプション・バイト 4	XL1	<p>1... .. ビット 1 = ADV、ビット 0 = NOADV</p> <p>.1.. ビット 1 = APOST、ビット 0 = QUOTE</p> <p>..1. ビット 1 = DYNAM、ビット 0 = NODYNAM</p> <p>...1 ビット 1 = AWO、ビット 0 = NOAWO</p> <p>.... 1... ビット 1 = 指定済み RMODE、ビット 0 = RMODE(AUTO)</p> <p>.... .1.. ビット 1 = RENT、ビット 0 = NORENT</p> <p>.... ..1. ビット 1 = RES: COBOL の場合、このフラグは常にオンに設定されます。</p> <p>.... ...1 ビット 1 = RMODE(24)、ビット 0 = RMODE(ANY)</p>

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 5	XL1	<p>1... ビット 1 = SQLCCSID、ビット 0 = NOSQLCCSID</p> <p>.1.. ビット 1 = OPT、ビット 0 = NOOPT</p> <p>..1. ビット 1 = LIB、ビット 0 = NOLIB</p> <p>...1 ビット 1 = DBCS、ビット 0 = NODBCS</p> <p>.... 1... ビット 1 = OPT(FULL)、ビット 0 = OPT(FULL) ではない</p> <p>.... .1.. ビット 1 = SSRANGE、ビット 0 = NOSSRANGE</p> <p>.... ..1. ビット 1 = TEST、ビット 0 = NOTEST</p> <p>.... ...1 ビット 1 = PROBE、ビット 0 = NOPROBE (Windows のみ)</p>
オプション・バイト 6	XL1	<p>..1. ビット 1 = NUMPROC(PFD)、ビット 0 = NUMPROC(NOPFD)</p> <p>...1 ビット 1 = NUMCLS(ALT)、ビット 0 = NUMCLS(PRIM)</p> <p>.... .1.. ビット 1 = BINARY(S390)、ビット 0 = BINARY(NATIVE) (Windows および AIX のみ)</p> <p>.... ..1. ビット 1 = TRUNC(STD)、ビット 0 = TRUNC(OPT)</p> <p>.... ...1 ビット 1 = ZWB、ビット 0 = NOZWB</p> <p>11.. 1... 将来の利用のために予約済み</p>

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 7	XL1	<p>1... ビット 1 = ALLOWCBL、ビット 0 = NOALLOWCBL</p> <p>.1.. ビット 1 = TERM、ビット 0 = NOTERM</p> <p>..1. 1 = DUMP、ビット 0 = NODUMP</p> <p>.... ..1. ビット 1 = CURRENCY、ビット 0 = NOCURRENCY</p> <p>...1 11.1 将来の利用のために予約済み</p>
オプション・バイト 8	XL1	<p>1... ビット 1 = XMLPARSE(XMLSS)、ビット 0 = XMLPARSE(COMPAT)</p> <p>.1.. ビット 1 = OPTFILE、ビット 0 = OPTFILE 以外</p> <p>..1. ビット 1 = ADDR(64)、ビット 0 = ADDR(32)</p> <p>...1 1111 将来の利用のために予約済み</p>
オプション・バイト 9	XL1	<p>1... ビット 1 = DATA(24)、ビット 0 = DATA(31)</p> <p>.1.. ビット 1 = FASTSRT、ビット 0 = NOFASTSRT</p> <p>..1. ビット 1 = SIZE(MAX)、ビット 0 = SIZE(nnnn) または SIZE(nnnnK)</p> <p>.... ..1. ビット 1 = THREAD、ビット 0 = NOTHREAD</p> <p>...1 1.11 将来の利用のために予約済み</p>
オプション・バイト A	XL1	<p>1111 1111 将来の利用のために予約済み</p>
オプション・バイト B	XL1	<p>1111 1111 将来の利用のために予約済み</p>

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト C	XL1	<p>1... ビット 1 = NCOLLSEQ(LOCALE) (Windows および AIX のみ)</p> <p>.1.. 将来の利用のために予約済み</p> <p>..1. ビット 1 = INTDATE(LILIAN)、ビット 0 = INTDATE(ANSI)</p> <p>...1 ビット 1 = NCOLLSEQ(BINARY) (Windows および AIX のみ)</p> <p>.... 1... ビット 1 = CHAR(EBCDIC)、ビット 0 = CHAR(NATIVE) (Windows および AIX のみ)</p> <p>.... .1.. ビット 1 = FLOAT(HEX)、ビット 0 = FLOAT(NATIVE) (Windows および AIX のみ)</p> <p>.... ..1. ビット 1 = COLLSEQ(BINARY) (Windows および AIX のみ)</p> <p>.... ...1 ビット 1 = COLLSEQ(LOCALE) (Windows および AIX のみ)</p>

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト D	XL1	<p>1... .. ビット 1 = DLL、ビット 0 = NODLL</p> <p>.1.. ビット 1 = EXPORTALL、ビット 0 = NOEXPORTALL</p> <p>..1. ビット 1 = CODEPAGE</p> <p>...1 ビット 1 = DATEPROC、ビット 0 = NODATEPROC</p> <p>.... 1... ビット 1 = DATEPROC(FLAG)、ビット 0 = DATEPROC(NOFLAG)</p> <p>.... .1.. ビット 1 = YEARWINDOW</p> <p>.... ..1. ビット 1 = WSCLEAR、ビット 0 = NOWSCLEAR (Windows および AIX のみ)</p> <p>.... ...1 ビット 1 = BEOPT、ビット 0 = NOBEOPT (Windows および AIX のみ)</p>
オプション・バイト E	XL1	<p>1... .. ビット 1 = DATEPROC(TRIG)、ビット 0 = DATEPROC(NOTRIG)</p> <p>.1.. ビット 1 = DIAGTRUNC、ビット 0 = NODIAGTRUNC</p> <p>.... .1.. ビット 1 = LSTFILE(UTF-8)、ビット 0 = LSTFILE(LOCALE) (Windows および AIX のみ)</p> <p>.... ..1. ビット 1 = MDECK、ビット 0 = NOMDECK</p> <p>.... ...1 ビット 1 = MDECK(NOCOMPILE)</p> <p>..11 1... 将来の利用のために予約済み</p>
オプション・バイト F	XL1	<p>1111 1111 将来の利用のために予約済み</p>

|
|
|
|
|

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
フラグ・レベル	XL1	X'00' フラグ (I) X'04' フラグ (W) X'08' フラグ (E) X'0C' フラグ (S) X'10' フラグ (U) X'FF' Noflag
組み込み診断レベル	XL1	X'00' フラグ (I) X'04' フラグ (W) X'08' フラグ (E) X'0C' フラグ (S) X'10' フラグ (U) X'FF' Noflag
FLAGSTD (FIPS) 指定	XL1	1... .. 最小 .1.. .. 中間 ..1. 高 ...1 IBM 拡張 1... レベル 1 セグメンテーション 1.. レベル 2 セグメンテーション 1. デバッグ 1 廃止
フラグ用に予約済み	XL1	1111 1111 将来の利用のために予約済み
コンパイラー・モード	XL1	X'00' 無条件 Nocompile、Nocompile(I) X'04' Nocompile(W) X'08' Nocompile(E) X'0C' Nocompile(S) X'FF' コンパイル
スペース値	CL1	

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
3 値オプション用のデータ	XL1	<p>1... NAME(ALIAS) 指定</p> <p>.1.. NUMPROC(MIG) 指定</p> <p>..1. TRUNC(BIN) 指定</p> <p>...1 1111 将来の利用のために予約済み</p>
TEST サブオプション	XL1	<p>1... TEST(HOOK)</p> <p>.1.. TEST(SEP)</p> <p>..1. TEST(EJPD)</p> <p>...1 1111 TEST サブオプションのために予約済み</p>
OUTDD 名長	HL2	OUTDD 名の長さ
RWT ID 長	HL2	予約語テーブル ID の長さ
LVLINFO	CL4	ユーザー指定 LVLINFO データ
PGMNAME サブオプション	XL1	<p>1... ビット 1 = PGMNAME(COMPAT)</p> <p>.1.. ビット 1 = PGMNAME(LONGUPPER)</p> <p>..1. ビット 1 = PGMNAME(LONGMIXED)</p> <p>...1 1111 将来の利用のために予約済み</p>
記入項目インターフェース・サブオプション	XL1	<p>1... ビット 1 = EntryInterface(System) (Windows のみ)</p> <p>.1.. ビット 1 = EntryInterface(OptLink) (Windows のみ)</p> <p>..11 1111 将来の利用のために予約済み</p>

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
CallInterface サブオプション	XL1	1... .. ビット 1 = CallInterface(System) (Windows および AIX のみ) .1.. .. ビット 1 = CallInterface(OptLink) (Windows のみ) ...1 .. ビット 1 = CallInterface(Cdecl) (Windows のみ) 1... ビット 1 = CallInterface(System(Desc)) (Windows および AIX のみ) ..1. .111 将来の利用のために予約済み
ARITH サブオプション	XL1	1... .. ビット 1 = ARITH(COMPAT) .1.. .. ビット 1 = ARITH(EXTEND) 11 1111 将来の利用のために予約済み
DBCS 要件	FL4	DBCS XREF ストレージ要件
DBCS ORDPGM 長	HL2	DBCS 配列プログラムの名前の長さ
DBCS ENCTBL 長	HL2	DBCS エンコード・テーブルの名前の長さ
DBCS ORD TYPE	CL2	DBCS 配列型
予約済み	CL6	将来の利用のために予約済み
変換済み SO	CL1	変換済み SO 16 進数値
変換済み SI	CL1	変換済み SI 16 進数値
言語 ID	CL2	このフィールドには、LANGUAGE オプションからの 2 文字の省略形 (EN、UE、JA、または JP のいずれか) が保持されます。
予約済み	CL8	将来の利用のために予約済み
INEXIT 名長	HL2	SYSIN ユーザー出口名の長さ
PRTEXIT 名長	HL2	SYSPRINT ユーザー出口名の長さ
LIBEXIT 名長	HL2	'Library' ユーザー出口名の長さ
ADEXIT 名長	HL2	ADATA ユーザー出口名の長さ
CURROPT	CL5	CURRENCY オプション値
予約済み	CL1	将来の利用のために予約済み
YEARWINDOW	HL2	YEARWINDOW オプション値
CODEPAGE	HL2	CODEPAGE CCSID オプション値
予約済み	CL50	将来の利用のために予約済み
LINECNT	HL2	LINECOUNT 値
予約済み	CL2	将来の利用のために予約済み

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
BUFSIZE	FL4	BUFSIZE オプション値
サイズ値	FL4	SIZE オプション値
予約済み	FL4	将来の利用のために予約済み
フェーズ常駐ビット バイト 1	XL1	1... .. ビット 1 = ユーザー領域内の IGYCLIBR .1.. ビット 1 = ユーザー領域内の IGYCSCAN ..1. ビット 1 = ユーザー領域内の IGYCDSCN ...1 ビット 1 = ユーザー領域内の IGYCGROU 1... ビット 1 = ユーザー領域内の IGYCPSCN 1.. ビット 1 = ユーザー領域内の IGYCPANA 1. ビット 1 = ユーザー領域内の IGYCFGEN 1 ビット 1 = ユーザー領域内の IGYCPGEN
フェーズ常駐ビット バイト 2	XL1	1... .. ビット 1 = ユーザー領域内の IGYCOPTM .1.. ビット 1 = ユーザー領域内の IGYCLSTR ..1. ビット 1 = ユーザー領域内の IGYCXREF ...1 ビット 1 = ユーザー領域内の IGYCDMAP 1... ビット 1 = ユーザー領域内の IGYCASM1 1.. ビット 1 = ユーザー領域内の IGYCASM2 1. ビット 1 = ユーザー領域内の IGYCDIAG 1 将来の利用のために予約済み
フェーズ常駐ビット バイト 3 および 4	XL2	予約済み
予約済み	CL8	将来の利用のために予約済み
OUTDD 名	CL(n)	OUTDD 名
RWT	CL(n)	予約語テーブル ID
DBCS ORDPGM	CL(n)	DBCS 配列プログラム名

表 99. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
DBCS ENCTBL	CL(n)	DBCS エンコード・テーブル名
INEXIT 名	CL(n)	SYSIN ユーザー出口名
PRTEXIT 名	CL(n)	SYSPRINT ユーザー出口名
LIBEXIT 名	CL(n)	'Library' ユーザー出口名
ADEXIT 名	CL(n)	ADATA ユーザー出口名

外部シンボル・レコード: X'0020'

次の表に、外部シンボル・レコードの内容を示します。

表 100. SYSADATA 外部シンボル・レコード

フィールド	サイズ	説明
セクション・タイプ	XL1	<p>X'00' PROGRAM-ID 名 (メインエントリー・ポイント名)</p> <p>X'01' ENTRY 名 (2 次エントリー・ポイント名)</p> <p>X'02' 外部参照 (参照先の外部エントリー・ポイント)</p> <p>X'04' COBOL の場合は適用外</p> <p>X'05' COBOL の場合は適用外</p> <p>X'06' COBOL の場合は適用外</p> <p>X'0A' COBOL の場合は適用外</p> <p>X'12' 内部参照 (参照先の内部サブプログラム)</p> <p>X'C0' 外部クラス名 (OO COBOL クラス定義)</p> <p>X'C1' METHOD-ID 名 (OO COBOL メソッド定義)</p> <p>X'C6' メソッド参照 (OO COBOL メソッド参照)</p> <p>X'FF' COBOL の場合は適用外</p> <p>タイプ X'12'、X'C0'、X'C1'、および X'C6' は、COBOL のみ対応です。</p>
フラグ	XL1	COBOL の場合は適用外
予約済み	HL2	将来の利用のために予約済み
記号 ID	FL4	参照を含むプログラムの記号 ID (タイプ x'02' および x'12' の場合のみ)
行番号	FL4	参照を含むステートメントの行番号 (タイプ x'02' および x'12' の場合のみ)
セクション長	FL4	COBOL の場合は適用外
LD ID	FL4	COBOL の場合は適用外
予約済み	CL8	将来の利用のために予約済み
外部名の長さ	HL2	外部名の文字数
別名の長さ	HL2	COBOL の場合は適用外
外部名	CL(n)	外部名

表 100. SYSADATA 外部シンボル・レコード (続き)

フィールド	サイズ	説明
別名セクション名	CL(n)	COBOL の場合は適用外

構文解析ツリー・レコード: X'0024'

次の表に、構文解析ツリー・レコードの内容を示します。

表 101. SYSADATA 構文解析ツリー・レコード

フィールド	サイズ	説明
ノード番号	FL4	コンパイラーが生成するノード番号。1 から開始
ノード・タイプ	HL2	ノードのタイプ: 001 プログラム 002 クラス 003 メソッド
		101 見出し部 102 環境部 103 データ部 104 手続き部 105 終了プログラム/メソッド/クラス
		201 宣言本文 202 非宣言本文
		301 セクション 302 プロシージャー・セクション
		401 段落 402 手順段落
		501 文 502 ファイル定義 503 ソート・ファイル定義 504 プログラム名 505 プログラム属性 508 ENVIRONMENT DIVISION 節 509 CLASS 属性 510 METHOD 属性 511 USE ステートメント

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		<p>601 ステートメント</p> <p>602 データ記述節</p> <p>603 データ入力項目</p> <p>604 ファイル記述節</p> <p>605 データ入力項目名</p> <p>606 データ入力項目レベル</p> <p>607 EXEC 記入項目</p>
		<p>701 EVALUATE サブジェクト句</p> <p>702 EVALUATE WHEN 句</p> <p>703 EVALUATE WHEN OTHER 句</p> <p>704 SEARCH WHEN 句</p> <p>705 INSPECT CONVERTING 句</p> <p>706 INSPECT REPLACING 句</p> <p>707 INSPECT TALLYING 句</p> <p>708 PERFORM UNTIL 句</p> <p>709 PERFORM VARYING 句</p> <p>710 PERFORM AFTER 句</p> <p>711 ステートメント・ブロック</p> <p>712 範囲終了符号</p> <p>713 INITIALIZE REPLACING 句</p> <p>714 EXEC CICS コマンド</p> <p>720 DATA DIVISION 句</p>
		<p>801 句</p> <p>802 ON 句</p> <p>803 NOT 句</p> <p>804 THEN 句</p> <p>805 ELSE 句</p> <p>806 条件</p> <p>807 式</p> <p>808 相対索引付け</p> <p>809 EXEC CICS オプション</p> <p>810 予約語</p> <p>811 INITIALIZE REPLACING カテゴリー</p>

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		<p>901 セクションまたは段落名</p> <p>902 ID</p> <p>903 英字名</p> <p>904 クラス名</p> <p>905 条件名</p> <p>906 ファイル名</p> <p>907 指標名</p> <p>908 簡略名</p> <p>910 シンボリック文字</p> <p>911 リテラル</p> <p>912 関数 ID</p> <p>913 データ名</p> <p>914 特殊レジスター</p> <p>915 プロシージャー参照</p> <p>916 算術演算子</p> <p>917 全プロシージャー</p> <p>918 INITIALIZE リテラル (トークンなし)</p> <p>919 ALL リテラルまたは形象定数</p> <p>920 キーワード・クラス・テスト名</p> <p>921 ID レベルの予約語</p> <p>922 単項演算子</p> <p>923 関係演算子</p>
		<p>1001 添え字</p> <p>1002 参照変更</p>
ノード・サブタイプ	HL2	<p>ノードのサブタイプ。</p> <p>セクション・タイプの場合:</p> <p>0001 CONFIGURATION セクション</p> <p>0002 INPUT-OUTPUT セクション</p> <p>0003 FILE セクション</p> <p>0004 WORKING-STORAGE セクション</p> <p>0005 LINKAGE セクション</p> <p>0006 LOCAL-STORAGE セクション</p> <p>0007 REPOSITORY セクション</p>

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		段落タイプの場合: 0001 PROGRAM-ID 段落 0002 AUTHOR 段落 0003 INSTALLATION 段落 0004 DATE-WRITTEN 段落 0005 SECURITY 段落 0006 SOURCE-COMPUTER 段落 0007 OBJECT-COMPUTER 段落 0008 SPECIAL-NAMES 段落 0009 FILE-CONTROL 段落 0010 I-O-CONTROL 段落 0011 DATE-COMPILED 段落 0012 CLASS-ID 段落 0013 METHOD-ID 段落 0014 REPOSITORY 段落
		環境部節タイプの場合: 0001 WITH DEBUGGING MODE 0002 MEMORY-SIZE 0003 SEGMENT-LIMIT 0004 CURRENCY-SIGN 0005 DECIMAL POINT 0006 PROGRAM COLLATING SEQUENCE 0007 ALPHABET 0008 SYMBOLIC-CHARACTER 0009 CLASS 0010 ENVIRONMENT NAME 0011 SELECT

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		データ記述節タイプの場合:
		0001 BLANK WHEN ZERO
		0002 DATA-NAME OR FILLER
		0003 JUSTIFIED
		0004 OCCURS
		0005 PICTURE
		0006 REDEFINES
		0007 RENAMES
		0008 SIGN
		0009 SYNCHRONIZED
		0010 USAGE
		0011 VALUE
		0023 GLOBAL
		0024 EXTERNAL

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		ファイル記述節タイプの場合:
		0001 FILE STATUS
		0002 ORGANIZATION
		0003 ACCESS MODE
		0004 RECORD KEY
		0005 ASSIGN
		0006 RELATIVE KEY
		0007 PASSWORD
		0008 PROCESSING MODE
		0009 RECORD DELIMITER
		0010 PADDING CHARACTER
		0011 BLOCK CONTAINS
		0012 RECORD CONTAINS
		0013 LABEL RECORDS
		0014 VALUE OF
		0015 DATA RECORDS
		0016 LINAGE
		0017 ALTERNATE KEY
		0018 LINES AT TOP
		0019 LINES AT BOTTOM
		0020 CODE-SET
		0021 RECORDING MODE
		0022 RESERVE
		0023 GLOBAL
		0024 EXTERNAL
		0025 LOCK

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		ステートメント・タイプの場合:
		0002 NEXT SENTENCE
		0003 ACCEPT
		0004 ADD
		0005 ALTER
		0006 CALL
		0007 CANCEL
		0008 CLOSE
		0009 COMPUTE
		0010 CONTINUE
		0011 DELETE
		0012 DISPLAY
		0013 DIVIDE (INTO)
		0113 DIVIDE (BY)
		0014 ENTER
		0015 ENTRY
		0016 EVALUATE
		0017 EXIT
		0018 GO
		0019 GOBACK
		0020 IF
		0021 INITIALIZE
		0022 INSPECT

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		0023 INVOKE
		0024 MERGE
		0025 MOVE
		0026 MULTIPLY
		0027 OPEN
		0028 PERFORM
		0029 READ
		0030 READY
		0031 RELEASE
		0032 RESET
		0033 RETURN
		0034 REWRITE
		0035 SEARCH
		0036 SERVICE
		0037 SET
		0038 SORT
		0039 START
		0040 STOP
		0041 STRING
		0042 SUBTRACT
		0043 UNSTRING
		0044 EXEC SQL
		0144 EXEC CICS
		0045 WRITE
		0046 XML

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		句タイプの場合:
		0001 INTO
		0002 DELIMITED
		0003 INITIALIZE. . .REPLACING
		0004 INSPECT. . .ALL
		0005 INSPECT. . .LEADING
		0006 SET. . .TO
		0007 SET. . .UP
		0008 SET. . .DOWN
		0009 PERFORM. . .TIMES
		0010 DIVIDE. . .REMAINDER
		0011 INSPECT. . .FIRST
		0012 SEARCH. . .VARYING
		0013 MORE-LABELS
		0014 SEARCH ALL
		0015 SEARCH. . .AT END
		0016 SEARCH. . .TEST INDEX
		0017 GLOBAL
		0018 LABEL
		0019 DEBUGGING
		0020 SEQUENCE
		0021 将来の利用のために予約済み
		0022 将来の利用のために予約済み
		0023 将来の利用のために予約済み
		0024 TALLYING
		0025 将来の利用のために予約済み
		0026 ON SIZE ERROR
		0027 ON OVERFLOW
		0028 ON ERROR
		0029 AT END
		0030 INVALID KEY

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		0031 END-OF-PAGE
		0032 USING
		0033 BEFORE
		0034 AFTER
		0035 EXCEPTION
		0036 CORRESPONDING
		0037 将来の利用のために予約済み
		0038 RETURNING
		0039 GIVING
		0040 THROUGH
		0041 KEY
		0042 DELIMITER
		0043 POINTER
		0044 COUNT
		0045 METHOD
		0046 PROGRAM
		0047 INPUT
		0048 OUTPUT
		0049 I-O
		0050 EXTEND
		0051 RELOAD
		0052 ASCENDING
		0053 DESCENDING
		0054 DUPLICATES
		0055 NATIVE (USAGE)
		0056 INDEXED
		0057 FROM
		0058 FOOTING
		0059 LINES AT BOTTOM
		0060 LINES AT TOP
		0061 XML ENCODING
		0062 XML GENERATE XML-DECLARATION
		0063 XML GENERATE ATTRIBUTES
		0064 XML GENERATE NAMESPACE
		0065 XML PARSE PROCESSING

|
|
|
|
|

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		関数 ID タイプの場合:
		0001 COS
		0002 LOG
		0003 MAX
		0004 MIN
		0005 MOD
		0006 ORD
		0007 REM
		0008 SIN
		0009 SUM
		0010 TAN
		0011 ACOS
		0012 ASIN
		0013 ATAN
		0014 CHAR
		0015 MEAN
		0016 SQRT
		0017 LOG10
		0018 RANGE
		0019 LENGTH
		0020 MEDIAN
		0021 NUMVAL
		0022 RANDOM
		0023 ANNUITY
		0024 INTEGER
		0025 ORD-MAX
		0026 ORD-MIN
		0027 REVERSE
		0028 MIDRANGE
		0029 NUMVAL-C
		0030 VARIANCE
		0031 FACTORIAL
		0032 LOWER-CASE

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		0033 UPPER-CASE 0034 CURRENT-DATE 0035 INTEGER-PART 0036 PRESENT-VALUE 0037 WHEN-COMPILED 0038 DAY-OF-INTEGERS 0039 INTEGER-OF-DAY 0040 DATE-OF-INTEGERS 0041 INTEGER-OF-DATE 0042 STANDARD-DEVIATION 0043 YEAR-TO-YYYY 0044 DAY-TO-YYYYDDD 0045 DATE-TO-YYYYMMDD 0046 UNDATE 0047 DATEVAL 0048 YEARWINDOW 0049 DISPLAY-OF 0050 NATIONAL-OF
		特殊レジスタ・タイプの場合: 0001 ADDRESS OF 0002 LENGTH OF
		キーワード・クラス・テスト名タイプの場合: 0001 ALPHABETIC 0002 ALPHABETIC-LOWER 0003 ALPHABETIC-UPPER 0004 DBCS 0005 KANJI 0006 NUMERIC 0007 NEGATIVE 0008 POSITIVE 0009 ZERO

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		予約語タイプの場合: 0001 TRUE 0002 FALSE 0003 ANY 0004 THRU
		ID、データ名、索引名、条件名、または簡略名タイプの場合: 0001 REFERENCED 0002 CHANGED 0003 REFERENCED & CHANGED
		初期化リテラル・タイプの場合: 0001 ALPHABETIC 0002 ALPHANUMERIC 0003 NUMERIC 0004 ALPHANUMERIC-EDITED 0005 NUMERIC-EDITED 0006 DBCS/EGCS 0007 NATIONAL 0008 NATIONAL-EDITED
		プロシージャ名タイプの場合: 0001 SECTION 0002 PARAGRAPH
		ID レベルの予約語タイプの場合: 0001 ROUNDED 0002 TRUE 0003 ON 0004 オフ 0005 SIZE 0006 DATE 0007 DAY 0008 DAY-OF-WEEK 0009 TIME 0010 WHEN-COMPILED 0011 PAGE 0012 DATE YYYYMMDD 0013 DAY YYYYDDD

I

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		算術演算子タイプの場合: 0001 PLUS 0002 MINUS 0003 TIMES 0004 DIVIDE 0005 DIVIDE REMAINDER 0006 EXPONENTIATE 0007 NEGATE
		関係演算子タイプの場合: 0008 LESS 0009 LESS OR EQUAL 0010 EQUAL 0011 NOT EQUAL 0012 GREATER 0013 GREATER OR EQUAL 0014 AND 0015 OR 0016 CLASS CONDITION 0017 NOT CLASS CONDITION
親ノード番号	FL4	ノードの親のノード番号
左方兄弟ノード番号	FL4	ノードの左方兄弟のノード番号 (ある場合)。なしの場合、値はゼロ。
記号 ID	FL4	以下のタイプのいずれかのユーザー名の場合は、ノードの記号 ID。 <ul style="list-style-type: none"> • データ入力項目 • ID • ファイル名 • 指標名 • プロシージャ名。 • 条件名 • 簡略名 段落 ID に対応するプロシージャ名を除き、この値は記号 (タイプ 42) レコード内の記号 ID に対応します。 他のすべてのノード・タイプの場合、この値はゼロです。

表 101. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
セクション記号 ID	FL4	修飾段落名参照の場合は、ノードが含まれているセクションの記号 ID。この値は記号 (タイプ 42) レコード内のセクション ID に対応します。 他のすべてのノード・タイプの場合、この値はゼロです。
最初のトークン番号	FL4	ノードに関連付けられた最初のトークンの番号
最後のトークン番号	FL4	ノードに関連付けられた最後のトークンの番号
予約済み	FL4	将来の利用のために予約済み
フラグ	CL1	ノードに関する情報は、以下を参照してください。 X'80' 予約済み X'40' 生成されたノード、トークンなし
予約済み	CL3	将来の利用のために予約済み

トークン・レコード: X'0030'

コンパイラーは、コメント行として扱われる行のトークン・レコードを生成しません。そのような行としては、以下にリストするものがありますが、それらに限定されません。

- コメント行。これは、桁 7 にアスタリスク (*) またはスラッシュ (/) が指定されている行です。
- 以下のコンパイラー指示ステートメント:
 - *CBL (*CONTROL)
 - BASIS
 - >>CALLINT
 - COPY
 - DELETE
 - EJECT
 - INSERT
 - REPLACE
 - SKIP1
 - SKIP2
 - SKIP3
 - TITLE
- デバッグ行。これは桁 7 に D が指定されている行です (WITH DEBUGGING MODE が指定されていない場合)。

表 102. SYSADATA トークン・レコード

フィールド	サイズ	説明
トークン番号	FL4	コンパイラーが生成するソース・ファイル内のトークン番号。1 から開始。ソースにはコピーブックを組み込み済みです。

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
トークン・コード	HL2	<p>トークンの型 (ユーザー名、リテラル、予約語など)。 予約語の場合は、コンパイラ予約語テーブル値が使用されます。 PICTURE スtringの場合は、特別コード 0000 が使用されます。 継続されるトークンの各ピース (最後のピース以外) の場合は、特殊コード 3333 が使用されます。 その他の場合は、以下のコードが使用されます。</p> <p>0001 ACCEPT 0002 ADD 0003 ALTER 0004 CALL 0005 CANCEL 0007 CLOSE 0009 COMPUTE 0011 DELETE 0013 DISPLAY 0014 DIVIDE 0017 READY 0018 END-PERFORM 0019 ENTER 0020 ENTRY 0021 EXIT 0022 EXEC EXECUTE 0023 GO 0024 IF 0025 INITIALIZE 0026 INVOKE 0027 INSPECT 0028 MERGE 0029 MOVE</p>

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0030 MULTIPLY
		0031 OPEN
		0032 PERFORM
		0033 READ
		0035 RELEASE
		0036 RETURN
		0037 REWRITE
		0038 SEARCH
		0040 SET
		0041 SORT
		0042 START
		0043 STOP
		0044 STRING
		0045 SUBTRACT
		0048 UNSTRING
		0049 USE
		0050 WRITE
		0051 CONTINUE
		0052 END-ADD
		0053 END-CALL
		0054 END-COMPUTE
		0055 END-DELETE
		0056 END-DIVIDE
		0057 END-EVALUATE
		0058 END-IF
		0059 END-MULTIPLY
		0060 END-READ
		0061 END-RETURN
		0062 END-REWRITE
		0063 END-SEARCH
		0064 END-START
		0065 END-STRING
		0066 END-SUBTRACT
		0067 END-UNSTRING
		0068 END-WRITE
		0069 GOBACK

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0070 EVALUATE
		0071 RESET
		0072 SERVICE
		0073 END-INVOKE
		0074 END-EXEC
		0075 XML
		0076 END-XML
		0099 FOREIGN-VERB
		0101 DATA-NAME
		0105 DASHED-NUM
		0106 DECIMAL
		0107 DIV-SIGN
		0108 EQ
		0109 EXPONENTIATION
		0110 GT
		0111 INTEGER
		0112 LT
		0113 LPAREN
		0114 MINUS-SIGN
		0115 MULT-SIGN
		0116 NONUMLIT
		0117 PERIOD
		0118 PLUS-SIGN
		0121 RPAREN
		0122 SIGNED-INTEGERS
		0123 QUID
		0124 COLON
		0125 IEOF
		0126 EGCS-LIT
		0127 COMMA-SPACE
		0128 SEMICOLON-SPACE
		0129 PROCEDURE-NAME
		0130 FLT-POINT-LIT
		0131 言語環境プログラム

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0132 GE
		0133 IDREF
		0134 EXPREF
		0136 CICS
		0137 NEW
		0138 NATIONAL-LIT
		0200 ADDRESS
		0201 ADVANCING
		0202 AFTER
		0203 ALL
		0204 ALPHABETIC
		0205 ALPHANUMERIC
		0206 ANY
		0207 AND
		0208 ALPHANUMERIC-EDITED
		0209 BEFORE
		0210 BEGINNING
		0211 FUNCTION
		0212 CONTENT
		0213 CORR
		CORRESPONDING
		0214 DAY
		0215 DATE
		0216 DEBUG-CONTENTS
		0217 DEBUG-ITEM
		0218 DEBUG-LINE
		0219 DEBUG-NAME
		0220 DEBUG-SUB-1
		0221 DEBUG-SUB-2
		0222 DEBUG-SUB-3
		0223 DELIMITED
		0224 DELIMITER
		0225 DOWN

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0226 NUMERIC-EDITED
		0227 XML-EVENT
		0228 END-OF-PAGE
		EOP
		0229 EQUAL
		0230 ERROR
		0231 XML-NTEXT
		0232 EXCEPTION
		0233 EXTEND
		0234 FIRST
		0235 FROM
		0236 GIVING
		0237 GREATER
		0238 I-O
		0239 IN
		0240 INITIAL
		0241 INTO
		0242 INVALID
		0243 SQL
		0244 LESS
		0245 LINAGE-COUNTER
		0246 XML-TEXT
		0247 LOCK
		0248 GENERATE
		0249 NEGATIVE
		0250 NEXT
		0251 NO
		0252 NOT
		0253 NUMERIC
		0254 KANJI
		0255 OR
		0256 OTHER
		0257 OVERFLOW
		0258 PAGE
		0259 CONVERTING

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0260 POINTER
		0261 POSITIVE
		0262 DBCS
		0263 PROCEDURES
		0264 PROCEED
		0265 REFERENCES
		0266 DAY-OF-WEEK
		0267 REMAINDER
		0268 REMOVAL
		0269 REPLACING
		0270 REVERSED
		0271 REWIND
		0272 ROUNDED
		0273 RUN
		0274 SENTENCE
		0275 STANDARD
		0276 RETURN-CODE
		SORT-CORE-SIZE
		SORT-FILE-SIZE
		SORT-MESSAGE
		SORT-MODE-SIZE
		SORT-RETURN
		TALLY
		XML-CODE
		0277 TALLYING
		0278 SUM
		0279 TEST
		0280 THAN
		0281 UNTIL
		0282 UP
		0283 UPON
		0284 VARYING
		0285 RELOAD
		0286 TRUE

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0287 THEN
		0288 RETURNING
		0289 ELSE
		0290 SELF
		0291 SUPER
		0292 WHEN-COMPILED
		0293 ENDING
		0294 FALSE
		0295 REFERENCE
		0296 NATIONAL-EDITED
		0297 COM-REG
		0298 ALPHABETIC-LOWER
		0299 ALPHABETIC-UPPER
		0301 REDEFINES
		0302 OCCURS
		0303 SYNC
		SYNCHRONIZED
		0304 MORE-LABELS
		0305 JUST
		JUSTIFIED
		0306 SHIFT-IN
		0307 BLANK
		0308 VALUE
		0309 COMP
		COMPUTATIONAL
		0310 COMP-1
		COMPUTATIONAL-1
		0311 COMP-3
		COMPUTATIONAL-3
		0312 COMP-2
		COMPUTATIONAL-2
		0313 COMP-4
		COMPUTATIONAL-4
		0314 DISPLAY-1
		0315 SHIFT-OUT

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0316 INDEX
		0317 USAGE
		0318 SIGN
		0319 LEADING
		0320 SEPARATE
		0321 INDEXED
		0322 LEFT
		0323 RIGHT
		0324 PIC
		PICTURE
		0325 VALUES
		0326 GLOBAL
		0327 EXTERNAL
		0328 BINARY
		0329 PACKED-DECIMAL
		0330 EGCS
		0331 PROCEDURE-POINTER
		0332 COMP-5
		COMPUTATIONAL-5
		0333 FUNCTION-POINTER
		0334 TYPE
		0335 JNIEVPTR
		0336 NATIONAL
		0337 GROUP-USAGE
		0401 HIGH-VALUE
		HIGH-VALUES
		0402 LOW-VALUE
		LOW-VALUES
		0403 QUOTE
		QUOTES
		0404 SPACE
		SPACES
		0405 ZERO

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0406 ZEROES
		ZEROS
		0407 NULL
		NULLS
		0501 BLOCK
		0502 BOTTOM
		0505 CHARACTER
		0506 CODE
		0507 CODE-SET
		0514 FILLER
		0516 FOOTING
		0520 LABEL
		0521 LENGTH
		0524 LINAGE
		0526 OMITTED
		0531 RENAMES
		0543 TOP
		0545 TRAILING
		0549 RECORDING
		0601 INHERITS
		0603 RECURSIVE
		0701 ACCESS
		0702 ALSO
		0703 ALTERNATE
		0704 AREA
		AREAS
		0705 ASSIGN
		0707 COLLATING
		0708 COMMA
		0709 CURRENCY
		0710 CLASS
		0711 DECIMAL-POINT
		0712 DUPLICATES
		0713 DYNAMIC
		0714 EVERY

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0716 MEMORY
		0717 MODE
		0718 MODULES
		0719 MULTIPLE
		0720 NATIVE
		0721 オフ
		0722 OPTIONAL
		0723 ORGANIZATION
		0724 POSITION
		0725 PROGRAM
		0726 RANDOM
		0727 RELATIVE
		0728 RERUN
		0729 RESERVE
		0730 SAME
		0731 SEGMENT-LIMIT
		0732 SELECT
		0733 SEQUENCE
		0734 SEQUENTIAL
		0736 SORT-MERGE
		0737 STANDARD-1
		0738 TAPE
		0739 WORDS
		0740 PROCESSING
		0741 APPLY
		0742 WRITE-ONLY
		0743 COMMON
		0744 ALPHABET
		0745 PADDING
		0746 SYMBOLIC
		0747 STANDARD-2
		0748 OVERRIDE
		0750 PASSWORD

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0801 ARE
		IS
		0802 ASCENDING
		0803 AT
		0804 BY
		0805 CHARACTERS
		0806 CONTAINS
		0808 COUNT
		0809 DEBUGGING
		0810 DEPENDING
		0811 DESCENDING
		0812 DIVISION
		0814 FOR
		0815 ORDER
		0816 INPUT
		0817 REPLACE
		0818 KEY
		0819 LINE
		LINES
		0821 OF
		0822 ON
		0823 OUTPUT
		0825 RECORD
		0826 RECORDS
		0827 REEL
		0828 SECTION
		0829 SIZE
		0830 STATUS
		0831 THROUGH
		THRU
		0832 TIME
		0833 TIMES
		0834 TO
		0836 UNIT

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0837 USING
		0838 WHEN
		0839 WITH
		0901 PROCEDURE
		0902 DECLARATIVES
		0903 END
		1001 DATA
		1002 FILE
		1003 FD
		1004 SD
		1005 WORKING-STORAGE
		1006 LOCAL-STORAGE
		1007 LINKAGE
		1101 ENVIRONMENT
		1102 CONFIGURATION
		1103 SOURCE-COMPUTER
		1104 OBJECT-COMPUTER
		1105 SPECIAL-NAMES
		1106 REPOSITORY
		1107 INPUT-OUTPUT
		1108 FILE-CONTROL
		1109 I-O-CONTROL
		1201 ID
		IDENTIFICATION
		1202 PROGRAM-ID
		1203 AUTHOR
		1204 INSTALLATION
		1205 DATE-WRITTEN
		1206 DATE-COMPILED
		1207 SECURITY
		1208 CLASS-ID
		1209 METHOD-ID
		1210 METHOD
		1211 FACTORY

表 102. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		1212 OBJECT
		2020 TRACE
		3000 DATADEF
		3001 F-NAME
		3002 UPSI-SWITCH
		3003 CONDNAME
		3004 CONDVAR
		3005 BLOB
		3006 CLOB
		3007 DBCLOB
		3008 BLOB-LOCATOR
		3009 CLOB-LOCATOR
		3010 DBCLOB-LOCATOR
		3011 BLOB-FILE
		3012 CLOB-FILE
		3013 DBCLOB-FILE
		3014 DFHRESP
		5001 PARSE
		5002 AUTOMATIC
		5003 PREVIOUS
		9999 COBOL
トークン長	HL2	トークンの長さ
トークン列	FL4	ソース・リスト内のトークンの開始列番号
トークン行	FL4	ソース・リスト内のトークンの行番号
フラグ	CL1	<p>トークンに関する情報は、以下を参照してください。</p> <p>X'80' トークンは継続</p> <p>X'40' 継続されるトークンの最後のピース</p> <p>PICTURE スtringの場合は、ソース・トークンが継続されても、1 つのトークン・レコードしか生成されません。トークン・コード 0000、最初のピースのトークン列と行、完全Stringの長さ、継続なしフラグ・セット、完全Stringのトークン・テキストが与えられます。</p>
予約済み	CL7	将来の利用のために予約済み
トークン・テキスト	CL(n)	実トークン・String

ソース・エラー・レコード: X'0032'

次の表に、ソース・エラー・レコードの内容を示します。

表 103. SYSADATA ソース・エラー・レコード

フィールド	サイズ	説明
ステートメント番号	FL4	エラーのあるステートメントのステートメント番号
エラー ID	CL16	エラー・メッセージ ID (左寄せ、ブランク埋め込み)
エラーの重大度	HL2	エラーの重大度
エラー・メッセージ長	HL2	エラー・メッセージ・テキストの長さ
行位置	XL1	FIPS メッセージに指定される行位置標識
予約済み	CL7	将来の利用のために予約済み
エラー・メッセージ	CL(n)	エラー・メッセージ・テキスト

ソース・レコード: X'0038'

次の表に、ソース・レコードの内容を示します。

表 104. SYSADATA ソース・レコード

フィールド	サイズ	説明
行番号	FL4	ソース・レコードのリスト行番号
入力レコード番号	FL4	現行入力ファイル内の入力ソース・レコード番号
1 次ファイル番号	HL2	このレコードが基本入力ファイルからの場合は、入力ファイルの割り当てシーケンス番号 (ジョブ識別レコード内の入力ファイル n フィールドを参照してください)。
ライブラリー・ファイル番号	HL2	このレコードが COPYIBASIS 入力ファイルからの場合は、ライブラリー入力ファイルの割り当てシーケンス番号。(ライブラリー・レコード内のメンバー・ファイル ID n フィールドを参照してください。)
予約済み	CL8	将来の利用のために予約済み
親レコード番号	FL4	親ソース・レコード番号。COPYIBASIS ステートメントのレコード番号となります。
親 1 次ファイル番号	HL2	このレコードの親が 1 次入力ファイルからの場合は、親ファイルの割り当てシーケンス番号 (ジョブ識別レコード内の入力ファイル n フィールドを参照してください)。
親ライブラリー割り当てファイル番号	HL2	このレコードの親が COPYIBASIS 入力ファイルからの場合は、親ライブラリー・ファイルの割り当てシーケンス番号。(ライブラリー・レコード内の COPY/BASIS メンバー・ファイル ID n フィールドを参照してください。)
予約済み	CL8	将来の利用のために予約済み
ソース・レコードの長さ	HL2	後に続く実ソース・レコードの長さ。
予約済み	CL10	将来の利用のために予約済み
ソース・レコード	CL(n)	

COPY REPLACING レコード: X'0039'

REPLACING アクションが実行されるごとに、1 つの COPY REPLACING タイプのレコードが出力されます。すなわち、REPLACING 句の *operand-1* がコピーブックのテキストと一致するごとに、COPY REPLACING TEXT レコードが書き込まれます。

次の表に、COPY REPLACING レコードの内容を示します。

表 105. SYSADATA COPY REPLACING レコード

フィールド	サイズ	説明
置き換えられるストリングの開始行番号	FL4	REPLACING の結果であるテキストの開始のリスト行番号
置き換えられるストリングの開始桁番号	FL4	REPLACING の結果であるテキストの開始のリスト桁番号
置き換えられるストリングの終了行番号	FL4	REPLACING の結果であるテキストの終了のリスト行番号
置き換えられるストリングの終了桁番号	FL4	REPLACING の結果であるテキストの終了のリスト桁番号
オリジナル・ストリングの開始行番号	FL4	REPLACING により変更されたテキストの開始のソース・ファイル行番号
オリジナル・ストリングの開始列番号	FL4	REPLACING により変更されたテキストの開始のソース・ファイル桁番号
オリジナル・ストリングの終了行番号	FL4	REPLACING により変更されたテキストの終了のソース・ファイル行番号
オリジナル・ストリングの終了列番号	FL4	REPLACING により変更されたテキストの終了のソース・ファイル桁番号

記号レコード: X'0042'

次の表に、記号レコードの内容を示します。

表 106. SYSADATA 記号レコード

フィールド	サイズ	説明
記号 ID	FL4	固有の記号 ID
行番号	FL4	記号が定義または宣言されるソース・レコードのリスト行番号
レベル	XL1	記号の真のレベル番号 (または構造内のデータ項目の相対レベル番号)。COBOL の場合、これは、01 から 49 の範囲、66 (RENAMES 項目の場合)、77、または 88 (条件項目の場合) になります。
修飾標識	XL1	X'00' 固有の名前。修飾は不要。 X'01' このデータ項目は修飾が必須。名前はプログラム内で固有ではありません。このフィールドが適用されるのは、このデータ項目がレベル 01 名ではない場合のみです。

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
記号タイプ	XL1	<p>X'68' クラス名 (クラス ID)</p> <p>X'58' メソッド名</p> <p>X'40' データ名</p> <p>X'20' プロシージャ名。</p> <p>X'10' 簡略名</p> <p>X'08' プログラム名</p> <p>X'81' 予約済み</p> <p>以下は、上記のタイプの ORed です (該当する場合)。</p> <p>X'04' 外部</p> <p>X'02' グローバル</p>
記号属性	XL1	<p>X'01' 数値</p> <p>X'02' 次のいずれかのクラスの基本文字:</p> <ul style="list-style-type: none"> • 英字 • 英数字 • DBCS • 国別 <p>X'03' グループ</p> <p>X'04' ポインター</p> <p>X'05' 指標データ項目</p> <p>X'06' 指標名</p> <p>X'07' 条件</p> <p>X'0F' ファイル</p> <p>X'10' ソート・ファイル</p> <p>X'17' クラス名 (リポジトリ)</p> <p>X'18' オブジェクト参照</p>

|
|
|
|
|

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
節	XL1	<p>記号定義で指定されている文節。</p> <p>数値 (X'01'), 基本文字 (X'02'), グループ (X'03'), ポインター (X'04'), 指標データ項目 (X'05'), またはオブジェクト参照 (X'18') の記号属性を持つ記号の場合:</p> <p>1... .. 値</p> <p>.1.. .. 索引付き</p> <p>..1. 再定義</p> <p>...1 名前変更</p> <p>.... 1... 発生</p> <p>.... .1.. Occurs キーあり</p> <p>.... ..1. ケースにより発生</p> <p>.... ...1 親で発生</p> <p>両方のファイル・タイプの場合:</p> <p>1... .. 選択</p> <p>.1.. .. 割り当て</p> <p>..1. 再実行</p> <p>...1 同一領域</p> <p>.... 1... 同一レコード域</p> <p>.... .1.. 記録モード</p> <p>.... ..1. 予約済み</p> <p>.... ...1 レコード</p>

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
		簡略名記号の場合:
		01 CSP
		02 C01
		03 C02
		04 C03
		05 C04
		06 C05
		07 C06
		08 C07
		09 C08
		10 C09
		11 C10
		12 C11
		13 C12
		14 S01
		15 S02
		16 S03
		17 S04
		18 S05
		19 CONSOLE
		20 SYSINISYSIPT
		22 SYSOUTISYSLSTISYSLIST
		24 SYSPUNCHISYSPCH
		26 UPSI-0
		27 UPSI-1
		28 UPSI-2
		29 UPSI-3
		30 UPSI-4
		31 UPSI-5
		32 UPSI-6
		33 UPSI-7
		34 AFP-5A

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
データ・フラグ 1	XL1	<p>両方のファイル・タイプの場合、および数値 (X'01'), 基本文字 (X'02'), グループ (X'03'), ポインター (X'04'), 指標データ項目 (X'05'), またはオブジェクト参照 (X'18') の記号属性を持つ記号の場合:</p> <p>1... 再定義</p> <p>.1.. 名前変更</p> <p>..1. 同期化</p> <p>...1 暗黙的に再定義</p> <p>.... 1... 日付フィールド</p> <p>.... .1.. 暗黙の再定義</p> <p>.... ..1. FILLER</p> <p>.... ...1 レベル 77</p>

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
データ・フラグ 2	XL1	数値 (X'01') の記号属性を持つ記号の場合: 1... .. 2 進数 .1.. .. 外部浮動小数点 (USAGE DISPLAY または USAGE NATIONAL の) ..1. 内部浮動小数点 ...1 圧縮 1... 外部 10 進数 (USAGE DISPLAY または USAGE NATIONAL の)1.. 負の位取り1. 数字編集 (USAGE DISPLAY または USAGE NATIONAL の)1 将来の利用のために予約済み 基本文字 (X'02') またはグループ (X'03') の記号属性を持つ記号の場合: 1... .. 英字 .1.. .. 英数字 ..1. 編集英数字 ...1 グループは独自の ODO オブジェクトを含む 1... DBCS 項目1.. グループ可変長1. EGCS 項目1 編集 EGCS

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
		<p>両方のファイル・タイプの場合:</p> <p>1... .. レコード内の ODO のオブジェクト</p> <p>.1.. レコード内の ODO のサブジェクト</p> <p>..1. 順次アクセス</p> <p>...1 ランダム・アクセス</p> <p>.... 1... 動的アクセス</p> <p>.... .1.. 位置指定モード</p> <p>.... ..1. レコード域</p> <p>.... ...1 将来の利用のために予約済み</p> <p>他のすべてのデータ型の場合、フィールドはゼロです。</p>
データ・フラグ 3	XL1	<p>両方のファイル・タイプの場合:</p> <p>1... .. すべてのレコードが同じ長さ</p> <p>.1.. 固定長</p> <p>..1. 可変長</p> <p>...1 未定義</p> <p>.... 1... スパン</p> <p>.... .1.. ブロック</p> <p>.... ..1. 書き込みのみ適用</p> <p>.... ...1 同一ソート・マージ領域</p> <p>他のすべてのデータ型の場合、フィールドはゼロです。</p>

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
ファイル編成	XL1	<p>両方のファイル・タイプの場合:</p> <p>1... .. QSAM</p> <p>.1... .. ASCII</p> <p>..1. 標準ラベル</p> <p>...1 ユーザー・ラベル</p> <p>.... 1... VSAM 順次</p> <p>.... .1.. VSAM 索引</p> <p>.... ..1. VSAM 相対</p> <p>.... ...1 行順次</p> <p>他のすべてのデータ型の場合、フィールドはゼロです。</p>
USAGE 節	FL1	<p>X'00' USAGE IS DISPLAY</p> <p>X'01' USAGE IS COMP-1</p> <p>X'02' USAGE IS COMP-2</p> <p>X'03' USAGE IS PACKED-DECIMAL または USAGE IS COMP-3</p> <p>X'04' USAGE IS BINARY、USAGE IS COMP、または USAGE IS COMP-4</p> <p>X'05' USAGE IS DISPLAY-1</p> <p>X'06' USAGE IS POINTER</p> <p>X'07' USAGE IS INDEX</p> <p>X'08' USAGE IS PROCEDURE-POINTER</p> <p>X'09' USAGE IS OBJECT-REFERENCE</p> <p>X'0B' NATIONAL</p> <p>X'0A' FUNCTION-POINTER</p>
記号節	FL1	<p>X'00' SIGN 節なし</p> <p>X'01' SIGN IS LEADING</p> <p>X'02' SIGN IS LEADING SEPARATE CHARACTER</p> <p>X'03' SIGN IS TRAILING</p> <p>X'04' SIGN IS TRAILING SEPARATE CHARACTER</p>

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
標識	FL1	X'01' JUSTIFIED 節あり。右寄せ属性が有効。 X'02' BLANK WHEN ZERO 節あり。
サイズ	FL4	このデータ項目のサイズ。この項目がストレージで占有する実バイト数。DBCS 項目の場合、この数は、文字数ではなく、バイト数で示されます。可変長項目の場合、このフィールドには、コンパイラによってこの項目に予約されるストレージの最大サイズが反映されます。「長さ属性」とも呼ばれます。
精度	FL1	固定データ項目または浮動データ項目の精度
スケール	FL1	固定データ項目のスケール係数。小数点の右方の桁数。

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
ベース・ロケータ・タイプ	FL1	ホストの場合:
		01 ベース・ロケータ・ファイル
		02 ベース・ロケータ作業用ストレージ
		03 ベース・ロケータ・リンケージ・セクション
		05 ベース・ロケータ特殊レジスタ
		07 変数別索引付き
		09 COMREG 特殊レジスタ
		10 UPSI スイッチ
		13 Varloc 項目のベース・ロケータ
		14 外部データのベース・ロケータ
		15 ベース・ロケータ英数字 FUNC
		16 ベース・ロケータ英数字 EVAL
		17 オブジェクト・データのベース・ロケータ
		19 Local-Storage のベース・ロケータ
		20 ファクトリー・データ
		21 XML-TEXT および XML-NTEXT
		Windows および AIX の場合:
		01 ベース・ロケータ・ファイル
		02 ベース・ロケータ・リンケージ・セクション
		03 Varloc 項目のベース・ロケータ
		04 外部データのベース・ロケータ
		05 オブジェクト・データのベース・ロケータ
		06 XML-TEXT および XML-NTEXT
		10 ベース・ロケータ作業用ストレージ
		11 ベース・ロケータ特殊レジスタ
		12 ベース・ロケータ英数字 FUNC
13 ベース・ロケータ英数字 EVAL		
14 変数別索引付き		
16 COMREG 特殊レジスタ		
17 UPSI スイッチ		
18 ファクトリー・データ		
22 Local-Storage のベース・ロケータ		

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
日付形式	FL1	<p>日付形式は以下のとおりです。</p> <p>01 YY</p> <p>02 YYXX</p> <p>03 YYXXXX</p> <p>04 YYXXX</p> <p>05 YYYY</p> <p>06 YYYYXX</p> <p>07 YYYYXXXX</p> <p>08 YYYYXXX</p> <p>09 YYX</p> <p>10 YYYYX</p> <p>22 XXYY</p> <p>23 XXXXY</p> <p>24 XXXYY</p> <p>26 XXYYYY</p> <p>27 XXXXYYYY</p> <p>28 XXXYYYY</p> <p>29 XYY</p> <p>30 XYYYY</p>
データ・フラグ 4	XL1	<p>数値 (X'01') の記号属性を持つ記号の場合:</p> <p>1... 数値国別</p> <p>基本文字 (X'02') の記号属性を持つ記号の場合:</p> <p>1... 国別</p> <p>.1... 国別編集</p> <p>グループ (X'03') の記号属性を持つ記号の場合:</p> <p>1... 国別グループ (Group-Usage National)</p>
予約済み	FL3	将来の利用のために予約済み

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
住所情報	FL4	ホストの場合、ベース・ロケータ番号および変位は以下のとおりです。 ビット 0 から 4 未使用 ビット 5 から 19 ベース・ロケータ (BL) 番号 ビット 20 から 31 ベース・ロケータの変位 Windows および AIX の場合、W コード SymId。
構造変位	AL4	構造内の記号のオフセット。このオフセットは変数位置指定項目に 0 を設定します。
親変位	AL4	定義中の項目の即時親からのバイト・オフセット。
親 ID	FL4	定義中の項目の即時親の記号 ID。
再定義 ID	FL4	この項目が再定義するデータ項目の記号 ID (該当する場合)。
開始名前変更 ID	FL4	この項目がレベル 66 項目の場合は、この項目が名前変更した開始 COBOL データ項目の記号 ID。レベル 66 項目でない場合、このフィールドは 0 に設定されます。
終了名前変更 ID	FL4	この項目がレベル 66 項目の場合は、この項目が名前変更した終了 COBOL データ項目の記号 ID。レベル 66 項目でない場合、このフィールドは 0 に設定されます。
プログラム名記号 ID	FL4	プログラムのプログラム名の ID またはこの記号が定義されているクラスのクラス名。
OCCURS 最小 段落 ID	FL4	OCCURS の最小値 段落名の PROC 名 ID
OCCURS 最大 セクション ID	FL4	OCCURS の最大値 セクション名の PROC 名 ID
寸法	FL4	寸法の数
大/小文字ビット・ベクトル	XL4	シンボル名の文字の大/小文字は、1 文字につき 1 ビットで表現されます。それぞれのビットには次の意味がありません。 0 大文字 1 小文字 ビット 0 は、先頭文字の大/小文字を示します。ビット 1 は、2 番目の文字の大/小文字を示します。ビット 3 以降も同様です。
予約済み	CL8	将来の利用のために予約済み
値の組のカウント	HL2	値の組のカウント
記号名の長さ	HL2	記号名の文字数

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
データ名のピクチャー・データ長 または ファイル名の割り当て名の長さ	HL2	ピクチャー・データ内の文字の数: 関連した PICTURE 節を記号が持たない場合はゼロ。(PICTURE フィールドの長さ。) 長さは、ソース入力内で検出されたときのフィールドを表します。この長さは、複製係数を含んでいる PICTURE 項目の拡張フィールドを表すものではありません。PICTURE スtringの最大 COBOL 長は、50 バイトです。このフィールドがゼロの場合は、PICTURE が指定されていないことを示します。 これがファイル名の場合は、外部ファイル名の文字数。これは、割り当て名の DD 名部分です。ファイル名および ASSIGN USING が指定されている場合は、ゼロ。
データ名の初期値の長さ CLASS-ID の外部クラス名の長さ	HL2	記号値の文字の数。記号に初期値がない場合はゼロ。 CLASS-ID の外部クラス名の文字の数
データ名の ODO 記号名 ID ファイル名の場合、ASSIGN データ名の ID	FL4	データ名の場合、ODO 記号名の ID。ODO が指定されていない場合はゼロ。 ファイル名の場合、ASSIGN USING データ名の記号 ID。ASSIGN TO が指定されている場合はゼロ。
キー・カウント	HL2	定義されたキーの数
索引カウント	HL2	索引記号 ID のカウント。指定がない場合は、ゼロ
記号名	CL(n)	
データ名のピクチャー・データ・String または ファイル名の割り当て名	CL(n)	ユーザーが入力したとおりの PICTURE 文字String。文字Stringには、全記号、括弧、およびレプリカの生成係数が含まれます。 これがファイル名の場合は外部ファイル名。これは、割り当て名の DD 名部分です。
索引 ID リスト	(n)FL4	各索引記号名の ID
キー	(n)XL8	このフィールドには、配列に対して指定するキーを記述するデータが含まれます。以下の 3 つのフィールドは、「キー・カウント」フィールドに指定されている回数だけ繰り返されます。
...キー・シーケンス	FL1	昇順または降順の標識。 X'00' DESCENDING X'01' ASCENDING
...充てん文字	CL3	予約済み
...キー ID	FL4	配列内のキー・フィールドであるデータ項目の記号 ID

表 106. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
データ名の初期値データ CLASS-ID の外部クラス名	CL(n)	このフィールドには、この記号の INITIAL VALUE 節で指定するデータが含まれます。以下の 4 つの適切なサブフィールドは、「値の組カウント」フィールド内のカウントに応じて繰り返されます。このフィールドのデータの全長は、「初期値の長さ」フィールドに入れられます。 CLASS-ID の外部クラス名。
..第 1 値の長さ	HL2	最初の値の長さ
..第 1 値のデータ	CL(n)	最初の値。 このフィールドには、ソース・ファイルの VALUE 節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始/終了区切り文字、組み込み引用符、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、そのワードに関連付けられた値ではなく、実際の予約語が入ります。
..第 2 値の長さ	HL2	2 番目の値の長さ — THRU 値の組でない場合は、ゼロ
..第 2 値のデータ	CL(n)	2 番目の値。 このフィールドには、ソース・ファイルの VALUE 節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始/終了区切り文字、組み込み引用符、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、そのワードに関連付けられた値ではなく、実際の予約語が入ります。

記号相互参照レコード: X'0044'

次の表に、記号相互参照レコードの内容を示します。

表 107. SYSADATA 記号相互参照レコード

フィールド	サイズ	説明
記号長	HL2	記号の長さ
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号 VERB XREF の場合のみ: 動詞カウント。この動詞への参照の合計数。
参照の数 ¹	HL2	このレコード内の、後に続く記号への参照の数

表 107. SYSADATA 記号相互参照レコード (続き)

フィールド	サイズ	説明
相互参照タイプ	XL1	X'01' プログラム X'02' プロシージャ X'03' 動詞。 X'04' 記号またはデータ名 X'05' メソッド X'06' クラス
予約済み	CL7	将来の利用のために予約済み
記号名	CL(n)	記号。可変長。
...参照フラグ	CL1	記号またはデータ名参照の場合: C' ' ブランクは参照のみの意味 C'M' 修正参照フラグ プロシージャ型記号参照の場合: C'A' ALTER (procedure-name) C'D' GO TO (procedure-name) DEPENDING ON C'E' (PERFORM) から (procedure-name) までの範囲の終わり C'G' GO TO (procedure-name) C'P' PERFORM (procedure-name) C'T' (ALTER) TO PROCEED TO (procedure-name) C'U' デバッグ用に使用 (procedure-name)
...ステートメント番号	XL4	記号または動詞が参照されているステートメント番号
<p>1. 参照フラグ・フィールドおよびステートメント番号フィールドは、参照フィールドで指示される回数分だけ、発生します。例えば、参照フィールドの数に 10 の値が入っているとき、データ名、プロシージャ、またはプログラム式シンボルの場合には、参照フラグおよびステートメント番号ペアは 10 回発生し、動詞の場合には、ステートメント番号が 10 回発生します。</p> <p>参照数が、SYSADATA ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。レコードの共通ヘッダー・セクションに、継続フラグが設定されます。</p>		

ネストされたプログラム・レコード: X'0046'

次の表に、ネストされたプログラム・レコードの内容を示します。

表 108. SYSADATA ネストされたプログラム・レコード

フィールド	サイズ	説明
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号
ネスト・レベル	XL1	プログラム・ネスト・レベル

表 108. SYSADATA ネストされたプログラム・レコード (続き)

フィールド	サイズ	説明
プログラム属性	XL1	1... .. 初期 .1... .. 共通 ...1. PROCEDURE DIVISION using ...1 1111 将来の利用のために予約済み
予約済み	XL1	将来の利用のために予約済み
プログラム名の長さ	XL1	次のフィールドの長さ
プログラム名	CL(n)	プログラム名

ライブラリー・レコード: X'0060'

次の表に、SYSADATA ライブラリー・レコードの内容を示します。

表 109. SYSADATA ライブラリー・レコード

フィールド	サイズ	説明
メンバーの数 ¹	HL2	このレコードに記述される COPY/INCLUDE コード・メンバーの数のカウント
ライブラリー名長	HL2	ライブラリー名の長さ
ライブラリー・ボリューム長	HL2	ライブラリー・ボリューム ID の長さ
連結番号	XL2	ライブラリーの連結番号
ライブラリー DD 名長	HL2	ライブラリー DD 名の長さ
予約済み	CL4	将来の利用のために予約済み
ライブラリー名	CL(n)	COPY/INCLUDE メンバーが取り出された元のライブラリーの名前
ライブラリー・ボリューム	CL(n)	ライブラリーが常駐するボリュームのボリューム識別
ライブラリー DD 名	CL(n)	このライブラリーに使用される DD 名 (または同義)
...COPY/BASIS メンバー・ファイル ID ²	HL2	... の後の名前のライブラリー・ファイル ID
...COPY/BASIS 名長	HL2	... の後の名前の長さ
...COPY/BASIS 名	CL(n)	使用されてきた COPY/BASIS メンバーの名前

表 109. SYSADATA ライブラリー・レコード (続き)

フィールド	サイズ	説明
1. ライブラリーから COPY メンバーが 10 メンバー取り出された場合には、「メンバーの数」フィールドに 10 が入り、「COPY/BASIS メンバー・ファイル ID」フィールド、「COPY/BASIS 名長」フィールド、および「COPY/BASIS 名」フィールドのオカレンスが 10 回出現します。		
2. COPY/BASIS メンバーが別のライブラリーから取り出された場合には、それぞれの一意のライブラリーごとに、ライブラリー・レコードは SYSADATA ファイルに書き込まれます。		

統計レコード: X'0090'

次の表に、統計レコードの内容を示します。

表 110. SYSADATA 統計レコード

フィールド	サイズ	説明
ソース・レコード	FL4	処理されたソース・レコードの数
DATA DIVISION ステートメント	FL4	処理された DATA DIVISION ステートメントの数
PROCEDURE DIVISION ステートメント	FL4	処理された PROCEDURE DIVISION ステートメントの数
コンパイル番号	HL2	バッチ・コンパイル番号
エラーの重大度	XL1	最高のエラー・メッセージ重大度
フラグ	XL1	1... .. ジョブ終了標識 .1... .. クラス定義標識 ..11 1111 将来の利用のために予約済み
EOJ 重大度	XL1	コンパイル・ジョブの最大戻りコード
プログラム名の長さ	XL1	プログラム名の長さ
プログラム名	CL(n)	プログラム名

EVENTS レコード: X'0120'

以前のレベルのコンパイラーとの互換性を持たせるために、イベント・レコードが ADATA ファイルに含まれます。

イベント・レコードには次のタイプがあります。

- タイム・スタンプ
- プロセッサ
- ファイル終了
- プログラム

- ファイル ID
- エラー

表 111. SYSADATA EVENTS TIMESTAMP レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ TIMESTAMP のレコード	CL12	C'TIMESTAMP'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
日付	XL8	YYYYMMDD
時間	XL2	HH
分	XL2	MI
秒	XL2	SS

表 112. SYSADATA EVENTS PROCESSOR レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ PROCESSOR のレコード	CL9	C'PROCESSOR'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
出力ファイル ID	XL1	
ブランク・セパレータ ー	CL1	
行クラス標識	XL1	

表 113. SYSADATA EVENTS FILE END レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)

表 113. SYSADATA EVENTS FILE END レコードのレイアウト (続き)

フィールド	サイズ	説明
EVENTS レコード・ タイプ FILE END の レコード	CL7	C'FILEEND'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
入力ファイル ID	XL1	
ブランク・セパレータ ー	CL1	
拡張標識	XL1	

表 114. SYSADATA EVENTS PROGRAM レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ PROGRAM の レコード	CL7	C'PROGRAM'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
出力ファイル ID	XL1	
ブランク・セパレータ ー	CL1	
プログラム入力レコー ド番号	XL1	

表 115. SYSADATA EVENTS FILE ID レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ FILE ID のレ コード	CL7	C'FILEID'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	

表 115. SYSADATA EVENTS FILE ID レコードのレイアウト (続き)

フィールド	サイズ	説明
ブランク・セパレータ	CL1	
入力ソース・ファイル ID	XL1	ソース・ファイルのファイル ID
ブランク・セパレータ	CL1	
参照標識	XL1	
ブランク・セパレータ	CL1	
ソース・ファイル名の長さ	H2	
ブランク・セパレータ	CL1	
ソース・ファイル名	CL(n)	

表 116. SYSADATA EVENTS ERROR レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・タイプ ERROR のレコード	CL5	C'ERROR'
ブランク・セパレータ	CL1	
改訂レベル	XL1	
ブランク・セパレータ	CL1	
入力ソース・ファイル ID	XL1	ソース・ファイルのファイル ID
ブランク・セパレータ	CL1	
Annot クラス	XL1	Annot クラス・メッセージの配置
ブランク・セパレータ	CL1	
エラー入力レコード番号	XL10	
ブランク・セパレータ	CL1	
エラー開始行番号	XL10	
ブランク・セパレータ	CL1	

表 116. SYSADATA EVENTS ERROR レコードのレイアウト (続き)

フィールド	サイズ	説明
エラー・トークン開始 番号	XL1	エラー・トークン開始の桁番号
ブランク・セパレータ ー	CL1	
エラー終了行番号	XL10	
ブランク・セパレータ ー	CL1	
エラー・トークン終了 番号	XL1	エラー・トークン終了の桁番号
ブランク・セパレータ ー	CL1	
エラー・メッセージの ID 番号	XL9	
ブランク・セパレータ ー	CL1	
エラー・メッセージの 重大度コード	XL1	
ブランク・セパレータ ー	CL1	
エラー・メッセージの 重大度レベル番号	XL2	
ブランク・セパレータ ー	CL1	
エラー・メッセージ長	HL3	
ブランク・セパレータ ー	CL1	
エラー・メッセージ・ テキスト	CL(n)	

付録 J. ランタイム・メッセージ

COBOL for AIX のメッセージには、メッセージ接頭語、メッセージ番号、重大度コード、および説明テキストが含まれています。

メッセージ接頭語は、常に IWZ です。重大度コードは、I (通知)、W (警告)、S (重大)、C (クリティカル) のいずれかになります。メッセージ・テキストは、当該条件に関する簡単な説明です。

IWZ2519S The seconds value in a CEEISEC call was not recognized.

上のメッセージ例を以下に説明します。

- メッセージ接頭語は IWZ です。
- メッセージ番号は 2519 です。
- 重大度コードは S です。
- メッセージ・テキストは「The seconds value in a CEEISEC call was not recognized.」です。

日付および時刻 の呼び出し可能サービスからのメッセージには、シンボリック・フィードバック・コードも含まれています。このコードは、12 バイトの条件トークンのうち、最初の 8 バイトを表しています。シンボリック・フィードバック・コードは、条件のニックネームと考えることができます。呼び出し可能サービスのメッセージには、4 桁のメッセージ番号が含まれています。

メッセージ・カタログのメッセージ端末にてリスト全体を参照するには次のコマンドを実行してください。

```
dspcat cobol2run.cat
```

メッセージ番号が *nnnn* であるメッセージの内容のみを参照するには、次のコマンドを実行してください。

```
dspcat cobol2run.cat nnnn
```

コマンド行からアプリケーションを実行する場合に、ランタイム・メッセージを取り込むには、`stdout` および `stderr` をファイルに転送します。以下に例を示します。

```
program-name program-arguments >combined-output-file 2>&1
```

次の例は、出力内容を別のファイルに書き込む方法を示しています。

```
program-name program-arguments >output-file 2>error-file
```

表 117. ランタイム・メッセージ

メッセージ番号	メッセージ・テキスト
790 ページの『IWZ006S』	行 <i>line-number</i> で、verb 番号 <i>verb-number</i> によるテーブル <i>table-name</i> への参照が、テーブル領域外の領域をアドレス指定しました。

表 117. ランタイム・メッセージ (続き)

790 ページの『IWZ007S』	行 <i>line-number</i> で、verb 番号 <i>verb-number</i> による可変長グループ <i>group-name</i> への参照が、グループの最大定義長を超える領域をアドレス指定しました。
790 ページの『IWZ012I』	ソートまたはマージの実行中に、無効な実行単位終了が発生しました。
791 ページの『IWZ013S』	ソートまたはマージが要求されましたが、別のスレッドでソートまたはマージを実行中です。
791 ページの『IWZ026W』	SORT-RETURN 特殊レジスタは参照されませんでしたが、現在の内容は、行番号 <i>line-number</i> でプログラム <i>program-name</i> のソートまたはマージ操作が失敗したことを示しています。ソートまたはマージの戻りコードは <i>return code</i> です。
791 ページの『IWZ029S』	プログラム <i>program-name</i> の行 <i>line-number</i> にある関数 <i>function-name</i> の Argument-1 が 0 未満でした。
792 ページの『IWZ030S』	プログラム <i>program-name</i> の行 <i>line-number</i> にある関数 <i>function-name</i> の Argument-2 が正の整数ではありません。
792 ページの『IWZ036W』	プログラム <i>program-name</i> の行番号 <i>line-number</i> で、高位桁位置の切り捨てが発生しました。
792 ページの『IWZ037I』	プログラム <i>program-name</i> の制御フローが、プログラムの最終行を超えました。制御権は、プログラム <i>program-name</i> の呼び出し元に戻りました。
792 ページの『IWZ038S』	行 <i>line-number</i> にある <i>reference-modification-value</i> の参照変更長の値が 1 ではないことが、データ項目 <i>data-item</i> への参照内で検出されました。
793 ページの『IWZ039S』	無効なオーバパンチ符号が検出されました。
793 ページの『IWZ040S』	無効な分離符号が検出されました。
794 ページの『IWZ045S』	プログラム <i>program-name</i> の行番号 <i>line number</i> でメソッド <i>method-name</i> を呼び出すことができません。
794 ページの『IWZ047S』	クラス <i>class-name</i> の行番号 <i>line number</i> で、メソッド <i>method-name</i> を呼び出すことができません。
794 ページの『IWZ048W』	指数式で、負の基数が小数で累乗されました。基数の絶対値が使用されました。
794 ページの『IWZ049W』	指数式で、0 の基数が 0 で累乗されました。結果は 1 に設定されました。
795 ページの『IWZ050S』	指数式で、0 の基数が負数で累乗されました。
795 ページの『IWZ051W』	オペランドまたは受信側で指定された小数点位が多すぎるため、プログラム <i>program-name</i> の固定小数点指数演算に有効桁が残っていません。
795 ページの『IWZ053S』	浮動小数点への変換時にオーバーフローが発生しました。
796 ページの『IWZ054S』	浮動小数点例外が発生しました。
796 ページの『IWZ055W』	浮動小数点への変換時にアンダーフローが発生しました。結果は 0 に設定されました。

表 117. ランタイム・メッセージ (続き)

796 ページの『IWZ058S』	指数オーバーフローが発生しました。
796 ページの『IWZ059W』	9 桁を超える指数が切り捨てられました。
797 ページの『IWZ060W』	高位桁位置の切り捨てが発生しました。
797 ページの『IWZ061S』	0 による除算が発生しました。
797 ページの『IWZ063S』	<i>program-name</i> の行番号 <i>line-number</i> にある数値編集送信フィールドで、無効な符号が検出されました。
797 ページの『IWZ064S』	コンパイル単位 <i>compilation-unit</i> でのアクティブ・プログラム <i>program-name</i> に対する再帰呼び出しが試行されました。
798 ページの『IWZ065I』	コンパイル単位 <i>compilation-unit</i> で、アクティブ・プログラム <i>program-name</i> のキャンセルが試行されました。
798 ページの『IWZ066S』	プログラム <i>program-name</i> の外部データ・レコード <i>data-record</i> の長さが、既存のレコード長と一致しませんでした。
798 ページの『IWZ071S』	行 <i>line-number</i> で、verb 番号 <i>verb-number</i> によるテーブル <i>table-name</i> への ALL 添え字付きテーブル参照に、OCCURS DEPENDING ON 次元に対して指定された ALL 添え字があり、オブジェクトの値は 0 以下でした。
799 ページの『IWZ072S』	行 <i>line-number</i> で、 <i>reference-modification-value</i> の参照変更開始位置の値が、データ項目 <i>data-item</i> の領域外の領域を参照しました。
799 ページの『IWZ073S』	行 <i>line-number</i> にある <i>reference-modification-value</i> の参照変更長の値が正数ではないことが、データ項目 <i>data-item</i> の参照内で検出されました。
799 ページの『IWZ074S』	行 <i>line-number</i> にある <i>reference-modification-value</i> の参照変更開始位置の値と <i>length</i> の長さ値により、データ項目 <i>data-item</i> の右端文字を超える参照が行われました。
799 ページの『IWZ075S』	プログラム <i>program-name</i> の EXTERNAL ファイル <i>file-name</i> で矛盾が検出されました。attribute-1 attribute-2 attribute-3 attribute-4 attribute-5 attribute-6 attribute-7 の各ファイル属性が、設定済み外部ファイルのファイル属性と一致しませんでした。
800 ページの『IWZ076W』	INSPECT REPLACING CHARACTERS BY データ名の文字数が 1 ではありません。先頭文字が使用されました。
800 ページの『IWZ077W』	INSPECT データ項目の長さが等しくありません。短い方の長さが使用されました。
800 ページの『IWZ078S』	行 <i>line-number</i> で、verb 番号 <i>verb-number</i> によるテーブル <i>table-name</i> への ALL 添え字付き参照が、テーブルの上限を超えています。

表 117. ランタイム・メッセージ (続き)

801 ページの『IWZ096C』	<p>プログラム <i>program-name</i> の動的呼び出しが失敗しました。メッセージには、次のような種類があります。</p> <ul style="list-style-type: none"> モジュール <i>module-name</i> のロードが失敗しました。エラー・コードは <i>error-code</i> です。 モジュール <i>module-name</i> のロードが失敗しました。戻りコードは <i>return-code</i> です。 プログラム <i>program-name</i> の動的呼び出しが失敗しました。リソースが不十分です。 プログラム <i>program-name</i> の動的呼び出しが失敗しました。環境内で COBPATH が検出されませんでした。 プログラム <i>program-name</i> の動的呼び出しが失敗しました。入力項目 <i>entry-name</i> が検出されませんでした。 動的呼び出しが失敗しました。ターゲット・プログラムの名前に有効な文字が含まれていません。 プログラム <i>program-name</i> の動的呼び出しが失敗しました。ロード・モジュール <i>load-module</i> が、COBPATH 環境変数で識別されたディレクトリ内にありませんでした。
801 ページの『IWZ097S』	関数 <i>function-name</i> の Argument-1 に桁がありません。
802 ページの『IWZ100S』	関数 <i>function-name</i> の Argument-1 の値が -1 以下です。
802 ページの『IWZ103S』	関数 <i>function-name</i> の Argument-1 の値が 0 未満か、または 99 を超えています。
802 ページの『IWZ104S』	関数 <i>function-name</i> の Argument-1 の値が 0 未満か、または 99999 を超えています。
802 ページの『IWZ105S』	関数 <i>function-name</i> の Argument-1 の値が 0 未満か、または 999999 を超えています。
802 ページの『IWZ151S』	関数 <i>function-name</i> の Argument-1 が 18 桁を超えています。
803 ページの『IWZ152S』	関数 <i>function-name</i> の argument-1 にある列 <i>column-number</i> で、無効な文字 <i>character</i> が検出されました。
803 ページの『IWZ155S』	関数 <i>function-name</i> の argument-2 にある列 <i>column-number</i> で、無効な文字 <i>character</i> が検出されました。
803 ページの『IWZ156S』	関数 <i>function-name</i> の Argument-1 の値が 0 未満か、または 28 を超えています。
803 ページの『IWZ157S』	関数 <i>function-name</i> の Argument-1 の長さが 1 ではありません。
803 ページの『IWZ158S』	関数 <i>function-name</i> の Argument-1 の値が 0 未満か、または 29 を超えています。
804 ページの『IWZ159S』	関数 <i>function-name</i> の Argument-1 の値が 1 未満か、または 3067671 を超えています。
804 ページの『IWZ160S』	関数 <i>function-name</i> の Argument-1 の値が 16010101 未満か、または 99991231 を超えています。
804 ページの『IWZ161S』	関数 <i>function-name</i> の Argument-1 の値が 1601001 未満か、または 9999365 を超えています。

表 117. ランタイム・メッセージ (続き)

804 ページの『IWZ162S』	関数 <i>function-name</i> の Argument-1 の値が 1 未満か、またはプログラムの照合シーケンスの桁数を超過しています。
805 ページの『IWZ163S』	関数 <i>function-name</i> の Argument-1 の値が 0 未満です。
805 ページの『IWZ165S』	行 <i>line-number</i> で、 <i>start-position-value</i> の参照変更開始位置の値が、 <i>function-result</i> の関数結果領域外の領域を参照しました。
805 ページの『IWZ166S』	行 <i>line-number</i> にある <i>length</i> の参照変更長の値が正数ではないことが、 <i>function-result</i> の関数結果への参照内で検出されました。
805 ページの『IWZ167S』	行 <i>line-number</i> にある参照変更の開始位置の値 <i>start-position</i> と長さの値 <i>length</i> が原因で、関数結果 <i>function-result</i> の右端の文字を超えて参照が行われました。
806 ページの『IWZ168W』	SYSPUNCH/SYSPCH は、システムの論理出力装置をデフォルトに取ります。対応する環境変数が設定されていません。
806 ページの『IWZ170S』	DISPLAY オペランドのデータ型が正しくありません。
806 ページの『IWZ171I』	<i>string-name</i> は有効なランタイム・オプションではありません。
806 ページの『IWZ172I』	ストリング <i>string-name</i> は、ランタイム・オプション <i>option-name</i> の有効なサブオプションではありません。
807 ページの『IWZ173I』	ランタイム・オプション <i>option-name</i> のサブオプション・ストリング <i>string-name</i> の文字長は <i>number</i> でなければなりません。デフォルトが使用されます。
807 ページの『IWZ174I』	ランタイム・オプション <i>option-name</i> のサブオプション・ストリング <i>string-name</i> に無効な文字が 1 つ以上含まれています。デフォルトが使用されます。
807 ページの『IWZ175S』	このシステムでは、ルーチン <i>routine-name</i> がサポートされていません。
807 ページの『IWZ176S』	関数 <i>function-name</i> の Argument-1 が <i>decimal-value</i> より大きい値でした。
808 ページの『IWZ177S』	関数 <i>function-name</i> の Argument-2 が <i>decimal-value</i> と等しい値でした。
808 ページの『IWZ178S』	関数 <i>function-name</i> の Argument-1 の値が <i>decimal-value</i> 以下です。
808 ページの『IWZ179S』	関数 <i>function-name</i> の Argument-1 が <i>decimal-value</i> より小さい値でした。
808 ページの『IWZ180S』	関数 <i>function-name</i> の Argument-1 の値が整数ではありません。
808 ページの『IWZ181I』	ランタイム・オプション <i>option-name</i> の数値ストリング <i>string</i> で、無効な文字が検出されました。デフォルトが使用されます。
809 ページの『IWZ182I』	ランタイム・オプション <i>option-name</i> の数値 <i>number</i> が、 <i>min-range</i> から <i>max-range</i> の範囲を超えています。デフォルトが使用されません。
809 ページの『IWZ183S』	_IWZCOBOLInit 内の関数名が戻りを行いました。

表 117. ランタイム・メッセージ (続き)

809 ページの『IWZ200S』	ファイル <i>file-name</i> に対する <i>I/O operation</i> の実行中に、エラーが検出されました。ファイル状況は <i>file-status</i> です。
809 ページの『IWZ200S』	入出力エラー <i>error-code</i> により、STOP または ACCEPT が失敗しました。実行単位は終了します。
810 ページの『IWZ201C』	
810 ページの『IWZ203W』	有効なコード・ページが DBCS コード・ページではありません。
811 ページの『IWZ204W』	ASCII DBCS から EBCDIC DBCS への変換中にエラーが発生しました。
811 ページの『IWZ221S』	コード・ページ <i>codepage value</i> の ICU コンバーターをオープンできません。
811 ページの『IWZ222S』	エラー・コード <i>error code value</i> により、ICU を使用したデータ変換が失敗しました。
812 ページの『IWZ223S』	エラー・コード <i>error code value</i> により、ICU コンバーターのクローズが失敗しました。
812 ページの『IWZ224S』	ロケール値 <i>locale value</i> の ICU コレクターをオープンできません。エラー・コードは <i>error code value</i> です。
812 ページの『IWZ225S』	エラー・コード <i>error code value</i> により、ICU を使用した Unicode ケース・マッピング関数が失敗しました。
812 ページの『IWZ230W』	現行のコード・セット <i>ASCII codeset-id</i> から EBCDIC コード・セット <i>EBCDIC codeset-id</i> への変換テーブルを使用できません。デフォルトの ASCII/EBCDIC 変換テーブルが使用されます。
813 ページの『IWZ230W』	指定された EBCDIC コード・ページ <i>EBCDIC codepage</i> がロケール <i>locale</i> と一貫していませんが、要求どおりに使用されます。
813 ページの『IWZ230W』	指定された EBCDIC コード・ページ <i>EBCDIC codepage</i> はサポートされていません。デフォルトの EBCDIC コード・ページ <i>EBCDIC codepage</i> が使用されます。
813 ページの『IWZ230S』	EBCDIC 変換テーブルをオープンできません。
813 ページの『IWZ230S』	EBCDIC 変換テーブルを構築できません。
814 ページの『IWZ230S』	メインプログラムは <i>-host</i> フラグと <i>CHAR(NATIVE)</i> オプションの両方でコンパイルされましたが、これらには互換性がありません。
814 ページの『IWZ231S』	現行のロケール設定の照会が失敗しました。

表 117. ランタイム・メッセージ (続き)

<p>815 ページの『IWZ232W』</p>	<p>メッセージには、次のような種類があります。</p> <ul style="list-style-type: none"> • データ項目 <i>data-name</i> を EBCDIC に変換するときに、プログラム <i>program-name</i> の行番号 <i>decimal-value</i> でエラーが発生しました。 • データ項目 <i>data-name</i> を ASCII に変換するときに、プログラム <i>program-name</i> の行番号 <i>decimal-value</i> でエラーが発生しました。 • データ項目 <i>data-name</i> について EBCDIC に変換するときに、プログラム <i>program-name</i> の行番号 <i>decimal-value</i> でエラーが発生しました。 • データ項目 <i>data-name</i> について ASCII に変換するときに、プログラム <i>program-name</i> の行番号 <i>decimal-value</i> でエラーが発生しました。 • プログラム <i>program-name</i> の行番号 <i>decimal-value</i> で、ASCII から EBCDIC への変換中にエラーが発生しました。 • プログラム <i>program-name</i> の行番号 <i>decimal-value</i> で、EBCDIC から ASCII への変換中にエラーが発生しました。
<p>815 ページの『IWZ233S』</p>	<p>COBOL プログラム <i>program-name</i> は NOTHREAD コンパイラー・オプションを使用してコンパイルされ、マルチスレッド化されたライブラリーにリンクされているため、実行できません。</p>
<p>815 ページの『IWZ240S』</p>	<p>プログラム <i>program-name</i> の基本年が 1900 から 1999 の有効範囲内にありません。スライディング・ウィンドウ値 <i>window-value</i> は、<i>base-year</i> の基本年になります。</p>
<p>816 ページの『IWZ241S』</p>	<p>現行年が、プログラム <i>program-name</i> に使用されている <i>year-start</i> から <i>year-end</i> の 100 年間隔内にありません。</p>
<p>816 ページの『IWZ242S』</p>	<p>XML PARSE ステートメントを開始しようとしたのですが、この操作は無効です。</p>
<p>817 ページの『IWZ243S』</p>	<p>XML PARSE ステートメントを終了しようとしたのですが、この操作は無効です。</p>
<p>817 ページの『IWZ250S』</p>	<p>内部エラー: JNI_GetCreatedJavaVMs への呼び出しにより、エラー、戻りコード <i>nm</i> が戻されました。</p>
<p>817 ページの『IWZ251S』</p>	<p>内部エラー: <i>n</i> 個のアクティブ Java VM が検出されましたが、期待される数は 1 つだけです。</p>
<p>817 ページの『IWZ253S』</p>	<p><i>nm</i> 個を超える JVM 初期化オプションが指定されました。</p>
<p>818 ページの『IWZ254S』</p>	<p>内部エラー: JNI_CreateJavaVM への呼び出しによりエラーが戻されました。</p>
<p>818 ページの『IWZ255S』</p>	<p>内部エラー: 現行のスレッドが JVM に接続されていないため、GetEnv への呼び出しにより、コード <i>nm</i> が戻されました。</p>
<p>818 ページの『IWZ256S』</p>	<p>内部エラー: JVM バージョンがサポートされていないため、GetEnv への呼び出しにより、コード <i>nm</i> が戻されました。</p>
<p>818 ページの『IWZ257S』</p>	<p>内部エラー: GetEnv への呼び出しにより、認識されていない戻りコード <i>nm</i> が戻されました。</p>
<p>819 ページの『IWZ258S』</p>	<p>内部エラー: GetByteArrayElements が、インスタンス・データを指すポインターを取得できませんでした。</p>

表 117. ランタイム・メッセージ (続き)

819 ページの『IWZ259S』	インスタンス・データを指す直接ポインタを取得できません。インストール済みの JVM では、pinned (滞留) されるバイト配列はサポートされていません。
819 ページの『IWZ260S』	Java クラス <i>name</i> を検出できませんでした。
819 ページの『IWZ813S』	使用可能なストレージが不十分なため、ストレージ取得要求を満たすことができません。
820 ページの『IWZ901S』	メッセージには、次のような種類があります。 <ul style="list-style-type: none"> • 重大エラーまたはクリティカル・エラーが発生したため、プログラムが終了します。 • プログラムの終了: ERRCOUNT を超えるエラー数が発生しました。
820 ページの『IWZ902S』	システムが 10 進数除算例外を検出しました。
820 ページの『IWZ903S』	システムがデータ例外を検出しました。
820 ページの『IWZ907S』	メッセージには、次のような種類があります。 <ul style="list-style-type: none"> • ストレージが不十分です。 • ストレージが不十分です。storage 用に <i>number-bytes</i> バイトのスペースを取得できません。
821 ページの『IWZ993W』	ストレージが不十分です。メッセージ <i>message-number</i> 用のスペースを検出できません。
821 ページの『IWZ994W』	メッセージ・カタログ内でメッセージ <i>message-number</i> を検出できません。
821 ページの『IWZ995C』	メッセージには、次のような種類があります。 <ul style="list-style-type: none"> • オフセット <i>0xoffset-value</i> でルーチン <i>routine-name</i> の実行中に、<i>System exception</i> シグナルを受信しました。 • <i>0xoffset-value</i> の場所でコードの実行中に、<i>System exception</i> シグナルを受信しました。 • <i>System exception</i> シグナルを受信しました。場所を判別できませんでした。
822 ページの『IWZ2502S』	システムから UTC/GMT を使用できませんでした。
822 ページの『IWZ2503S』	UTC/GMT から現地時間までのオフセットをシステムから使用できませんでした。
822 ページの『IWZ2505S』	CEEDATM または CEESECI への呼び出し内の <i>input_seconds</i> 値が、対応範囲内にありませんでした。
823 ページの『IWZ2506S』	CEEDATM に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、または <CCCCCCCC>) が使用されていましたが、入力された秒数値が対応範囲内にありませんでした。元号を判別できませんでした。
823 ページの『IWZ2507S』	CEEDAYS または CEESECS に渡されたデータが不十分です。リアン日付の値は計算されませんでした。
823 ページの『IWZ2508S』	CEEDAYS または CEESECS に渡された日付値が無効です。

表 117. ランタイム・メッセージ (続き)

824 ページの『IWZ2509S』	CEEDAYS または CEESECS に渡された元号が認識されませんでした。
824 ページの『IWZ2510S』	CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。
824 ページの『IWZ2511S』	CEEISEC の呼び出しで渡された日のパラメーターが、指定された年および月に対して無効です。
825 ページの『IWZ2512S』	CEEDATE または CEEDYWK への呼び出しで渡されたりリアン日付値が、対応範囲内にありませんでした。
825 ページの『IWZ2513S』	CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。
825 ページの『IWZ2514S』	CEEISEC の呼び出しで渡された年の値が、対応範囲内にありませんでした。
825 ページの『IWZ2515S』	CEEISEC 呼び出し内のミリ秒の値が認識されませんでした。
826 ページの『IWZ2516S』	CEEISEC 呼び出し内の分の値が認識されませんでした。
826 ページの『IWZ2517S』	CEEISEC 呼び出し内の月の値が認識されませんでした。
827 ページの『IWZ2518S』	日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。
827 ページの『IWZ2519S』	CEEISEC 呼び出し内の秒の値が認識されませんでした。
827 ページの『IWZ2520S』	CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。
828 ページの『IWZ2521S』	CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。
828 ページの『IWZ2522S』	CEEDATE に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、または <CCCCCCCC>) が使用されていましたが、リアン日付値が対応範囲内にありませんでした。元号を判別できませんでした。
828 ページの『IWZ2525S』	CEESECS が数値フィールド内に非数値データを検出したか、あるいはタイム・スタンプ・ストリングとピクチャー・ストリングが一致しませんでした。
829 ページの『IWZ2526S』	CEEDATE によって戻された日付ストリングが切り捨てられました。
829 ページの『IWZ2527S』	CEEDATM によって戻されたタイム・スタンプ・ストリングが切り捨てられました。
829 ページの『IWZ2531S』	システムから現地時間を使用できませんでした。
829 ページの『IWZ2533S』	CEESCEN に渡された値が 0 から 100 の範囲内にありませんでした。
830 ページの『IWZ2534W』	CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。出力はブランクに設定されました。

IWZ006S

行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への参照が、テーブル領域外の領域をアドレス指定しました。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、固定長テーブルに添え字があるためにテーブルの定義サイズを超えているか、または可変長テーブルの場合はテーブルの最大サイズを超えていることを示します。

添え字の複合に対して範囲検査が実行された結果、テーブルの領域外のアドレスが戻されました。可変長テーブルで、すべての OCCURS DEPENDING ON オブジェクトが最大値になっている場合、このアドレスは定義されたテーブルの領域外にあります。ODO オブジェクトの現行値は考慮されません。個々の添え字に対しては、検査が行われていません。

プログラマー応答: 実行時に評価されるリテラル添え字や変数添え字の値が、失敗したステートメント内の添え字付きデータに対する添え字付き次元を超えないようにしてください。

システム処置: アプリケーションは終了します。

IWZ007S

行 *line-number* で、verb 番号 *verb-number* による可変長グループ *group-name* への参照が、グループの最大定義長を超える領域をアドレス指定しました。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、OCCURS DEPENDING ON によって生成された可変長グループの長さが 0 未満か、または OCCURS DEPENDING ON 文節で定義された限度を超えていることを示します。

範囲検査は、個々の OCCURS DEPENDING ON オブジェクトに対してではなく、グループの複合長に対して実行されています。

プログラマー応答: 実行時に評価される OCCURS DEPENDING ON オブジェクトが、参照先グループ項目内のテーブルに関する次元の最大オカレンス数を超えないようにしてください。

システム処置: アプリケーションは終了します。

IWZ012I

ソートまたはマージの実行中に、無効な実行単位終了が発生しました。

説明: COBOL プログラムによって開始されたソートまたはマージの進行中に、次のいずれかが試行されました。

1. STOP RUN が出された。
2. ソートまたはマージを開始した COBOL プログラムの入力プロシージャまたは出力プロシージャで、GOBACK または EXIT PROGRAM が出された。

GOBACK および EXIT PROGRAM ステートメントは、入力プロシージャまたは出力プロシージャによって呼び出されたプログラム内で使用できることに注意してください。

プログラマー応答: 上記のメソッドを使用せずにソートまたはマージを終了するようにアプリケーションを変更してください。

システム処置: アプリケーションは終了します。

IWZ013S

ソートまたはマージが要求されましたが、別のスレッドでソートまたはマージを実行中です。

説明: 2 つ以上のスレッドでソートまたはマージを同時に実行することはできません。

プログラマー応答: ソートまたはマージは、必ず同じスレッドで実行してください。あるいは、ソートまたはマージの各呼び出しの前に、ソートまたはマージを別のスレッドで実行しているかどうかを判断するコードを組み込むことができます。ソートまたはマージが別のスレッドで実行中の場合は、そのスレッドが完了するのを待ちます。そうでない場合は、ソートまたはマージを実行することを示すフラグを設定してから、ソートまたはマージを呼び出します。

システム処置: スレッドは終了します。

IWZ026W

SORT-RETURN 特殊レジスターは参照されませんでしたが、現在の内容は、行番号 *line-number* でプログラム *program-name* のソートまたはマージ操作が失敗したことを示しています。ソートまたはマージの戻りコードは *return code* です。

説明: COBOL ソースには、**SORT-RETURN** レジスターへの参照が含まれていません。コンパイラーは、ソートまたはマージの各 verb の後でテストを生成します。ソートまたはマージによって、ゼロ以外の戻りコードがプログラムに戻されています。

プログラマー応答: ソートまたはマージが失敗した理由を判断し、問題を修正してください。可能な戻りコードのリストについては 165 ページの『ソートおよびマージ・エラー番号』を参照してください。

システム処置: システムのアクションはとられません。

IWZ029S

プログラム *program-name* の行 *line-number* にある関数 *function-name* の **Argument-1** が 0 未満です。

説明: argument-1 に対して無効な値が使用されました。

プログラマー応答: argument-1 の値が 0 以上であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ030S

プログラム *program-name* の行 *line-number* にある関数 *function-name* の Argument-2 が正の整数ではありません。

説明: argument-1 に対して無効な値が使用されました。

プログラマー応答: argument-2 の値が正の整数であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ036W

プログラム *program-name* の行番号 *line-number* で、高位桁位置の切り捨てが発生しました。

説明: 生成されたコードによって、中間結果 (算術演算中に使用される一時記憶域) が 30 桁に切り詰められましたが、切り捨てられた桁の中に、値が 0 でないものがあります。

プログラマー応答: 中間結果については、このセクションの末尾にある関連概念を参照してください。

システム処置: システムのアクションはとられません。

IWZ037I

プログラム *program-name* の制御フローが、プログラムの最終行を超えました。制御権は、プログラム *program-name* の呼び出し元に戻りました。

説明: プログラムに終止符 (STOP、GOBACK、または EXIT) がないため、制御権が最後の命令をフォールスルーしました。

プログラマー応答: プログラムのロジックを検査してください。次のいずれかの論理エラーがあると、このエラーが発生する場合があります。

- プログラム内の最後の段落は PERFORM ステートメントの結果として制御権を受け取るものと想定されていたが、論理エラーがあるために、GO TO ステートメントによって分岐された。
- プログラム内の最後の段落が「フォールスルー」パスの結果として実行されたが、段落の末尾にプログラムを終了するためのステートメントがない。

システム処置: アプリケーションは終了します。

IWZ038S

行 *line-number* にある *reference-modification-value* の参照変更長の値が 1 ではないことが、データ項目 *data-item* への参照内で検出されました。

説明: 参照変更で指定された長さ値が 1 ではありません。この長さ値は 1 でなければなりません。

プログラマー応答: プログラム内の指定された行番号を検査して、参照変更された長さ値がすべて 1 になっていることを確認してください (そうでない場合は、値を修正してください)。

システム処置: アプリケーションは終了します。

IWZ039S

無効なオーバーパンチ符号が検出されました。

説明: 符号位置の値が無効です。

X'sd' (s は符号表現であり、d は数字を表す) の場合、外部 10 進数 (SIGN IS SEPARATE 文節を指定しない USAGE DISPLAY) についての有効な符号表現は、以下のとおりです。

正: 0、1、2、3、8、9、A、および B
負: 4、5、6、7、C、D、E、および F

内部生成される符号は、正および符号なしの場合は 3、負の場合は 7 になります。

X'ds' (d は数字、s は符号表現を表す) の場合、内部 10 進数 (USAGE PACKED-DECIMAL) COBOL データの有効な符号表現は次のようになります。

正: A、C、E、および F
負: B および D

内部生成される符号は、正および符号なしの場合は C、負の場合は D になります。

プログラマー応答: REDEFINES 文節に符号位置が含まれている場合、符号位置を含むグループを移動した場合、または位置が初期設定されていない場合には、このエラーが発生する場合があります。このようなことがないかどうかを検査してください。

システム処置: アプリケーションは終了します。

IWZ040S

無効な分離符号が検出されました。

説明: 分離記号を使用して定義されたデータを操作しようとしていました。符号位置の値はプラス (+) またはマイナス (-) ではありませんでした。

プログラマー応答: REDEFINES 文節に符号位置が含まれている場合、符号位置を含むグループを移動した場合、または位置が初期設定されていない場合には、このエラーが発生する場合があります。このようなことがないかどうかを検査してください。

システム処置: アプリケーションは終了します。

IWZ045S

プログラム *program-name* の行番号 *line number* で、メソッド *method-name* を呼び出すことができません。

説明: 現行のオブジェクト参照のクラスに対し、特定のメソッドがサポートされていません。

プログラマー応答: プログラム内の指定された行番号を検査して、現行のオブジェクト参照のクラスが、呼び出されるメソッドをサポートすることを確認してください。

システム処置: アプリケーションは終了します。

IWZ047S

クラス *class-name* の行番号 *line number* で、メソッド *method-name* を呼び出すことができません。

説明: 現行のオブジェクト参照のクラスに対し、特定のメソッドがサポートされていません。

プログラマー応答: クラス内の指定された行番号を検査して、現行のオブジェクト参照のクラスが、呼び出されるメソッドをサポートすることを確認してください。

システム処置: アプリケーションは終了します。

IWZ048W

指数式で、負の基数が小数で累乗されました。基数の絶対値が使用されました。

説明: ライブラリー・ルーチン内で、負数が小数で累乗されました。

COBOL では、負の数を小数で累乗した値は定義されていません。当該ステートメントに SIZE ERROR 文節があれば、SIZE ERROR 命令が使用されますが、実際には SIZE ERROR 文節は存在しないため、基数の絶対値が指数に使用されました。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ049W

指数式で、0 の基数が 0 で累乗されました。結果は 1 に設定されました。

説明: ライブラリー・ルーチン内で、0 の値が 0 で累乗されました。

COBOL では、0 の値を 0 で累乗する演算は定義されていません。当該ステートメントに SIZE ERROR 文節があれば、SIZE ERROR 命令が使用されますが、実際には SIZE ERROR 文節は存在しないため、戻り値は 1 になりました。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ050S

指数式で、0 の基数が負数で累乗されました。

説明: ライブラリー・ルーチン内で、0 の値が負数で累乗されました。

0 の値を負数で累乗する演算は定義されていません。当該ステートメントに SIZE ERROR 文節があれば、SIZE ERROR 命令が使用されますが、実際には SIZE ERROR 文節は存在しません。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ051W

オペランドまたは受信側で指定された小数点位が多すぎるため、プログラム *program-name* の固定小数点指数演算に有効桁が残っていません。

説明: オペランドまたは受信側で指定されている小数点位が多すぎるため、固定小数点計算で生成された結果に有効数字が含まれていません。

プログラマー応答: 必要に応じて、オペランドまたは受け取り側の数値項目の PICTURE 文節を変更して、整数位を増やし、小数点位を減らしてください。

システム処置: システムのアクションはとられません。

IWZ053S

浮動小数点への変換時にオーバーフローが発生しました。

説明: プログラム内で生成された数値が大きすぎて、浮動小数点で表現できません。

プログラマー応答: プログラムを正しく修正して、オーバーフローを回避する必要があります。

システム処置: アプリケーションは終了します。

IWZ054S

浮動小数点例外が発生しました。

説明: 浮動小数点計算で生成された結果が正しくありません。浮動小数点計算は IEEE 浮動小数点数演算を使用して実行されますが、その際に NaN (非数値) と呼ばれる結果が出る場合があります。例えば、0 を 0 で除算すると、結果は NaN になります。

プログラマー応答: NaN が生成されないように、この演算に対する引数をテストするようにプログラムを修正してください。

システム処置: アプリケーションは終了します。

IWZ055W

浮動小数点への変換時にアンダーフローが発生しました。結果は 0 に設定されました。

説明: 浮動小数点への変換時に、負の指数がハードウェアの限度を超えました。浮動小数点値は 0 に設定されました。

プログラマー応答: 必須の処置は特にありませんが、必要に応じて、アンダーフローを回避するようにプログラムを修正してください。

システム処置: システムのアクションはとられません。

IWZ058S

指数オーバーフローが発生しました。

説明: 浮動小数点の指数オーバーフローが、ライブラリー・ルーチンで発生しました。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ059W

9 桁を超える指数が切り捨てられました。

説明: 固定小数点の累乗法では、指数は 9 桁以下でなければなりません。指数が 9 桁に切り詰められましたが、切り捨てられた桁の中に、値が 0 でないものがあります。

プログラマー応答: 必須の処置は特にありませんが、必要に応じて、失敗したステートメント内の指数を調整してください。

システム処置: システムのアクションはとられません。

IWZ060W

高位桁位置の切り捨てが発生しました。

説明: ライブラリー・ルーチン内のコードによって、中間結果 (算術演算中に使用される一時記憶域) が 30 桁に切り詰められましたが、切り捨てられた桁の中に、値が 0 でないものがあります。

プログラマー応答: 中間結果については、このセクションの末尾にある関連概念を参照してください。

システム処置: システムのアクションはとられません。

IWZ061S

0 による除算が発生しました。

説明: ライブラリー・ルーチン内で、0 による除算が行われました。0 による除算は定義されていません。当該ステートメントに `SIZE ERROR` 文節があれば、`SIZE ERROR` 命令が使用されますが、実際には `SIZE ERROR` 文節は存在しません。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ063S

program-name の行番号 *line-number* にある数値編集送信フィールドで、無効な符号が検出されました。

説明: `MOVE` ステートメントで、符号付き数値編集フィールドを符号付き数値または数値編集受信フィールドに移動しようとしたのですが、送信フィールド内の符号位置に含まれている文字が、対応する `PICTURE` に有効な符号文字ではありませんでした。

プログラマー応答: 失敗したステートメント内のプログラム変数が正しく設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ064S

コンパイル単位 *compilation-unit* で、アクティブ・プログラム *program-name* への再帰呼び出しが試行されました。

説明: COBOL では、内部プログラムの再帰呼び出しを行うことはできません。プログラムは実行を開始しましたが、まだ終了していません。例えば、内部プログラム A と B が収容プログラムの兄弟で、A が B を呼び出し、B が A を呼び出すと、このメッセージが出されます。

プログラマー応答: プログラムを調べて、アクティブな内部プログラムへの呼び出しを除去してください。

システム処置: アプリケーションは終了します。

IWZ065I

コンパイル単位 *compilation-unit* で、アクティブ・プログラム *program-name* のキャンセルが試行されました。

説明: アクティブな内部プログラムをキャンセルしようとしてしました。例えば、内部プログラム A と B が収容プログラム内の兄弟で、A が B を呼び出し、B が A をキャンセルすると、このメッセージが出されます。

プログラマー応答: プログラムを調べて、アクティブな内部プログラムのキャンセルを除去してください。

システム処置: アプリケーションは終了します。

IWZ066S

プログラム *program-name* の外部データ・レコード *data-record* の長さが、既存のレコード長と一致していませんでした。

説明: プログラムの初期化で外部データ・レコードを処理しているときに、実行単位内の別のプログラムで前に外部データ・レコードが定義されており、現行のプログラムで指定されているレコードの長さが、前に定義された長さと異なることが判別されました。

プログラマー応答: 現在のファイルを調べて、外部データ・レコードが正しく指定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ071S

行 *line-number* で、verb 番号 *verb-number* によるテーブル *table-name* への ALL 添え字付きテーブル参照に、OCCURS DEPENDING ON 次元に対して指定された ALL 添え字があり、オブジェクトの値は 0 以下でした。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、ALL による添え字付きの次元のオカレンス数が 0 であることを示します。

OCCURS DEPENDING ON オブジェクトの現行値に対して検査が実行されます。

プログラマー応答: 当該ステートメントにあるすべての添え字付き項目について、ALL 添え字付き次元の ODO オブジェクトが正数であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ072S

行 *line-number* で、*reference-modification-value* の参照変更開始位置の値が、データ項目 *data-item* の領域外の領域を参照しました。

説明: 参照変更指定の開始位置の値が 1 未満か、または参照変更されていたデータ項目の現行長を超えています。開始位置の値は、参照変更されるデータ項目の文字数以下の、正の整数でなければなりません。

プログラマー応答: 参照変更指定の開始位置の値を検査してください。

システム処置: アプリケーションは終了します。

IWZ073S

行 *line-number* にある *reference-modification-value* の参照変更長の値が正数ではないことが、データ項目 *data-item* への参照内で検出されました。

説明: 参照変更で指定された長さ値が 0 以下です。この長さ値は正の整数でなければなりません。

プログラマー応答: プログラム内の指定された行番号を検査して、参照変更された長さ値がすべて正の整数になっていることを確認してください (そうでない場合は、値を修正してください)。

システム処置: アプリケーションは終了します。

IWZ074S

行 *line-number* にある *reference-modification-value* の参照変更開始位置の値と *length* の長さ値により、データ項目 *data-item* の右端文字を超える参照が行われました。

説明: 参照変更指定の開始位置と長さ値を組み合わせるとアドレス指定された領域が、参照変更されるデータ項目の終わりを超えています。開始位置と長さ値の合計から 1 を引いた値が、参照変更されるデータ項目の文字数以下でなければなりません。

プログラマー応答: プログラム内の指定された行番号を検査して、参照がデータ項目の右端文字を超えないように、参照変更される開始値と長さ値が設定されていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ075S

プログラム *program-name* の EXTERNAL ファイル *file-name* で矛盾が検出されました。
attribute-1 attribute-2 attribute-3 attribute-4 attribute-5 attribute-6 attribute-7 の各ファイル属性が、設定済み外部ファイルのファイル属性と一致しませんでした。

説明: 外部ファイルの 1 つ以上の属性が、その外部ファイルを定義した 2 つのプログラム間で一致していません。

プログラマー応答: 外部ファイルを訂正してください。同じ外部ファイルの定義間でファイル属性の要約について一致させる必要があります。

システム処置: アプリケーションは終了します。

IWZ076W

INSPECT REPLACING CHARACTERS BY データ名の文字数が 1 ではありません。先頭文字が使用されました。

説明: INSPECT ステートメントの REPLACING 句の中の CHARACTERS 句にあるデータ項目の長さは、1 文字として定義する必要があります。このデータ項目の参照変更指定により、結果として生じる長さ値が 1 になりませんでした。長さ値は 1 でなければなりません。

プログラマー応答: 必要に応じて、失敗した INSPECT ステートメントの参照変更指定を訂正して、参照変更の長さが 1 になるようにしてください。プログラマーによる処置は必須ではありません。

システム処置: システムのアクションはとられません。

IWZ077W

INSPECT データ項目の長さが等しくありません。短い方の長さが使用されました。

説明: INSPECT ステートメントの REPLACING または CONVERTING 句にある 2 つのデータ項目は、2 番目の項目が表意定数の場合を除き、同じ長さでなければなりません。このようなデータ項目の一方または両方に対して参照変更が行われたため、結果的に長さ値が同じでなくなりました。短い方の長さが両方の項目に適用され、実行が続けられます。

プログラマー応答: 必要に応じて、失敗した INSPECT ステートメント内で長さの等しくないオペランドを調整してください。プログラマーによる処置は必須ではありません。

システム処置: システムのアクションはとられません。

IWZ078S

行 *line-number* で、**verb** 番号 *verb-number* によるテーブル *table-name* への ALL 添え字付き参照が、テーブルの上限を超えています。

説明: SSRANGE オプションが有効なときに、このメッセージが出された場合は、多次元テーブルで ALL が 1 つ以上の添え字として指定されているために、参照がテーブルの上限を超えてしまうことを示します。

範囲検査は、添え字の複合と、ALL 添え字付き次元の最大オカレンスに対して実行されています。可変長テーブルで、すべての OCCURS DEPENDING ON オブジェ

クトが最大値になっている場合、このアドレスは定義されたテーブルの領域外にあります。ODO オブジェクトの現行値は考慮されません。個々の添え字に対しては、検査が行われていません。

プログラマー応答: 実行時に評価される OCCURS DEPENDING ON オブジェクトが、失敗したステートメント内で参照されるテーブル項目に関する次元の最大オカレンス数を超えないようにしてください。

システム処置: アプリケーションは終了します。

IWZ096C

プログラム *program-name* の動的呼び出しが失敗しました。メッセージには、次のような種類があります。

- モジュール *module-name* のロードが失敗しました。エラー・コードは *error-code* です。
- モジュール *module-name* のロードが失敗しました。戻りコードは *return-code* です。
- プログラム *program-name* の動的呼び出しが失敗しました。リソースが不十分です。
- プログラム *program-name* の動的呼び出しが失敗しました。環境内で COBPATH が検出されませんでした。
- プログラム *program-name* の動的呼び出しが失敗しました。入力項目 *entry-name* が検出されませんでした。
- 動的呼び出しが失敗しました。ターゲット・プログラムの名前に有効な文字が含まれていません。
- プログラム *program-name* の動的呼び出しが失敗しました。ロード・モジュール *load-module* が、COBPATH 環境変数で識別されたディレクトリー内にありませんでした。

説明: 上記の各種メッセージに示されたいずれかの理由で、動的呼び出しが失敗しました。上記の *error-code* の値は、load によって設定されたエラー番号です。

プログラマー応答: COBPATH が定義されていることを確認してください。モジュールの存在を確認してください。AIX には、ディレクトリーとファイルを表示するためのグラフィカル・インターフェースがあります。ls コマンドも使用することができます。ロードされるモジュールの名前と、呼び出される入力項目の名前が一致していることを確認してください。適切な cob2 オプション。

システム処置: アプリケーションは終了します。

IWZ097S

関数 *function-name* の Argument-1 に桁がありません。

説明: 指定された関数の Argument-1 には、少なくとも 1 桁が含まれていなければなりません。

プログラマー応答: 失敗したステートメントで Argument-1 の桁数を調整してください。

システム処置: アプリケーションは終了します。

IWZ100S

関数 *function* の Argument-1 の値が -1 以下です。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: argument-1 が -1 より大きいことを確認してください。

システム処置: アプリケーションは終了します。

IWZ103S

関数 *function-name* の Argument-1 の値が 0 未満か、または 99 を超えています。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ104S

関数 *function-name* の Argument-1 の値が 0 未満か、または 99999 を超えています。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ105S

関数 *function-name* の Argument-1 の値が 0 未満か、または 999999 を超えています。

説明: Argument-1 に使用された値が正しくありません。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ151S

関数 *function-name* の Argument-1 が 18 桁を超えています。

説明: 指定された関数の Argument-1 の合計桁数が 18 桁を超えています。

プログラマー応答: 失敗したステートメントで Argument-1 の桁数を調整してください。

システム処置: アプリケーションは終了します。

IWZ152S

関数 *function-name* の *argument-1* にある列 *column-number* で、無効な文字 *character* が検出されました。

説明: NUMVAL/NUMVAL-C 関数の *argument-1* で、小数点、コンマ、スペース、または符号 (+、-、CR、DB) 以外の非数字文字が検出されました。

プログラマー応答: 指定されたステートメントで NUMVAL または NUMVAL-C の *argument-1* を調整してください。

システム処置: アプリケーションは終了します。

IWZ155S

関数 *function-name* の *argument-2* にある列 *column-number* で、無効な文字 *character* が検出されました。

説明: NUMVAL-C 関数の *argument-2* で、無効な文字が検出されました。

プログラマー応答: 関数の引数が構文規則に従っていることを確認してください。

システム処置: アプリケーションは終了します。

IWZ156S

関数 *function-name* の *Argument-1* の値が 0 未満か、または 28 を超えています。

説明: 関数 FACTORIAL に対する入力引数が 28 を超えているか、または 0 未満です。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ157S

関数 *function-name* の *Argument-1* の長さが 1 ではありません。

説明: ORD 関数に対する入力引数の長さが 1 ではありません。

プログラマー応答: 関数の引数が 1 バイト長であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ158S

関数 *function-name* の *Argument-1* の値が 0 未満か、または 29 を超えています。

説明: 関数 FACTORIAL に対する入力引数が 29 を超えているか、または 0 未満です。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ159S

関数 *function-name* の Argument-1 の値が 1 未満か、または 3067671 を超えています。

説明: DATE-OF-INTEGGER または DAY-OF-INTEGGER 関数に対する入力引数が 1 未満か、または 3067671 を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ160S

関数 *function-name* の Argument-1 の値が 16010101 未満か、または 99991231 を超えています。

説明: INTEGER-OF-DATE 関数に対する入力引数が 16010101 未満か、または 99991231 を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ161S

関数 *function-name* の Argument-1 の値が 1601001 未満か、または 9999365 を超えています。

説明: INTEGER-OF-DAY 関数に対する入力引数が 1601001 未満か、または 9999365 を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ162S

関数 *function-name* の Argument-1 の値が 1 未満か、またはプログラムの照合シーケンスの桁数を超えています。

説明: CHAR 関数に対する入力引数が 1 未満か、またはプログラムの照合シーケンスの最高位桁を超えています。

プログラマー応答: 関数の引数が有効範囲内にあることを確認してください。

システム処置: アプリケーションは終了します。

IWZ163S

関数 *function-name* の *Argument-1* の値が 0 未満です。

説明: RANDOM 関数に対する入力引数が 0 未満です。

プログラマー応答: 失敗したステートメントで RANDOM 関数の引数を訂正してください。

システム処置: アプリケーションは終了します。

IWZ165S

行 *line-number* で、*start-position-value* の参照変更開始位置の値が、*function-result* の関数結果領域外の領域を参照しました。

説明: 参照変更指定の開始位置の値が 1 未満か、または参照変更されていた関数結果の現行長を超えています。開始位置の値は、参照変更される関数結果の文字数以下の、正の整数でなければなりません。

プログラマー応答: 参照変更指定の開始位置の値と、実際の関数結果の長さを検査してください。

システム処置: アプリケーションは終了します。

IWZ166S

行 *line-number* にある *length* の参照変更長の値が正数ではないことが、*function-result* の関数結果への参照内で検出されました。

説明: 関数結果の参照変更で指定された長さ値が 0 以下です。この長さ値は正の整数でなければなりません。

プログラマー応答: 長さ値を確認し、適切な修正を行ってください。

システム処置: アプリケーションは終了します。

IWZ167S

行 *line-number* にある *start-position* の参照変更開始位置の値と *length* の長さ値により、*function-result* の関数結果の右端文字を超える参照が行われました。

説明: 参照変更指定の開始位置と長さ値を組み合わせるとアドレス指定された領域が、参照変更される関数結果の終わりを超えています。開始位置と長さ値の合計から 1 を引いた値が、参照変更される関数結果の文字数以下でなければなりません。

プログラマー応答: 参照変更指定の長さ値と実際の関数結果の長さを照合検査し、適切な修正を行ってください。

システム処置: アプリケーションは終了します。

IWZ168W

SYSPUNCH/SYSPCH は、システムの論理出力装置をデフォルトに取ります。対応する環境変数が設定されていません。

説明: COBOL 環境名 (SYSPUNCH/SYSPCH など) は、ACCEPT および DISPLAY ステートメントで使用される簡略名に対応する環境変数名として使用されます。これらは、既存のディレクトリー名ではなくファイルと等しくなるように設定します。環境変数は、`export` コマンドを使用して設定してください。

環境変数は、永続的に設定することも一時的に設定することも可能です。

プログラマー応答: SYSPUNCH/SYSPCH が表示画面をデフォルトに取らないようにするには、対応する環境変数を設定してください。

システム処置: システムのアクションはとられません。

IWZ170S

DISPLAY オペランドのデータ型が正しくありません。

説明: DISPLAY ステートメントのターゲットとして、無効なデータ型が指定されました。

プログラマー応答: 有効なデータ型を指定してください。次のデータ型は無効です。

- USAGE IS PROCEDURE-POINTER で定義されたデータ項目
- USAGE IS OBJECT REFERENCE で定義されたデータ項目
- USAGE IS INDEX で定義されたデータ項目または索引名

システム処置: アプリケーションは終了します。

IWZ171I

string-name は有効なランタイム・オプションではありません。

説明: *string-name* は有効なオプションではありません。

プログラマー応答: CHECK、DEBUG、ERRCOUNT、FILESYS、TRAP、および UPSI が有効なランタイム・オプションです。

システム処置: *string-name* が無視されます。

IWZ172I

ストリング *string-name* は、ランタイム・オプション *option-name* の有効なサブオプションではありません。

説明: *string-name* は、認識済みの値セットの中に含まれていませんでした。

プログラマー応答: 無効なサブオプション *string* を、ランタイム・オプション *option-name* から削除してください。

システム処置: 無効なサブオプションが無視されます。

IWZ173I

ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* の文字長は *number of* でなければなりません。デフォルトが使用されます。

説明: ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* の文字数が無効です。

プログラマー応答: デフォルトを使用したくない場合は、有効な文字長を指定してください。

システム処置: デフォルト値が使用されます。

IWZ174I

ランタイム・オプション *option-name* のサブオプション・ストリング *string-name* に無効な文字が 1 つ以上含まれています。デフォルトが使用されます。

説明: 指定されたサブオプション内で、無効文字が 1 つ以上検出されました。

プログラマー応答: デフォルトを使用したくない場合は、有効な文字を指定してください。

システム処置: デフォルト値が使用されます。

IWZ175S

このシステムでは、ルーチン *routine-name* がサポートされていません。

説明: *routine-name* はサポートされていません。

プログラマー応答:

システム処置: アプリケーションは終了します。

IWZ176S

関数 *function-name* の Argument-1 の値が *decimal-value* を超えています。

説明: argument-1 に対して無効な値が使用されました。

プログラマー応答: argument-1 の値が *decimal-value* 以下であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ177S

関数 *function-name* の Argument-2 の値が *decimal-value* と等しくなっています。

説明: argument-2 に対して無効な値が使用されました。

プログラマー応答: argument-1 の値が *decimal-value* と等しくないことを確認してください。

システム処置: アプリケーションは終了します。

IWZ178S

関数 *function-name* の Argument-1 の値が *decimal-value* 以下です。

説明: Argument-1 に対して無効な値が使用されました。

プログラマー応答: Argument-1 が *decimal-value* より大きいことを確認してください。

システム処置: アプリケーションは終了します。

IWZ179S

関数 *function-name* の Argument-1 の値が *decimal-value* 未満です。

説明: Argument-1 に対して無効な値が使用されました。

プログラマー応答: Argument-1 が *decimal-value* 以上であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ180S

関数 *function-name* の Argument-1 の値が整数ではありません。

説明: Argument-1 に対して無効な値が使用されました。

プログラマー応答: Argument-1 が整数であることを確認してください。

システム処置: アプリケーションは終了します。

IWZ181I

ランタイム・オプション *option-name* の数値ストリング *string* で、無効な文字が検出されました。デフォルトが使用されます。

説明: *string* にすべての 10 進数字が含まれていませんでした。

プログラマー応答: デフォルト値を使用したくない場合は、ランタイム・オプションのストリングにすべての数字が含まれるように修正してください。

システム処置: デフォルトが使用されます。

IWZ182I

ランタイム・オプション *option-name* の数値 *number* が、*min-range* から *max-range* の範囲を超えています。デフォルトが使用されます。

説明: *number* が *min-range* から *max-range* の範囲を超えました。

プログラマー応答: ランタイム・オプションのストリングが有効範囲内になるように修正してください。

システム処置: デフォルトが使用されます。

IWZ183S

`_iwzCOBOLInit` 内の関数名が戻りを行いました。

説明: 実行単位の終了出口ルーチンが、そのルーチンを呼び出した関数 (`function_code` で指定された関数) に戻りました。

プログラマー応答: 実行単位の終了出口ルーチンが、関数に戻るのではなく、`longjump` または `exit` を実行するように関数を書き直してください。

システム処置: アプリケーションは終了します。

IWZ200S

ファイル *file-name* に対する *I/O operation* の実行中に、エラーが検出されました。ファイル状況は *file-status* です。

説明: ファイル入出力操作中に、エラーが検出されました。当該ファイルに対して、ファイル状況が指定されておらず、また該当するエラー宣言も有効になっていません。

プログラマー応答: このメッセージで説明されている状態を修正してください。エラーを検出して、ソース・プログラムで適切な処置をとりたい場合は、当該ファイルに `FILE STATUS` 文節を指定してください。

システム処置: アプリケーションは終了します。

IWZ200S

入出力エラー *error-code* により、`STOP` または `ACCEPT` が失敗しました。実行単位は終了します。

説明: `STOP` または `ACCEPT` ステートメントが失敗しました。

プログラマー応答: STOP または ACCEPT が正当なファイルまたは端末を参照していることを確認してください。

システム処置: アプリケーションは終了します。

IWZ201C

メッセージには、次のような種類があります。

Access Intent List Error.	Address Error.
Concurrent Opens Exceeds Maximum.	Command Check.
Cursor Not Selecting a Record Position.	Duplicate File Name.
Data Stream Syntax Error.	End of File Condition.
Duplicate Key Different Index.	Existing Condition.
Duplicate Key Same Index.	File Handle Not Found.
Duplicate Record Number.	Field Length Error.
File Temporarily Not Available.	File Not Found.
File system cannot be found.	File Damaged.
File Space Not Available.	File is Full.
File Closed with Damage.	File In Use.
Invalid Key Definition.	Function Not Supported.
Invalid Base File Name.	Invalid Access Method.
Key Update Not Allowed by Different Index.	Invalid Data Record.
Key Update Not Allowed by Same Index.	Invalid Key Length.
No Update Intent on Record.	Invalid File Name.
Not Authorized to Use Access Method.	Invalid Request.
Not Authorized to Directory.	Invalid Flag.
Not Authorized to Function.	Object Not Supported.
Not authorized to File.	Record Not Available.
Parameter Value Not Supported.	Record Not Found.
Parameter Not Supported.	Record Inactive.
Record Number Out of Bounds.	Record Damaged.
Record Length Mismatch.	Record In Use.
Resource Limits Reached in Target System.	Update Cursor Error.
Resource Limits Reached in Source System.	

説明: VSAM ファイルに対するファイル入出力操作中に、エラーが検出されました。当該ファイルに対して、ファイル状況が指定されておらず、また該当するエラー宣言も有効になっていません。

プログラマー応答: このメッセージで説明されている状態を修正してください。詳しくは、AIX の「*VSAM in a Distributed Environment*」を参照してください。

システム処置: アプリケーションは終了します。

IWZ203W

有効なコード・ページが DBCS コード・ページではありません。

説明: 非 DBCS コード・ページが有効な状態で、DBCS データを指す参照が行われました。

プログラマー応答: DBCS データの場合は、有効な DBCS コード・ページを指定してください。有効な DBCS コード・ページを次に示します。

国または地域	コード・ページ
日本	IBM-932
韓国	IBM-1363

国または地域	コード・ページ
中華人民共和国 (簡体字)	IBM-1386
台湾 (繁体字)	

注: 上記のコード・ページは、プラットフォームの特定のバージョンまたはリリースではサポートされない場合があります。

システム処置: システムのアクションはとられません。

IWZ204W

ASCII DBCS から EBCDIC DBCS への変換中にエラーが発生しました。

説明: ASCII 文字ストリングから EBCDIC ストリングへの変換中にエラーが検出されたため、漢字または DBCS クラス・テストが失敗しました。

プログラマー応答: 有効なロケールと、テスト対象の ASCII 文字ストリングが一貫していることを検証してください。ロケール設定が正しい場合は、特に処置は必要ありません。クラス・テストは、当該ストリングが漢字または DBCS ではないことを正しく示していると思われます。

システム処置: システムのアクションはとられません。

IWZ221S

コード・ページ *codepage value* の ICU コンバーターをオープンできません。エラー・コードは *error code value* です。

説明: コード・ページと UTF-16 間の変換を行う ICU コンバーターをオープンできません。

プログラマー応答: ICU 変換ライブラリーでサポートされる基本コード・ページ名またはコード・ページ別名をコード・ページ値が識別することを確認してください (*International Components for Unicode: Converter Explorer* を参照)。コード・ページ値が有効な場合は、IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ222S

エラー・コード *error code value* により、ICU を使用したデータ変換が失敗しました。

説明: ICU を使用したデータ変換が失敗しました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ223S

エラー・コード *error code value* により、ICU コンバーターのクローズが失敗しました。

説明: ICU コンバーターのクローズが失敗しました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ224S

ロケール値 *locale value* の ICU コレクターをオープンできません。エラー・コードは *error code value* です。

説明: 当該ロケールの ICU コレクターをオープンできません。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ225S

エラー・コード *error code value* により、ICU を使用した Unicode ケース・マッピング関数が失敗しました。

説明: ICU のケース・マッピング関数が失敗しました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ230W

現行のコード・セット *ASCII codeset-id* から EBCDIC コード・セット *EBCDIC codeset-id* への変換テーブルを使用できません。デフォルトの ASCII/EBCDIC 変換テーブルが使用されます。

説明: アプリケーションに、CHAR(EBCDIC) コンパイラー・オプションを使用してコンパイルされたモジュールがあります。実行時には、現行の ASCII コード・ページから、EBCDIC_CODEPAGE 環境変数で指定された EBCDIC コード・ページへの変換を処理するための変換テーブルが構築されます。指定されたコード・ページに対して使用可能な変換テーブルがないか、または EBCDIC_CODE ページの指定が無効なため、このエラーが発生しました。実行は、ASCII コード・ページ IBM-1252 および EBCDIC コード・ページ IBM-037 に基づくデフォルトの変換テーブルを使用して続けられます。

プログラマー応答: EBCDIC_CODEPAGE 環境変数に有効な値が指定されていることを確認してください。

EBCDIC_CODEPAGE を設定しない場合は、デフォルト値の IBM-037 が使用されます。これは、Enterprise COBOL for z/OS によって使用されるデフォルトのコード・ページです。

システム処置: システムのアクションはとられません。

IWZ230W

指定された EBCDIC コード・ページ *EBCDIC codepage* がロケール *locale* と一貫していませんが、要求どおりに使用されます。

説明: アプリケーションに、CHAR(EBCDIC) コンパイラー・オプションを使用してコンパイルされたモジュールがあります。指定されたコード・ページが現行ロケールと同じ言語でないため、このエラーが発生しました。

プログラマー応答: EBCDIC_CODEPAGE 環境変数がこのロケールに対して有効なことを確認してください。

システム処置: システムのアクションはとられません。

IWZ230W

指定された EBCDIC コード・ページ *EBCDIC codepage* はサポートされていません。デフォルトの EBCDIC コード・ページ *EBCDIC codepage* が使用されます。

説明: アプリケーションに、CHAR(EBCDIC) コンパイラー・オプションを使用してコンパイルされたモジュールがあります。EBCDIC_CODEPAGE 環境変数の指定が無効なため、このエラーが発生しました。実行は、現行ロケールに対応するデフォルトのホスト・コード・ページを使用して続けられます。

プログラマー応答: EBCDIC_CODEPAGE 環境変数に有効な値が指定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ230S

EBCDIC 変換テーブルをオープンできません。

説明: 現行システムのインストールに、デフォルトの ASCII および EBCDIC コード・ページ用の変換テーブルが組み込まれていません。

プログラマー応答: コンパイラーおよびランタイムを再インストールしてください。それでも問題が続く場合は、IBM 担当員に連絡してください。

システム処置: アプリケーションは終了します。

IWZ230S

EBCDIC 変換テーブルを構築できません。

説明: ASCII から EBCDIC への変換テーブルはオープンされましたが、変換が失敗しました。

プログラマー応答: 新規ウィンドウから実行を再試行してください。

システム処置: アプリケーションは終了します。

IWZ230S

メインプログラムは **-host** フラグと **CHAR(NATIVE)** オプションの両方でコンパイルされましたが、これらには互換性がありません。

説明: **-host** フラグと **CHAR(NATIVE)** オプションの両方でのコンパイルはサポートされていません。

プログラマー応答: **-host** フラグまたは **CHAR(NATIVE)** オプションのいずれかを除去してください。**-host** フラグは **CHAR(EBCDIC)** を設定します。

システム処置: アプリケーションは終了します。

IWZ231S

現行のロケール設定の照会が失敗しました。

説明: 実行環境の照会で、有効なロケール設定を識別できませんでした。適切なメッセージ・ファイルにアクセスして照合順序を設定するには、現行ロケールを設定する必要があります。この設定は、日時サービスや EBCDIC 文字のサポートでも使用されます。

プログラマー応答: 次の環境変数の設定を確認してください。

LANG これは、ご使用のマシンにインストールされているロケールに設定してください。有効な値のリストを取得するには、`locale -a` と入力してください。デフォルトは `en_US` です。

システム処置: アプリケーションは終了します。

IWZ232W

メッセージには、次のような種類があります。

- プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* から EBCDIC への変換中にエラーが発生しました。
- プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* から ASCII への変換中にエラーが発生しました。
- プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* の EBCDIC への変換中にエラーが発生しました。
- プログラム *program-name* の行番号 *decimal-value* で、データ項目 *data-name* の ASCII への変換中にエラーが発生しました。
- プログラム *program-name* の行番号 *decimal-value* で、ASCII から EBCDIC への変換中にエラーが発生しました。
- プログラム *program-name* の行番号 *decimal-value* で、EBCDIC から ASCII への変換中にエラーが発生しました。

説明: CHAR(EBCDIC) コンパイラー・オプションで要求されたとおりに、ID 内のデータを ASCII 形式と EBCDIC 形式間で変換できませんでした。

プログラマー応答: 適切な ASCII および EBCDIC ロケールがインストール済みで、選択されていることを確認してください。ID 内のデータが有効で、ASCII と EBCDIC の両方の形式で表現できることを確認してください。

システム処置: システムのアクションはとられません。データは未変換形式のままになります。

IWZ233S

COBOL プログラム *program-name* は **NOTHREAD** コンパイラー・オプションを使用してコンパイルされ、マルチスレッド化されたライブラリーにリンクされているため、実行できません。

説明: CICS 以外のプログラムは、マルチスレッド化されたライブラリーにリンクされている場合、**THREAD** コンパイラー・オプションを使用してコンパイルする必要があります。

プログラマー応答: プログラムは、**THREAD** コンパイラー・オプションを使用してコンパイルするか、(cob2_r または cob2_j ではなく) cob2 を使用して静的ライブラリーにリンクしてください。

システム処置: アプリケーションは終了します。

IWZ240S

プログラム *program-name* の基本年が 1900 から 1999 の有効範囲内にありません。スライディング・ウィンドウ値 *window-value* は、*base-year* の基本年になります。

説明: 現行年と、YEARWINDOW コンパイラー・オプションで指定されたスライディング・ウィンドウ値を使用して 100 年間の範囲を計算しましたが、100 年間の基本となる年が 1900 から 1999 の有効範囲内にありませんでした。

プログラマー応答: アプリケーション設計を調べて、YEARWINDOW オプション値を変更できるかどうかを判別してください。YEARWINDOW オプション値を変更してアプリケーションを実行できる場合は、適切な YEARWINDOW オプション値を使用してプログラムをコンパイルしてください。YEARWINDOW オプション値を変更してアプリケーションを実行できない場合は、すべての日付フィールドを拡張日付に変換してから、NODATEPROC を使用してプログラムをコンパイルしてください。

システム処置: アプリケーションは終了します。

IWZ241S

現行年が、プログラム *program-name* に使用されている *year-start* から *year-end* の 100 年間隔内にありません。

説明: 現行年が、YEARWINDOW コンパイラー・オプション値で指定された 100 年間の固定範囲内にありませんでした。

例えば、YEARWINDOW(1920) を使用して COBOL プログラムをコンパイルした場合、そのプログラムに使用される 100 年間の範囲は 1920 から 2019 となります。このプログラムを 2020 年に実行すると、現行年がこの 100 年間の範囲内にないため、エラー・メッセージが出されます。

プログラマー応答: アプリケーション設計を調べて、YEARWINDOW オプション値を変更できるかどうかを判別してください。YEARWINDOW オプション値を変更してアプリケーションを実行できる場合は、適切な YEARWINDOW オプション値を使用してプログラムをコンパイルしてください。YEARWINDOW オプション値を変更してアプリケーションを実行できない場合は、すべての日付フィールドを拡張日付に変換してから、NODATEPROC を使用してプログラムをコンパイルしてください。

システム処置: アプリケーションは終了します。

IWZ242S

XML PARSE ステートメントを開始しようとしたますが、この操作は無効です。

説明: COBOL プログラムによって開始された XML PARSE ステートメントがすでに進行中のときに、同じ COBOL プログラムが別の XML PARSE ステートメントを実行しようとした。1 回の COBOL プログラム呼び出しでアクティブにできる XML PARSE ステートメントは 1 つだけです。

プログラマー応答: 同じ COBOL プログラム内から別の XML PARSE ステートメントを開始しないようにアプリケーションを変更してください。

システム処置: アプリケーションは終了します。

IWZ243S

XML PARSE ステートメントを終了しようとしたますが、この操作は無効です。

説明: COBOL プログラムによって開始された XML PARSE ステートメントの進行中に、次のいずれかが試行されました。

- XML PARSE ステートメントを開始した COBOL プログラム内で、GOBACK または EXIT PROGRAM ステートメントが発行された。
- XML PARSE ステートメントを開始したプログラムと関連付けられたユーザー・ハンドラーが、条件ハンドラー再開カーソルを移動した後で、アプリケーションを再開した。

プログラマー応答: 上記のメソッドを使用せずに XML PARSE ステートメントを終了するようにアプリケーションを変更してください。

システム処置: アプリケーションは終了します。

IWZ250S

内部エラー: JNI_GetCreatedJavaVMs への呼び出しにより、エラー、戻りコード *nn* が戻されました。

説明: JNI_GetCreatedJavaVMs 関数の呼び出しにより、戻りコード *nn* のエラーが戻されました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションの実行は終了します。

IWZ251S

内部エラー: *n* 個のアクティブ Java VM が検出されましたが、期待される数は 1 つだけです。

説明: 複数のアクティブ Java VM が検出されましたが、期待される Java VM は 1 つだけです。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションの実行は終了します。

IWZ253S

nn 個を超える JVM 初期化オプションが指定されました。

説明: COBJVMINIOPTIONS 環境変数で指定した Java 初期化オプションの数が最大許容数を超過しています。上限は 256 です。

プログラマー応答: COBJVMINIOPTIONS 環境変数に指定するオプションの数を減らしてください。

システム処置: アプリケーションの実行は終了します。

IWZ254S

内部エラー: JNI_CreateJavaVM への呼び出しによりエラーが戻されました。

説明: JNI_CreateJavaVM の呼び出しによりエラーが戻されました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションの実行は終了します。

IWZ255S

内部エラー: 現行のスレッドが JVM に接続されていないため、GetEnv への呼び出しにより、コード *nm* が戻されました。

説明: 現行のスレッドが JVM に接続されていないため、GetEnv への呼び出しにより、コード *nm* が戻されました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションの実行は終了します。

IWZ256S

内部エラー: JVM バージョンがサポートされていないため、GetEnv への呼び出しにより、コード *nm* が戻されました。

説明: JVM バージョンがサポートされていないため、GetEnv への呼び出しにより、コード *nm* が戻されました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションの実行は終了します。

IWZ257S

内部エラー: GetEnv への呼び出しにより、認識されていない戻りコード *nm* が戻されました。

説明: GetEnv への呼び出しにより、認識されていない戻りコード *nm* が戻されました。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションの実行は終了します。

IWZ258S

内部エラー: `GetByteArrayElements` が、インスタンス・データを指すポインターを取得できませんでした。

説明: `GetByteArrayElements` サービスが、インスタンス・データを指すポインターを取得できませんでした。

プログラマー応答: IBM 担当員に連絡してください。

システム処置: アプリケーションの実行は終了します。

IWZ259S

インスタンス・データを指す直接ポインターを取得できません。インストール済みの JVM では、`pinned` (滞留) されるバイト配列はサポートされていません。

説明: `pinned` (滞留) されるバイト配列に対応していない、サポートされていない JVM (Sun 1.4.1 など) が使用されています。`pinning` (滞留) について詳しくは、このセクションの末尾にある `Java Native Interface` についての関連参照を参照してください。

プログラマー応答: サポートされる JVM を実行してください。

システム処置: アプリケーションの実行は終了します。

IWZ260S

Java クラス `name` を検出できませんでした。

説明: プログラムが、`CLASSPATH` 環境変数で定義または指定されていないクラス名を参照しています。

プログラマー応答: `name` を参照するプログラムを検査し、その参照を修正するか、欠落している `name.class` クラスを指定してください。

システム処置: アプリケーションの実行は終了します。

IWZ813S

使用可能なストレージが不十分なため、ストレージ取得要求を満たすことができません。

説明: 使用可能なフリー・ストレージが不十分なため、ストレージ取得要求または再割り振り要求を満たすことができません。このメッセージは、ストレージ管理でオペレーティング・システムから十分なストレージを取得できなかったことを意味します。

プログラマー応答: アプリケーションを実行できるだけの十分なストレージを確保してください。

システム処置: ストレージは割り振られません。

シンボリック・フィードバック・コード: CEE0PD

IWZ901S

メッセージには、次のような種類があります。

- 重大エラーまたはクリティカル・エラーが発生したため、プログラムが終了します。
- プログラムの終了: **ERRCOUNT** を超えるエラー数が発生しました。

説明: それぞれの重大メッセージまたはクリティカル・メッセージの後には、IWZ901 メッセージが続きます。ERRCOUNT ランタイム・オプションを使用した場合に、警告メッセージの数が **ERRCOUNT** を超えると、IWZ901 メッセージも出されます。

プログラマー応答: 重大メッセージまたはクリティカル・メッセージを参照して、**ERRCOUNT** の値を増やしてください。

システム処置: アプリケーションは終了します。

IWZ902S

システムが 10 進数除算例外を検出しました。

説明: ある数値を 0 で除算しようとしたことが検出されました。

プログラマー応答: プログラムを修正してください。例えば、フラグ付きのステートメントに **ON SIZE ERROR** を追加します。

システム処置: アプリケーションは終了します。

IWZ903S

システムがデータ例外を検出しました。

説明: データに無効値が含まれているため、パック 10 進数データまたはゾーン 10 進数データに対する操作が失敗しました。

プログラマー応答: データが有効なパック 10 進数データまたはゾーン 10 進数データであることを確認してください。

システム処置: アプリケーションは終了します。

IWZ907S

メッセージには、次のような種類があります。

- ストレージが不十分です。
- ストレージが不十分です。storage 用に *number-bytes* バイトのスペースを取得できません。

説明: ランタイム・ライブラリーが仮想メモリー・スペースを要求しましたが、オペレーティング・システムがこの要求を拒否しました。

プログラマー応答: プログラムが大量の仮想メモリーを使用するため、スペースが不足しています。一般に、この問題の原因は特定のステートメントではなく、プログラム全体に関連しています。OCCURS 文節の使用状況を調べて、テーブルのサイズを減らしてください。

システム処置: アプリケーションは終了します。

IWZ993W

ストレージが不十分です。メッセージ *message-number* 用のスペースを検出できません。

説明: ランタイム・ライブラリーが仮想メモリー・スペースを要求しましたが、オペレーティング・システムがこの要求を拒否しました。

プログラマー応答: プログラムが大量の仮想メモリーを使用するため、スペースが不足しています。一般に、この問題の原因は特定のステートメントではなく、プログラム全体に関連しています。OCCURS 文節の使用状況を調べて、テーブルのサイズを減らしてください。

システム処置: システムのアクションはとられません。

IWZ994W

メッセージ・カタログ内でメッセージ *message-number* を検出できません。

説明: ランタイム・ライブラリーが、メッセージ・カタログ自体またはメッセージ・カタログ内の特定のメッセージを検出できません。

プログラマー応答: COBOL ライブラリーおよびメッセージが正しくインストールされていることと、LANG および NLSPATH が正しく指定されていることを確認してください。

システム処置: システムのアクションはとられません。

IWZ995C

メッセージには、次のような種類があります。

- オフセット *0xoffset-value* でルーチン *routine-name* の実行中に、*system exception* シグナルを受信しました。
- *0xoffset-value* の場所でコードの実行中に、*system exception* シグナルを受信しました。
- *system exception* シグナルを受信しました。場所を判別できませんでした。

説明: オペレーティング・システムが、無許可のアクション (保護記憶域にデータを格納しようとしたなど) を検出したか、または割り込みキー (一般には Control + C キーだが、再構成はできない) が押されたことを検出しました。

プログラマー応答: 無許可のアクションがシグナルの原因の場合は、デバッガーでプログラムを実行すると、エラーの発生場所に関する詳細情報を得ることができます。このタイプのエラーの例としては、無効な値を持つポインターなどが挙げられます。

システム処置: アプリケーションは終了します。

IWZ2502S

システムから UTC/GMT を使用できませんでした。

説明: システム・クロックが無効な状態になっていたため、CEEUTC または CEEGMT の呼び出しが失敗しました。現在時刻を判別できませんでした。

プログラマー応答: システム・クロックが無効な状態になっていることをシステム・サポート担当者に連絡してください。

システム処置: 出力値がすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2E6

IWZ2503S

UTC/GMT から現地時間までのオフセットをシステムから使用できませんでした。

説明: (1) 現行のオペレーティング・システムを判別できなかったか、(2) オペレーティング・システムの制御ブロックにある時間帯フィールドに無効なデータが含まれている可能性があるため、CEEGMT の呼び出しが失敗しました。

プログラマー応答: オペレーティング・システムに格納されている現地時間のオフセットに無効なデータが含まれている可能性があることをシステム・サポート担当者に連絡してください。

システム処置: 出力値がすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2E7

IWZ2505S

CEEDATM または CEESECI への呼び出し内の `input_seconds` 値が、対応範囲内にありませんでした。

説明: CEEDATM または CEESECI の呼び出しで渡された `input_seconds` 値が、86,400.0 から 265,621,679,999.999 の範囲内の浮動小数点数ではありませんでした。入力パラメーターは、1582 年 10 月 14 日の 00:00:00 から数えた秒数で表す必要があります。ここでサポートされる最初の日時は 1582 年 10 月 15 日の 00:00:00.000、最後の日時は 9999 年 12 月 31 日の 23:59:59.999 です。

プログラマー応答: 入力パラメーターに含まれている浮動小数点値が 86,400.0 から 265,621,679,999.999 の範囲内にあることを確認してください。

システム処置: CEEDATM の場合は、出力値がブランクに設定されます。CEESECI の場合は、出力パラメーターがすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2E9

IWZ2506S

CEEDATM に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、<CCCCCCCC>) が使用されていましたが、入力された秒数値が対応範囲内にありませんでした。元号を判別できませんでした。

説明: CEEDATM の呼び出しでは、ピクチャー・ストリングは入力値が元号に変換されることを示しますが、指定された入力値は、サポートされる元号の範囲内にありません。

プログラマー応答: サポートされる元号の範囲内にある有効な秒数値が入力値に含まれていることを確認してください。

システム処置: 出力値がブランクに設定されます。

IWZ2507S

CEEDAYS または CEESECS に渡されたデータが不十分です。リリアン日付の値は計算されませんでした。

説明: CEEDAYS または CEESECS の呼び出しで渡されたピクチャー・ストリングに、十分な情報が含まれていませんでした。例えば、CEEDAYS または CEESECS の呼び出しでピクチャー・ストリング 'MM/DD' (月と日のみ) を使用すると、年の値がないためエラーとなります。リリアン日付の値を計算するために最低限必要な情報は、(1) 月、日、年、または (2) 年、ユリウス日のいずれかです。

プログラマー応答: CEEDAYS または CEESECS の呼び出しで指定されたピクチャー・ストリングに少なくとも、(1) 月、日、年、または (2) 年、ユリウス日のいずれかの入力ストリング内の場所が指定されていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EB

IWZ2508S

CEEDAYS または CEESECS に渡された日付値が無効です。

説明: CEEDAYS または CEESECS の呼び出しで、DD または DDD フィールドの値が当該年/月に対して無効です。例えば、1990 年はうるう年ではないため、'MM/DD/YY' の値が '02/29/90' の場合や 'YYYY.DDD' の値が '1990.366' の場合は無効となります。また、6 月 31 日や 1 月 0 日など、存在しない日付値を指定した場合にも、このコードが戻されることがあります。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、入力データに有効な日付が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EC

IWZ2509S

CEEDAYS または CEESECS に渡された元号が認識されませんでした。

説明: CEEDAYS または CEESECS の呼び出しで渡された <JJJJ>、<CCCC>、または <CCCCCCCC> フィールドの値に、サポートされる元号名が含まれていません。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、元号名のスペルが正しいことを確認してください。元号名は、正しい DBCS ストリングでなければならないことに注意してください。'<' の位置には、元号名の先頭バイトが含まれていなければなりません。

システム処置: 出力値が 0 に設定されます。

IWZ2510S

CEEISEC または CEESECS への呼び出しで時間の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しで、時のパラメーターに 0 から 23 までの範囲内の数値が含まれていませんでした。あるいは、(2) CEESECS の呼び出しで、HH (時) フィールドの値に 0 から 23 までの範囲内の数値が含まれていないか、または「AP」(午前/午後) フィールドが指定されていて、HH フィールドに 1 から 12 までの範囲内の数値が含まれていません。

プログラマー応答: CEEISEC の場合は、時のパラメーターに 0 から 23 までの範囲内の整数が含まれていることを確認してください。CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することと、時のフィールドに 0 から 23 (「AP」フィールドを使用している場合は 1 から 12) の範囲内の値が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィールドバック・コード: CEE2EE

IWZ2511S

CEEISEC の呼び出しで渡された日のパラメーターが、指定された年および月に対して無効です。

説明: CEEISEC の呼び出しで渡された日のパラメーターに、有効な日数値が含まれていませんでした。年、月、日の組み合わせが、無効な日付値になっています。例えば、1990 年 2 月 29 日、6 月 31 日、0 日などの日付は無効です。

プログラマー応答: 日のパラメーターに 1 から 31 の範囲内の整数が含まれていることと、年、月、日の組み合わせが有効な日付を表していることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィールドバック・コード: CEE2EF

IWZ2512S

CEEDATE または CEEDYWK への呼び出しで渡されたりリアン日付値が、対応範囲内にありませんでした。

説明: CEEDATE または CEEDYWK の呼び出しで渡されたりリアン日付値が、1 から 3,074,324 の範囲内の数値ではありませんでした。

プログラマー応答: 入力パラメーターに 1 から 3,074,324 の範囲内の整数が含まれていることを確認してください。

システム処置: 出力値がブランクに設定されます。

シンボリック・フィードバック・コード: CEE2EG

IWZ2513S

CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、対応範囲内にありませんでした。

説明: CEEISEC、CEEDAYS、CEESECS のいずれかの呼び出しで渡された入力日付が、1582 年 10 月 15 日より前か、9999 年 12 月 31 日より後に設定されていました。

プログラマー応答: CEEISEC の場合は、年、月、日のパラメーターが、1582 年 10 月 15 日以降の日付になっていることを確認してください。CEEDAYS および CEESECS の場合は、入力される日付の形式がピクチャー・ストリング指定と一致することと、入力される日付が対応範囲内にあることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EH

IWZ2514S

CEEISEC の呼び出しで渡された年の値が、対応範囲内にありませんでした。

説明: CEEISEC の呼び出しで渡された年のパラメーターに、1582 から 9999 の範囲内の数値が含まれていませんでした。

プログラマー応答: 年のパラメーターに有効なデータが含まれていることと、年のパラメーターに世紀が含まれている (例えば、90 年ではなく 1990 年と指定している) ことを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EI

IWZ2515S

CEEISEC 呼び出し内のミリ秒の値が認識されませんでした。

説明: CEEISEC の呼び出しで、ミリ秒のパラメーター (*input_milliseconds*) に 0 から 999 の範囲内の数値が含まれていませんでした。

プログラマー応答: ミリ秒のパラメーターに 0 から 999 の範囲内の整数が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EJ

IWZ2516S

CEEISEC 呼び出し内の分の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しの場合は、分のパラメーター (*input_minutes*) に 0 から 59 の範囲内の数値が含まれていませんでした。(2) CEESECS の呼び出しの場合は、MI (分) フィールドの値に 0 から 59 の範囲内の数値が含まれていませんでした。

プログラマー応答: CEEISEC の場合は、分のパラメーターに 0 から 59 の範囲内の整数が含まれていることを確認してください。CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することと、分のフィールドに 0 から 59 の範囲内の値が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EK

IWZ2517S

CEEISEC 呼び出し内の月の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しの場合は、月のパラメーター (*input_month*) に 1 から 12 の範囲内の数値が含まれていませんでした。(2) CEEDAYS または CEESECS の呼び出しの場合は、MM フィールドの値に 1 から 12 の範囲内の数値が含まれていないか、あるいは MMM や MMMM などのフィールドの値に、現在アクティブな各国語で正しいスペルの月の名前または省略後が含まれていませんでした。

プログラマー応答: CEEISEC の場合は、月のパラメーターに 1 から 12 の範囲内の整数が含まれていることを確認してください。CEEDAYS および CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することを確認してください。MM フィールドの場合は、入力値が 1 から 12 の範囲内にあることを確認してください。月名 (MMM、MMMM など) を指定する場合は、その月名のスペルまたは省略語が、現在アクティブな各国語で正しく指定されていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EL

IWZ2518S

日時サービスへの呼び出しに無効なピクチャー・ストリングが指定されました。

説明: いずれかの日時サービスの呼び出しで指定されたピクチャー・ストリングが無効です。指定できるのは 1 つの元号文字ストリングだけです。

プログラマー応答: ピクチャー・ストリングに有効なデータが含まれていることを確認してください。ピクチャー・ストリングに複数の元号記述子 (<JJJJ> と <CCCC> の両方など) が含まれている場合は、一方の元号だけを使用するようにピクチャー・ストリングを変更してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EM

IWZ2519S

CEEISEC 呼び出し内の秒の値が認識されませんでした。

説明: (1) CEEISEC の呼び出しの場合は、秒のパラメーター (*input_seconds*) に 0 から 59 の範囲内の数値が含まれていませんでした。(2) CEESECS の呼び出しの場合は、SS (秒) フィールドの値に 0 から 59 の範囲内の数値が含まれていませんでした。

プログラマー応答: CEEISEC の場合は、秒のパラメーターに 0 から 59 の範囲内の整数が含まれていることを確認してください。CEESECS の場合は、入力データの形式がピクチャー・ストリング指定と一致することと、秒のフィールドに 0 から 59 の範囲内の値が含まれていることを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EN

IWZ2520S

CEEDAYS が数値フィールド内に非数値データを検出したか、あるいは日付ストリングとピクチャー・ストリングが一致しませんでした。

説明: CEEDAYS の呼び出しで渡された入力値が、ピクチャー指定で記述された形式ではありませんでした (例えば、数字のみが期待される場所に非数字があるなど)。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、数値フィールドに数値データしか含まれていないことを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2EO

IWZ2521S

CEEDAYS または CEESECS に渡された <JJJJ>、<CCCC>、または <CCCCCCCC> の元号年数値がゼロでした。

説明: CEEDAYS または CEESECS の呼び出しで、YY または ZYY ピクチャー・トークンが指定されている場合で、なおかつピクチャー・ストリングにいずれかの元号トークン (<CCCC> や <JJJJ> など) が含まれている場合は、年の値が 1 以上で、その元号に対して有効な値でなければなりません。この場合、YY または ZYY フィールドは当該元号における年を意味します。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、入力データが有効なことを確認してください。

システム処置: 出力値が 0 に設定されます。

IWZ2522S

CEEDATE に渡されたピクチャー・ストリング内に元号 (<JJJJ>、<CCCC>、<CCCCCCCC>) が使用されていましたが、リリアン日付値が対応範囲内にありませんでした。元号を判別できませんでした。

説明: CEEDATE の呼び出しでは、ピクチャー・ストリングはリリアン日付が元号に変換されることを示しますが、リリアン日付がサポートされる元号の範囲内にありません。

プログラマー応答: サポートされる元号の範囲内にある有効なリリアン日付値が入力値に含まれていることを確認してください。

システム処置: 出力値がブランクに設定されます。

IWZ2525S

CEESECS が数値フィールド内に非数値データを検出したか、あるいはタイム・スタンプ・ストリングとピクチャー・ストリングが一致しませんでした。

説明: CEESECS の呼び出しで渡された入力値が、ピクチャー指定で記述された形式ではありませんでした。例えば、数字のみが期待される場所に非数字がある、a.m./p.m. フィールド (AP、A.P. など) に 'AM' または 'PM' のストリングが含まれていないなどが考えられます。

プログラマー応答: 入力データの形式がピクチャー・ストリング指定と一致していることと、数値フィールドに数値データしか含まれていないことを確認してください。

システム処置: 出力値が 0 に設定されます。

シンボリック・フィードバック・コード: CEE2ET

IWZ2526S

CEEDATE によって戻された日付ストリングが切り捨てられました。

説明: CEEDATE の呼び出しで、出力ストリングのサイズが足りないため、フォーマットした日付値を格納できませんでした。

プログラマー応答: 出力ストリングのデータ項目が、フォーマットされた日付全体を格納できるだけの十分なサイズになっていることを確認してください。出力パラメーターが、少なくともピクチャー・ストリング・パラメーターと同じ長さになっていることを確認してください。

システム処置: 出力値が出力パラメーターの長さまで切り詰められます。

シンボリック・フィードバック・コード: CEE2EU

IWZ2527S

CEEDATM によって戻されたタイム・スタンプ・ストリングが切り捨てられました。

説明: CEEDATM の呼び出しで、出力ストリングのサイズが足りないため、フォーマットしたタイム・スタンプ値を格納できませんでした。

プログラマー応答: 出力ストリングのデータ項目が、フォーマットされたタイム・スタンプ全体を格納できるだけの十分なサイズになっていることを確認してください。出力パラメーターが、少なくともピクチャー・ストリング・パラメーターと同じ長さになっていることを確認してください。

システム処置: 出力値が出力パラメーターの長さまで切り詰められます。

シンボリック・フィードバック・コード: CEE2EV

IWZ2531S

システムから現地時間を使用できませんでした。

説明: システム・クロックが無効な状態になっていたため、CEELOCT の呼び出しが失敗しました。現在時刻を判別できません。

プログラマー応答: システム・クロックが無効な状態になっていることをシステム・サポート担当者に連絡してください。

システム処置: 出力値がすべて 0 に設定されます。

シンボリック・フィードバック・コード: CEE2F3

IWZ2533S

CEESCEN に渡された値が 0 から 100 の範囲内にありませんでした。

説明: CEESCEN の呼び出しで渡された *century_start* の値が、0 から 100 の範囲内にありませんでした。

プログラマー応答: 入力パラメーターが有効範囲内にあることを確認してください。

システム処置: システムのアクションはとられません。すべての 2 桁年号に対して想定される 100 年間の範囲は変わりません。

シンボリック・フィードバック・コード: CEE2F5

IWZ2534W

CEEDATE または CEEDATM への呼び出しで、月または曜日名に対して指定されたフィールド幅が不十分です。出力はブランクに設定されました。

説明: CEEDATE または CEEDATM 呼び出し可能サービスで、ピクチャー・ストリングに正しいスペルの月名や曜日名が要求される MMM、MMMMMZ、WWW、Wwww などが含まれているにもかかわらず、現在フォーマットされている月名に含まれている文字数が指定フィールド内に収まらない場合は、このメッセージが出されます。

プログラマー応答: フォーマットされる最長の月名または曜日名を格納できるだけの十分な数の M または W を指定して、フィールド幅を増やしてください。

システム処置: 幅が不十分な月名および曜日名フィールドは、ブランクに設定されます。残りの出力ストリングは影響を受けません。処理を続行します。

シンボリック・フィードバック・コード: CEE2F6

関連概念

631 ページの『付録 D. 中間結果および算術精度』

関連タスク

225 ページの『環境変数の設定』

243 ページの『コンパイル・エラー・メッセージのリストの生成』

関連参照

The Java Native Interface

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確認できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _年を入れる_. All rights reserved.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

以下は、International Business Machines Corporation の米国およびその他の国における商標です。

IBM
IBM logo
ibm.com
AIX
ATCICS
COBOL/370
Database 2
DB2
Encina
Language Environment
MVS
OS/390
TXSeriesz/OS
zSeries

Intel は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc. の米国およびその他の国における商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft および Windows は、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

用語集

この用語集に記載されている用語は、COBOL における意味に従って定義されています。これらの用語は、他の言語では同じ意味を持つことも、持たないこともあります。

この用語集には、以下の資料からの用語および定義が記載されています。

- 「ANSI INCITS 23-1985, Programming languages - COBOL」 (「ANSI INCITS 23a-1989, Programming Languages - COBOL - Intrinsic Function Module for COBOL」 および 「ANSI INCITS 23b-1993, Programming Languages - Correction Amendment for COBOL」 で改訂)
- 「ANSI X3.172-2002, American National Standard Dictionary for Information Systems

米国標準規格 (ANS) の定義の前にはアスタリスク (*) を付けています。

この用語集には、Sun Microsystems, Inc が Java および J2EE の用語集用に作成した定義が含まれています。Sun による定義には、その旨が示されています。

ア

アクセス・モード (* access mode)

ファイル内でレコードが操作される方式。

遊びバイト (slack bytes)

一部の数値項目の位置合わせが正しく行われるように、データ項目相互間またはレコード相互間に挿入されるバイト。遊びバイトには意味のあるデータは含まれない。コンパイラーによって挿入される場合もあれば、プログラマーが挿入する必要がある場合もある。SYNCHRONIZED 節は、正しいアライメントが必要な場合に遊びバイトを挿入するようにコンパイラーに指示する。レコード間遊びバイトは、プログラマーが挿入する。

暗黙の範囲終了符号 (* implicit scope terminator)

終了していないステートメントが前にある場合、その範囲を区切る分離文字ピリオド。または、前にある句の中に含まれるス

テートメントがある場合、そのステートメントの範囲の終わりをそれが現れることによって示すステートメントの句。

異常終了 (abend)

プログラムの異常終了。

移植する、ポート (port)

(1) 異なるプラットフォームで実行できるようにコンピューター・プログラムを変更すること。(2) インターネット・プロトコルでは、Transmission Control Protocol (TCP) プロトコルまたは User Datagram Protocol (UDP) プロトコルと高水準のプロトコルまたはアプリケーションの間の特定の論理結合子。ポートはポート番号によって識別される。

移植性 (portability)

あるアプリケーション・プラットフォームから別のアプリケーション・プラットフォームに、ソース・プログラムに比較的わずかな変更を加えるだけでアプリケーション・プログラムを移行できる能力。

インスタンス・データ (instance data)

オブジェクトの状態を定義するデータ。クラスによって導入されるインスタンス・データは、クラス定義の OBJECT 段落の DATA DIVISION のWORKING-STORAGE SECTION に定義される。オブジェクトの状態には、クラスが導入した、現行クラスによって継承されているインスタンス変数の状態も含まれる。インスタンス・データの個々のコピーは、各オブジェクト・インスタンスごとに作成される。

隠蔽 (hide)

サブクラスのファクトリーまたは静的メソッド (親クラスから継承された) を再定義すること。

インライン (inline)

プログラムでは、ルーチン、サブルーチン、または他のプログラムに分岐することなく、順次に行われる命令。

ウィンドウ表示西暦年 (windowed year)

2桁の年だけから構成される日付フィールド。この2桁の年は、世紀ウィンドウを使用して解釈できる。例えば、08は2008として解釈できる。世紀ウィンドウ (century window) も参照。拡張西暦年 (expanded year) と比較。

ウィンドウ表示日付フィールド (windowed date field) ウィンドウ表示 (2桁) 年を含む日付フィールド。日付フィールド (date field) およびウィンドウ表示西暦年 (windowed year) も参照。

埋め込み文字 (padding character)

物理レコード内の未使用文字位置を埋めるのに使用される英数字または国別文字。

英字 (* alphabetic character)

文字または空白文字。

英字データ項目 (alphabetic data item)

記号 A のみを含む PICTURE 文字ストリングが記述されたデータ項目。英字データ項目は USAGE DISPLAY を持ちます。

英字名 (* alphabet-name)

ENVIRONMENT DIVISION の SPECIAL-NAMES 段落のユーザー定義語であり、特定の文字セットまたは照合シーケンス (あるいはその両方) に名前を割り当てるもの。

英数字 (* alphanumeric character)

コンピューターの1バイト文字セットの任意の文字。

英数字関数 (* alphanumeric function)

コンピューターの英数字セットからの1つ以上の文字のストリングで値が構成されている関数。

英数字グループ項目 (alphanumeric group item)

GROUP-USAGE NATIONAL 節なしで定義されたグループ項目。INSPECT、STRING、および UNSTRING などの操作の場合、英数字グループ項目は、実際のグループの内容にかかわらず、その内容すべてが USAGE DISPLAY として記述されているかのように処理されます。グループ内の基本項目を処理する必要のある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、または INITIALIZE など) の場合、英数字グル

ープ項目はグループ・セマンティクスを使用して処理されます。

英数字データ項目 (alphanumeric data item)

暗黙的または明示的に USAGE DISPLAY として記述された、カテゴリ-英数字、英数字編集、または数字編集を持つデータ項目を指す一般的な呼び方。

英数字編集データ項目 (alphanumeric-edited data item)

少なくとも1つの記号 A または X のインスタンスおよび少なくとも1つの単純挿入記号 B、0、または / を含んでいる、PICTURE 文字ストリングで記述されたデータ項目。英数字編集データ項目は USAGE DISPLAY を持ちます。

英数字リテラル (alphanumeric literal)

次のセットからの開始区切り文字を有するリテラル。'、"、X'、X"、Z'、または Z"。この文字ストリングには、コンピューターの有する文字セットの任意の文字を含めることができる。

エレメント (テキスト・エレメント) (element (text element))

1つのデータ項目または動詞の記述などのようなテキスト・ストリングの1つの論理単位で、その前にエレメント・タイプを識別する固有のコードが付けられたもの。

エンコード・ユニット (encoding unit)

文字エンコード・ユニット (character encoding unit) を参照。

演算、操作 (operation)

オブジェクトに関して要求できるサービ

演算符号 (* operational sign)

値が正であるか負であるかを示すために数字データ項目または数字リテラルに付けられる代数符号。

オーバーフロー条件 (overflow condition)

ある演算結果の一部が意図した記憶単位の容量を超えた場合に発生する条件。

オープン・モード (* open mode)

OPEN ステートメントが実行されてから、REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。個々のオープン・モードは、

OPEN ステートメントの中で、INPUT、OUTPUT、I-O、または EXTEND のいずれかとして指定する。

オブジェクト (object)

状態 (そのデータ値) および演算 (そのメソッド) を持つエンティティ。オブジェクトは状態と動作をカプセル化する手段である。クラス内の各オブジェクトは、そのクラスの 1 つのインスタンスであると言われる。

オブジェクト・インスタンス (object instance)

オブジェクト (*object*) を参照。

オブジェクト・コード (object code)

コンパイラまたはアセンブラからの出力。それ自体が実行可能なマシン・コードか、またはその種のコードの作成を目的として処理するのに適する。

オブジェクト・コンピューター記入項目 (* object computer entry)

ENVIRONMENT DIVISION の OBJECT-COMPUTER 段落内の記入項目。この記入項目には、オブジェクト・プログラムが実行されるコンピューター環境を記述する節が入っている。

オブジェクト・プログラム (object program)

問題を解決するためにデータと相互に作用することを目的とする実行可能なマシン言語命令とその他の要素の集合またはグループ。このコンテキストでは、オブジェクト・プログラムとは一般に、COBOL コンパイラがソース・プログラムまたはクラス定義を操作した結果得られるマシン言語である。あいまいになる危険がない場合には、オブジェクト・プログラム という用語の代わりにプログラム というワードだけが使用される。

オブジェクト参照 (object reference)

クラスのインスタンスを識別する値。クラスが指定されなかった場合、オブジェクト参照は一般的なものとなり、任意のクラスのインスタンスに適用できる。

オブジェクト時 (* object time)

オブジェクト・プログラムが実行される時。実行時 (*run time*) と同義。

オブジェクト指向プログラミング (object-oriented programming)

カプセル化および継承の概念に基づいたプログラミング・アプローチ。プロシージャ型プログラミング技法とは異なり、オブジェクト指向プログラミングでは、何かが達成される方法ではなく、問題を含むデータ・オブジェクトとその操作方法に重点を置く。

オプション・ファイル (optional file)

オブジェクト・プログラムが実行されるたびに必ずしも使用可能でなくてもよいものとして宣言されているファイル。

オプション・ワード (* optional word)

言語を読みやすくする目的でのみ特定の形式で含まれる予約語。このようなワードが表示されている形式をソース単位内で使用する場合、そのワードの有無はユーザーが選択できる。

オペランド (* operand)

(1) オペランドの一般的な定義は、「操作の対象となるコンポーネント」である。(2) 本書の目的に沿った言い方をすれば、ステートメントや記入項目の形式中に現れる小文字または日本語で書かれた語 (または語群) はオペランドと見なされ、そのオペランドによって指示されたデータに対して暗黙の参照を行う。

カ

ガーベッジ・コレクション (garbage collection)

参照されなくなったオブジェクトのメモリーを、Java ランタイム・システムが自動的に解放すること。

下位終了 (* low-order end)

文字ストリングの右端の文字。

階層ファイル・システム (hierarchical file system)

階層構造で編成されたファイルとディレクトリーの集合であり、z/OS UNIX を使用してアクセスできる。

外部 10 進数データ項目 (external decimal data item)

ゾーン 10 進数データ項目 (*zoned decimal data item*) および 国別 10 進数データ項目 (*national decimal data item*) を参照。

外部コード・ページ

ASCII または UTF-8 XML 文書の場合は、現在のランタイム・ロケールが指示するコード・ページ。EBCDIC XML 文書の場合は、次のいずれか。

- EBCDIC_CODEPAGE 環境変数で指定されたコード・ページ
- EBCDIC_CODEPAGE 環境変数が設定されていない場合に現在のランタイム・ロケールとして選択されたデフォルトの EBCDIC コード・ページ。

外部スイッチ (* external switch)

インプリメントする人によって定義され指名されたハードウェアまたはソフトウェア装置であり、2 つの代替状態のいずれかが存在していることを示す。

外部データ (* external data)

プログラムの中で外部データ項目および外部ファイル結合子として記述されるデータ。

外部データ・レコード (* external data record)

実行単位の 1 つ以上のプログラムにおいて記述される論理レコードであり、そのデータ項目は、それらが記述されている任意のプログラムから参照できる。

外部データ項目 (* external data item)

実行単位の 1 つ以上のプログラムにおいて外部レコードの一部として記述されるデータ項目であり、その項目が記述されている任意のプログラムから参照することができる。

外部ファイル結合子 (* external file connector)

実行単位の 1 つ以上のオブジェクト・プログラムにアクセス可能なファイル結合子。

外部浮動小数点データ項目 (external floating-point data item)

表示浮動小数点データ項目 (*display floating-point data item*) および 国別浮動小数点データ項目 (*national floating-point data item*) を参照。

外部プログラム (external program)

最外部プログラム。ネストされていないプログラム。

カウンター (* counter)

他の数字を使ってその数字分だけ増減したり、あるいは 0 または任意の正もしくは負の値に変更またはリセットしたりできるようにした、数または数表現を収めるために使用されるデータ項目。

拡張 (extensions)

COBOL 85 標準で記述されるもの以外で、IBM コンパイラーでサポートされる COBOL 構文とセマンティクス。

拡張西暦年 (expanded year)

4 桁の年だけから構成される日付フィールド。その値には世紀が含まれる (例えば、1998)。ウィンドウ表示西暦年 (*windowed year*) と比較。

拡張日付フィールド (expanded date field)

拡張 (4 桁) 年を含む日付フィールド。日付フィールド (*date field*) および 拡張西暦年 (*expanded year*) も参照。

拡張モード (* extend mode)

ファイルに対する EXTEND 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

型式化オブジェクト参照 (typed object reference)

指定されたクラスまたはそのサブクラスのオブジェクトだけを参照できるデータ名。

カタログ式プロシージャ (cataloged procedure)

プロシージャ・ライブラリー (SYS1.PROCLIB) と呼ばれる区分データ・セットに置かれた一連のジョブ制御ステートメント。カタログ式プロシージャを使用すると、JCL をコーディングする時間を節約して、エラーを減らすことができる。

カプセル化 (encapsulation)

オブジェクト指向プログラミングでは、オブジェクトの固有の詳細を隠すのに使用される技法。オブジェクトは、基礎構造を露出しなくても、データの照会と操作を行うインターフェースを提供する。「情報隠蔽 (*information hiding*)」と同義。

可変位置グループ (* variably located group)

同じレコード内の可変長テーブルに続くグループ項目 (可変長テーブルに從属するわ

けではない)。グループ項目は、英数字グループでも国別グループでも構いません。

可変位置項目 (* **variably located item**)

同じレコード内の可変長テーブルに続くデータ項目 (可変長テーブルに従属するわけではない)。

可変オカレンス・データ項目 (*

variable-occurrence data item)

可変オカレンス・データ項目とは、反復される回数が可変であるテーブル・エレメントを言う。そのような項目は、そのデータ記述記入項目内に OCCURS DEPENDING ON 節を持っているか、またはそのような項目に従属していなければならない。

可変長項目 (**variable-length item**)

OCCURS 節の DEPENDING 句で記述された表を含んだグループ項目。

可変長レコード (* **variable-length record**)

ファイル記述項目またはソート・マージ・ファイル記述記入項目が、文字位置の数が可変であるレコードを許容しているファイルに関連付けられているレコード。

環境節 (* **environment clause**)

ENVIRONMENT DIVISION 記入項目の一部として現れる節。

環境変数 (**environment variable**)

コンピューター環境の一部の局面を定義する多数の変数のいずれかであり、その環境で動作するプログラムからアクセス可能。環境変数は、動作環境に依存するプログラムの動作に影響を与える。

環境名 (**environment-name**)

IBM が指定する名前であり、システム論理装置、プリンターおよびカード穿孔装置の制御文字、報告書コード、またはプログラム・スイッチ、あるいはそれらの組み合わせを識別する。環境名が ENVIRONMENT DIVISION の簡略名と関連付けられている場合は、その簡略名を、置換が有効な任意の形式で置き換えることができる。

関係 (* **relation**)

関係演算子 (*relational operator*) または 比較条件 (*relation condition*) を参照。

関係演算子 (* **relational operator**)

比較条件の構造で使用される、予約語、比

較文字、連続する予約語のグループ、または連続する予約語と比較文字のグループ。使用できる演算子とそれらの意味は次のとおり。

文字	意味
IS GREATER THAN	より大きい
IS >	より大きい
IS NOT GREATER THAN	より大きくない (以下)
IS NOT >	より大きくない (以下)
IS LESS THAN	より小さい
IS <	より小さい
IS NOT LESS THAN	より小さくない (以上)
IS NOT <	より小さくない (以上)
IS EQUAL TO	に等しい
IS =	に等しい
IS NOT EQUAL TO	に等しくない
IS NOT =	に等しくない
IS GREATER THAN OR EQUAL TO	より大きいか等しい (以上)
IS >=	より大きいか等しい (以上)
IS LESS THAN OR EQUAL TO	より小さいか等しい (以下)
IS <=	より小さいか等しい (以下)

関数 ID (* **function-identifier**)

関数を参照する文字ストリングと区切り文字の構文的に正しい組み合わせ。関数で表現されるデータ項目は、関数名と引数 (ある場合) によって一意的に識別される。関数 ID は、参照修飾子を含むことができる。英数字関数を参照する関数 ID は、一定の制限に従いつつ ID が指定できる一般フォーマットの中ならばどこにでも指定できる。整数関数または数字関数を参照する関数 ID は、算術式が指定できる一般フォーマットの中ならばどこにおいても指定できる。

関数ポインター・データ項目 (**function-pointer data item**)

入り口点を指すポインターを保管できるデータ項目。USAGE IS FUNCTION-POINTER 節で定義されるデータ項目に、関数入り口点のアドレスが含まれる。一般的に、C および Java プログラムと通信するために使用される。

関数名 (function-name)

必要な引数を指定した呼び出しによって、関数の値が決定されるメカニズムを指名するワード。

簡略複合比較条件 (* abbreviated combined relation condition)

連続した一連の比較条件において、共通サブジェクトの明示的な省略、または共通サブジェクトと共通関係演算子の明示的な省略によって生じる複合条件。

簡略名 (* mnemonic-name)

ENVIRONMENT DIVISION において、指定されたインプリメントする人の名前に関連したユーザー定義語。

キー (* key)

レコードの位置を識別するデータ項目、またはデータの順序付けを識別するための一連のデータ項目。

キーワード (* keyword)

予約語または関数名で、その語の現れる形式がソース・プログラムの中で使用されるときには必須である。

疑似テキスト (* pseudo-text)

ソース・プログラムまたは COBOL ライブラリーにおいて、疑似テキスト区切り文字によって区切られた一連のテキスト・ワード、コメント行、または区切り文字スペース (疑似テキスト区切り文字を含まない)。

疑似テキスト区切り文字 (* pseudo-text delimiter)

疑似テキストを区切るために使用される 2 つの連続する等号文字 (==)。

記入項目のオブジェクト (* object of entry)

COBOL プログラムの DATA DIVISION 記入項目内の一連のオペランドと予約語であり、その記入項目のサブジェクトの直後に続く。

記入項目のサブジェクト (* subject of entry)

DATA DIVISION の記入項目内において、レベル標識またはレベル番号の直後に現れるオペランドまたは予約語。

機能 (* function)

ステートメントの実行中に参照された時点で決定される値を持つ、一時的なデータ項目。

基本項目 (* elementary item)

それ以上論理的に分割されないものとして記述されるデータ項目。

基本レコード・キー (* prime record key)

索引付きファイルのレコードを固有なものとして識別する内容を持つキー。

共通プログラム (* common program)

別のプログラムに直接的に含まれているにもかかわらず、その別のプログラムに直接的または間接的に含まれている任意のプログラムから呼び出すことができるプログラム。

共用ライブラリー (shared library)

リンカーによって作成され、少なくとも 1 つのサブルーチンを含み、複数のプロセスで使用できるライブラリー。プログラムとサブルーチンは従来どおりリンクされるが、異なるサブルーチンに共通するコードは結合されて 1 つのライブラリー・ファイルに入れられる。このライブラリー・ファイルは実行時にロード可能で、多数のプログラムで共用できる。この共用ライブラリー・ファイルを識別するキーは、各サブルーチンのヘッダーにある。

切り替え状況条件 (switch-status condition)

オンまたはオフに設定可能な UPSI スイッチが、特定の状況に設定されているという命題で、これに関して真理値を判別することができる。

記録モード (recording mode)

ファイル内の論理レコードの形式。記録モードは、F (固定長)、V (可変長)、S (スパン)、または U (不定形式) とすることができる。

キロバイト (KB) (kilobyte (KB))

1 キロバイトは 1024 バイトに相当する。

句 (* phrase)

連続する 1 つ以上の COBOL 文字ストリングを配列したセットで、COBOL プロシージャ・ステートメントまたは COBOL 節の一部を構成する。

空白文字 (white space)

文書にスペースを挿入する文字。空白文字には以下のものがある。

- スペース

- 水平タブ
- 復帰
- 改行
- 次の行

Unicode 標準では上記のように呼ばれる。

区切り文字 (* delimiter)

1 つの文字、または一連の連続する文字であり、文字ストリングの終わりを識別し、その文字ストリングを後続の文字ストリングから区切る。区切り文字は、これを使用して区切られる文字ストリングの一部ではない。

句読文字 (* punctuation character)

以下のセットに属する文字。

文字	意味
,	コンマ
;	セミコロン
:	コロン
.	ピリオド (終止符)
"	引用符
(左括弧
)	右括弧
=	スペース 等号

国別 10 進数データ項目 (national decimal data item)

暗黙的または明示的に USAGE NATIONAL として記述されており、PICTURE の記号 9、S、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。

国別グループ項目 (national group item)

明示的または暗黙的に GROUP-USAGE NATIONAL 節で記述されたグループ項目。国別グループ項目は、INSPECT、STRING、および UNSTRING などの操作で、カテゴリー国別の基本データ項目として定義されているかのように処理されます。英数字グループ項目内で USAGE NATIONAL データ項目を定義するのは対照的に、この処理により、国別文字の埋め込みおよび切り捨てが確実に正しく行われます。グループ内の基本項目を処理する必要のある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、および INITIALIZE など) の場合、国別グループはグループ・セマンティクスを使用して処理されます。

国別データ項目 (national data item)

カテゴリー国別、国別編集、または USAGENATIONAL の数字編集のデータ項目。

国別浮動小数点データ項目 (national floating-point data item)

暗黙的または明示的に USAGE NATIONAL として記述されており、浮動小数点データ項目を記述する PICTURE 文字ストリングを持っている、外部浮動小数点データ項目。

国別編集データ項目 (national-edited data item)

少なくとも 1 つの N のインスタンスおよび単純挿入記号 B、0、または / の少なくとも 1 つを含んでいる PICTURE 文字ストリングで記述されている、データ項目。国別編集データ項目は USAGE NATIONAL を持ちます。

国別文字 (national character)

(1) 国別リテラルまたは USAGE NATIONAL の UTF-16 文字。(2) UTF-16 で表される任意の文字。

国別文字位置 (national character position)

文字位置 (*character position*) を参照。

組み込み関数 (built-in function)

組み込み関数 (*intrinsic function*) を参照。

組み込み関数 (intrinsic function)

よく使用される算術関数のような事前定義関数で、組み込み関数参照によって呼び出される。

クライアント (client)

オブジェクト指向プログラミングにおいて、クラス内の 1 つ以上のメソッドからサービスを要求するプログラムまたはメソッド。

クラス (* class)

ゼロ、1 つ、または複数のオブジェクトの共通の動作およびインプリメンテーションを定義するエンティティ。同じ具体化を共用するオブジェクトは、同じクラスのオブジェクトとみなされる。クラスは階層として定義でき、あるクラスを別のクラスから継承することができる。

クラス・オブジェクト (class object)

クラスを表す実行時オブジェクト。

クラス階層 (class hierarchy)

オブジェクト・クラス間の関係を示すツリーのような構造。最上部に 1 つのクラスが置かれ、その下に 1 つ以上のクラスの層が置かれる。「継承階層 (*inheritance hierarchy*)」と同義。

クラス識別記入項目 (* class identification entry)

IDENTIFICATION DIVISION の CLASS-ID 段落内の記入項目であり、クラス名を指定する節と、選択した属性をクラス定義に割り当てる節を含む。

クラス条件 (* class condition)

項目の内容がすべて英字であるか、すべて数字であるか、すべて DBCS であるか、すべて漢字であるか、あるいはクラス名の定義においてリストされた文字だけで構成されるかという命題で、それに関して真の値を判別することができる。

クラス定義 (* class definition)

クラスを定義する COBOL ソース単位。

クラス名 (オブジェクト指向) (class-name (object-oriented))

オブジェクト指向 COBOL クラス定義の名前。

クラス名 (データの) (* class-name (of data))

ENVIRONMENT DIVISION の SPECIAL-NAMES 段落で定義されるユーザー定義語であり、真理値を定義できる命題に名前を割り当てる。データ項目の内容は、クラス名の定義にリストされている文字だけで構成される。

グループ区切り文字 (grouping separator)

読みやすさのために数値を何桁かまとめて区切るのに使用される文字。デフォルトはコンマである。

グループ項目 (group item)

- (1) 複数の従属データ項目で構成されるデータ項目。英数字グループ項目 (*alphanumeric group item*) および 国別グループ項目 (*national group item*) を参照。
- (2) 国別グループまたは英数字グループとして明示的に (またはコンテキストで) 限定されていない場合、この用語は一般のグループを指します。

グローバル名 (* global name)

1 つのプログラムにおいてのみ宣言されるが、そのプログラム、またはそのプログラム内に含まれている任意のプログラムから参照できる名前。条件名、データ名、ファイル名、レコード名、報告書名、およびいくつかの特殊レジスターが、グローバル名となり得る。

ケース構造 (case structure)

結果として生じた多数のアクションの中から選択を行うために、一連の条件をテストするプログラム処理ロジック。

継承 (inheritance)

クラスのインプリメンテーションを、別のクラスを基にして使用するメカニズム。定義により、継承するクラスは継承されるクラスに準拠する。COBOL for AIX は 多重継承 をサポートしない。サブクラスは、必ず 1 つの即時スーパークラスを有する。

継承階層 (inheritance hierarchy)

クラス階層 (*class hierarchy*) を参照。

形象定数 (* figurative constant)

ある予約語を使用して参照されるコンパイラー生成の値。

桁位置 (* digit position)

1 つの桁を保管するために必要な物理ストレージの大きさ。この大きさは、データ項目を定義するデータ記述記入項目に指定された用途によって異なる。

結果 ID (* resultant identifier)

算術演算の結果が収められるユーザー定義のデータ項目。

現行レコード (* current record)

ファイル処理では、ファイルに関連したレコード域に使用できるレコード。

言語間通信 (ILC) (interlanguage communication (ILC))

異なるプログラム言語で書かれた複数のルーチンが通信できること。ILC サポートにより、各種言語で書かれたコンポーネント・ルーチンからアプリケーションを簡単に構築することができる。

言語名 (* language-name)

特定のプログラミング言語を指定するシステム名。

コード・ページ (code page)

すべてのコード・ポイントに図形文字および制御機能の意味を割り当てるもの。例えば、あるコード・ページでは、8 ビット・コードに対して 256 コード・ポイントに文字と意味を割り当て、別のコード・ページでは、7 ビット・コードに対して 128 コード・ポイントに文字と意味を割り当てることができる。ワークステーション上の英語の IBM コード・ページは IBM-1252 で、ホストは IBM-1047 である。

コード・ポイント (code point)

コード化文字セット (コード・ページ) に定義する固有のビット・パターン。コード・ポイントには、グラフィック・シンボルおよび制御文字が割り当てられる。

コード化文字セット (coded character set)

文字セットを設定し、その文字セットの文字とコード化表現との間の関係を設定する明確な規則の集まり。コード化文字セットの例として、ASCII もしくは EBCDIC コード・ページで、または Unicode 対応の UTF-16 エンコード・スキームで表す文字セットがある。

コード化文字セット ID (coded character set identifier (CCSID))

特定のコード・ページを識別する 1 から 65,535 までの IBM 定義番号。

高位終了 (* high-order end)

文字ストリングの左端の文字。

降順キー (* descending key)

値に基づくキーであり、そのデータが、キーの最高値からキーの最低値まで、データ項目比較規則に従って順序付けられている。

構造化プログラミング (structured programming)

コンピューター・プログラムを編成してコーディングするための技法であり、この技法では、プログラムはセグメントの階層で構成され、それぞれのセグメントには 1 つの入り口点と 1 つの出口点がある。制

御は、構造の下方へと渡され、階層内のより上位レベルへの無条件分岐は行われな

構文 (syntax)

(1) 意味や解釈および使用の方法に依存しない、文字同士または文字のグループ同士の間の関係。(2) 言語における表現の構造。(3) 言語構造を支配する規則。(4) 記号相互の関係。(5) ステートメントの構築にかかわる規則。

項目 (* entry)

分離文字ピリオドで終了させられる連続する節の記述セットであり、COBOL プログラムの IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、または DATA DIVISION に書き込まれる。

互換性のある日付フィールド (compatible date field)

互換 という用語の意味は、日付フィールドに適用される場合、それが COBOL のどの部で使用されるかによって異なる。

- DATA DIVISION: 2 つの日付フィールドが同一の USAGE を持ち、以下の条件の少なくとも 1 つを満たしている場合、それらの日付フィールドは互換性があります。
 - 同じ日付形式を持つ。
 - とともにウィンドウ表示日付フィールドであり、一方がウィンドウ表示西暦年 DATE FORMAT YY だけで構成される。
 - とともに拡張日付フィールドであり、一方が拡張西暦年 DATE FORMAT YYYY だけで構成される。
 - 一方が DATE FORMAT YYXXXX で、他方が YYXX の形式である。
 - 一方が DATE FORMAT YYYYXXXX で、他方が YYYYXX の形式である。

ウィンドウ表示日付フィールドは、拡張日付グループであるデータ項目に從属することができる。2 つの日付フィールドに互換性があると言われるのは、從属日付フィールドが USAGE DISPLAY を持ち、グループ拡張日付フィールドの開始より 2 バイト後で始まっており、2 つ

のフィールドが以下の少なくとも 1 つの条件を満たしている場合である。

- 従属日付フィールドの DATE FORMAT パターンが、グループ日付フィールドの DATE FORMAT パターンと同じ数の X を持つ。
- 従属日付フィールドが DATE FORMAT YY を持つ。
- グループ日付フィールドが DATE FORMAT YYYYXXXX を持ち、従属日付フィールドが DATE FORMAT YYXX を持つ。
- PROCEDURE DIVISION: 2 つの日付フィールドが、ウィンドウ表示または拡張できる年部分を除いて、同じ日付形式を持っている場合、それらのフィールドは互換性があります。例えば、DATE FORMAT YYXXX という形式のウィンドウ表示日付フィールドは、以下のものと互換性がある。
 - DATE FORMAT YYXXX という形式の別のウィンドウ表示日付フィールド。
 - DATE FORMAT YYYYXXX という形式の拡張日付フィールド。

固定小数点項目 (**fixed-point item**)

PICTURE 節で定義される数値データ項目であり、オプションの符号の位置、その中に含まれる桁数、およびオプションの小数点の位置を指定するもの。2 進数、パック 10 進数、または外部 10 進数のいずれかのフォーマットをとることができる。

固定長レコード (*** fixed-length record**)

ファイル記述項目またはソート・マージ記述記入項目が、すべてのレコードのバイトの個数が同じであるように要求しているファイルに関連付けられたレコード。

固定ファイル属性 (*** fixed file attributes**)

ファイルに関する情報であり、ファイルの作成時に設定され、それ以降はファイルが存在する限り変更できない。これらの属性には、ファイル (順次、相対、または索引付き) の編成、基本レコード・キー、代替レコード・キー、コード・セット、最小および最大レコード・サイズ、レコード・タイプ (固定または可変)、索引付きファイル

のキーの照合シーケンス、ブロック化因数、埋め込み文字、およびレコード区切り文字がある。

コピーブック (**copybook**)

一連のコードが含まれたファイルまたはライブラリー・メンバーであり、コンパイル時に COPY ステートメントを使用してソース・プログラムに組み込まれる。ファイルはユーザーが作成する場合、COBOL によって提供される場合、または他の製品によって供給される場合とがある。「コピー・ファイル (*copy file*)」と同義。

コメント記入項目 (*** comment-entry**)

IDENTIFICATION DIVISION 内の記入項目であり、コンピューターの文字セットから任意の文字を組み合わせることができる。

コメント行 (*** comment line**)

行の標識区域ではアスタリスク (*), およびその行の区域 A および B ではコンピューターの文字セットの任意の文字で表されるソース・プログラム行。コメント行は、文書化にのみ役立つ。行の標識区域では斜線 (/), そしてその行の区域 A および B ではコンピューター文字セットの任意の文字で表される特殊形式のコメント行があると、コメントの印刷前に改ページが行われる。

固有照合シーケンス (*** native collating sequence**)

OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した照合シーケンス。

固有文字セット (*** native character set**)

OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した文字セット。

コンパイラー (**compiler**)

高水準言語で記述されたソース・コードをマシン言語のオブジェクト・コードに変換するプログラム。

コンパイラー指示 (**compiler directive**)

コンパイル時にコンパイラーに特定の処置を行わせる指示。COBOL for AIX には、CALLINTERFACE という 1 つのコンパイラー指示がある。プログラム内に CALLINTERFACE 指示をコーディングすれ

ば、個々の CALL ステートメントに固有のインターフェース規則を使用できる。

コンパイラ指示ステートメント (**compiler-directing statement**)

コンパイル時にコンパイラに特定の処置を行わせるステートメント。標準コンパイラ指示ステートメントには、COPY、REPLACE、および USE がある。

コンパイル (* **compile**)

(1) 高水準言語で表現されたプログラムを、中間言語、アセンブリ言語、またはコンピューター言語で表現されたプログラムに変換すること。(2) あるプログラミング言語で書かれたコンピューター・プログラムから、プログラムの全体的なロジック構造を利用することによって、または 1 つの記号ステートメントから複数のコンピューター命令を作り出すことによって、またはアセンブラの機能のようにこれら両方を使用することによって、マシン言語プログラムを生成すること。

コンパイル時間 (* **compile time**)

COBOL コンパイラによって、COBOL ソース・コードが COBOL オブジェクト・プログラムに変換される時間。

コンパイル用コンピューター記入項目 (* **source computer entry**)

ENVIRONMENT DIVISION の SOURCE-COMPUTER 段落内の記入項目であり、ソース・プログラムがコンパイルされるコンピューター環境を記述する節が入っている。

コンピューター名 (* **computer-name**)

プログラムがコンパイルまたは実行されるコンピューターを識別するシステム名。

コンポーネント (**component**)

(1) 関連ファイルからなる機能グループ化。(2) オブジェクト指向プログラミングでは、特定の機能を実行し、他のコンポーネントやアプリケーションと連携するように設計されている、再使用可能なオブジェクトまたはプログラム。JavaBeans™は、Sun Microsystems, Inc. のコンポーネント作成用アーキテクチャーである。

再帰 それ自体を呼び出すプログラム、または、それ自体で呼び出したプログラムのいずれかによって直接あるいは間接に呼び出されるプログラム。

再帰可能 (**recursively capable**)

PROGRAM-ID ステートメントで RECURSIVE 属性が指定されていれば、プログラムは再帰可能である (再帰的に呼び出すことができる)。

最後に使われた状態 (**last-used state**)

内部値がプログラム終了時と同じままで、初期値にリセットされない、プログラムの状態を言う。

再入可能 (**reentrant**)

プログラムまたはルーチンの属性。この属性によって、ロード・モジュールの 1 つのコピーを複数のユーザーが共用できる。

索引付きデータ名 (**indexed data-name**)

データ名とそれに続く 1 つ以上の (括弧で囲まれた) 索引名で構成される ID。

索引付きファイル (* **indexed file**)

索引編成のファイル。

索引名 (* **index-name**)

特定のテーブルに関係付けられた指標を指名するユーザー定義語。

サブクラス (* **subclass**)

別のクラスから継承するクラス。継承関係にある 2 つのクラスをまとめて考える場合、継承する側、つまり継承先のクラスをサブクラスといい、継承される側、つまり継承元のクラスをスーパークラスという。

サブプログラム (* **subprogram**)

呼び出し先プログラム (*called program*) を参照。

サロゲート・ペア (**surrogate pair**)

UTF-16 形式のユニコードで、共に 1 つのユニコード図形文字を表すエンコード方式ペアの単位。ペアの最初の単位は上位サロゲートと呼ばれ、第 2 の単位は下位サロゲートと呼ばれる。上位サロゲートのコード値の範囲は、X'D800' から X'DBFF' である。下位サロゲートのコード値の範囲は、X'DC00' から X'DFFF' である。サロゲート・ペアは、Unicode 16

ビット・コード文字セットで表現できる 65,536 文字より多くの文字を表現できる。

算術演算 (* arithmetic operation)

ある算術ステートメントが実行されることにより、またはある算術式が計算されることにより生じるプロセスで、そこで与えられている引数に対して数学的に正しい解が求められる。

算術演算子 (* arithmetic operator)

次に示す集合に属する 1 文字、または 2 文字で構成された固定した組み合わせ。

文字	意味
+	加算
-	減算
*	乗算
/	除算
**	指数

算術式 (* arithmetic expression)

数字基本項目の ID、数値リテラル、そのような ID とリテラルを算術演算子で区切ったもの、2 つの算術式を算術演算子で区切ったもの、または算術式を括弧で囲んだもの。

算術ステートメント (* arithmetic statement)

算術演算を実行させるステートメント。算術ステートメントには、ADD、COMPUTE、DIVIDE、MULTIPLY、および SUBTRACT の各ステートメントがある。

参照キー (* key of reference)

索引付きファイルの中のレコードをアクセスするために現在使用されている基本キーまたは代替キー。

参照形式 (* reference format)

COBOL ソース・プログラムを記述するに際して標準的な方式を提供する形式。

参照修飾子 (* reference-modifier)

固有のデータ項目を定義する文字ストリングと区切り文字の構文的に正しい組み合わせ。区切り用の左括弧分離符号、左端の文字位置、分離符号のコロン、長さ (オプション)、および区切り用の右括弧分離符号を含む。

参照変更 (reference modification)

新規のカテゴリ-英数字、カテゴリ-

DBCS、またはカテゴリ-国別のデータ項目を定義する方法であり、USAGE DISPLAY、DISPLAY-1、または NATIONAL データ項目の左端文字および左端文字位置を基準にした長さを指定して定義する方法です。

式 (* expression)

算術式または条件式。

シグニチャー (signature)

- (1) ある操作とそのパラメーターの名前。
- (2) あるメソッドの名前とその仮パラメーターの数と型。

指数 (exponent)

別の数 (底) をべき乗する指数を示す数。正の指数は乗算を示し、負の指数は除算を示し、小数の指数は数量の根を示す。COBOL では、指数式は記号 ** の後に指数を付けて表す。

システム名 (* system-name)

オペレーティング環境と連絡し合うために使用される COBOL ワード。

事前初期設定 (preinitialization)

プログラム (特に非 COBOL プログラム) からの複数の呼び出しの準備としての COBOL ランタイム環境の初期設定。この環境は、明示的に終了されるまで終了されない。

実行時 (* run time)

オブジェクト・プログラムが実行される時。「オブジェクト時 (object time)」と同義。

実行時 (execution time)

実行時 (run time) を参照。

実行時環境 (execution-time environment)

ランタイム環境 (runtime environment) を参照。

実行単位 (* run unit)

1 つの独立型オブジェクト・プログラム、あるいは COBOL の CALL または INVOKE ステートメントによって相互作用し、実行時に 1 つのエンティティーとして機能する複数のオブジェクト・プログラム。

実際の小数点 (* actual decimal point)

10 進小数点文字のピリオド (.) またはコ

ンマ (,) を使用して、または、データ項目内の 10 進小数点のコンマ (,.)。

実際の文書エンコード (actual document encoding)

XML 文書のエンコード・カテゴリで、以下のいずれかとなる。XML パーサーは文書の最初の数バイトを調べて判別する。

- ASCII
- EBCDIC
- UTF-8
- UTF-16 (ビッグ・エンディアンまたはリトル・エンディアンのいずれか)
- これ以外のサポートされないエンコード
- 認識不能なエンコード

失敗した実行 (* unsuccessful execution)

ステートメントの実行が試みられたが、そのステートメントに指定された操作すべてを実行できなかったこと。あるステートメントの実行不成功は、そのステートメントによって参照されるデータには影響を及ぼさないが、状況表示には影響を与える可能性がある。

指定変更 (override)

サブクラスのインスタンス・メソッド (親クラスから継承された) を再定義すること。

指標 (* index)

その内容が、テーブル内の特定エレメントの識別を表す、コンピューターのストレージ域またはレジスター。

指標付き編成 (* indexed organization)

各レコードが、そのレコード内の 1 つ以上のキーの値で識別される、永続論理ファイル構造。

指標付け (indexing)

指標名を使用しての添え字付け と同義。

指標データ項目 (* index data item)

索引名および関連する値をインストール先指定の形式で保管できるデータ項目。

修飾子 (* qualifier)

(1) レベル標識と関連付けられるデータ名または名前であり、参照の際に、別のデータ名 (修飾子に従属する項目の名前) と一緒に、または条件名と一緒に使用される。

(2) セクション名。そのセクションの中で指定されている段落名と共に参照する際に使用される。(3) ライブラリー名。そのライブラリーと関連付けられたテキスト名と共に参照する際に使用される。

修飾データ名 (* qualified data-name)

データ名と、その後に連結語の OF または IN とデータ名修飾子を続けたものが 1 つ以上のセットで続いて構成される ID。

終了クラス・マーカ (end class marker)

語の組み合わせに分離文字ピリオドが続いたもので、COBOL クラス定義の終わりを示す。クラス終了マーカは次のとおり。

END CLASS *class-name*.

終了メソッド・マーカ (end method marker)

語の組み合わせに分離文字ピリオドが続いたもので、COBOL メソッド定義の終わりを示す。メソッド終了マーカは次のとおり。

END METHOD *method-name*.

出力ファイル (* output file)

出力モードまたは拡張モードのいずれかでオープンされるファイル。

出力プロシージャ (* output procedure)

SORT ステートメントの実行中にソート機能が完了した後で制御が渡されるステートメントの集合、または MERGE ステートメントの実行中に、要求があればマージ機能がマージ済みの順序になっているレコードのうち次のレコードを選択できるようになった後で制御が渡されるステートメントの集合。

出力モード (* output mode)

OUTPUT または EXTEND 句の指定のある OPEN ステートメントが実行されてから、REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。

順次アクセス (* sequential access)

ファイル内のレコードの順序によって規定されている、論理レコードの連続した前後関係順に、論理レコードをファイルから取り出したり、ファイルに書き込んだりするアクセス・モード。

順次ファイル (* sequential file)

順次編成のファイル。

順次編成 (* sequential organization)

レコードがファイルに書き込まれるときに確定されたレコードの前後関係によって識別されるような永続的な論理ファイル構造。

順序構造 (sequence structure)

一連のステートメントが、順序どおりに実行されるプログラムの処理ロジック。

条件 (式) (condition (expression))

真理値を判別できる、実行時のデータの状況。本書で使用される場合、または一般形式の「条件」(*condition-1*, *condition-2*, . . .)に関連して使用される場合は、必要に応じて括弧で囲んだ単純条件からなる条件式、または複合条件 (単純条件、論理演算子、および括弧を構文的に正しく組み合わせたもの) からなる条件式。この条件式に対して真理値を判別できる。単純条件 (*simple condition*)、複合条件 (*complex condition*)、単純否定条件 (*negated simple condition*)、複合条件 (*combined condition*)、および複合否定条件 (*negated combined condition*) も参照。

条件 (例外) (condition (exception))

アプリケーションの通常のプログラミングされたフローを変えるもの。条件は、ハードウェアまたはオペレーティング・システムによって検出され、その結果、割り込みが起こる。このほかにも、条件は言語特定の生成コードまたは言語ライブラリー・コードによっても検出できる。

条件句 (* conditional phrase)

ある条件ステートメントが実行された結果得られる条件の真理値の判別に基づいてとられるべき処置を指定する句。

条件式 (* conditional expression)

EVALUATE、IF、PERFORM、または SEARCH ステートメントの中で指定される単純条件または複合条件。単純条件 (*simple condition*) および 複合条件 (*complex condition*) も参照。

条件ステートメント (* conditional statement)

条件の真理値を判別することと、オブジェ

クト・プログラムの次の処理がこの真理値によって決まることを指定するステートメント。

条件変数 (* conditional variable)

1 つ以上の値を持つデータ項目であり、これらの値が、そのデータ項目に割り当てられた条件名を持つ。

条件名 (* condition-name)

条件変数が想定できる値のサブセットに名前を割り当てるユーザー定義語。または、インプリメントする人が定義したスイッチまたは装置の状況に割り当てられるユーザー定義語。

条件名条件 (* condition-name condition)

真理値を判別できる命題で、かつ、条件変数の値が、その条件変数と関連する条件名に属する一連の値のメンバーである命題。

照合シーケンス (* collating sequence)

コンピューターに受け入れられる文字がソート、マージ、比較を行うため、また索引付きファイルを順次処理するために順序付けられているシーケンス。

昇順キー (* ascending key)

データ項目を比較する際の規則に一致するように、最低のキー値から始めて最高のキー値へとデータを順序付けている値に即したキー。

初期状態 (* initial state)

実行単位で最初に呼び出されるときのプログラムの状態。

初期設定プログラム (* initial program)

プログラムが実行単位で呼び出されるたびに初期状態に設定されるプログラム。

シンボリック文字 (* symbolic-character)

ユーザー定義の形象定数を指定するユーザー定義語。

真理値 (* truth value)

2 つの値 (真または偽) のどちらか一方によって、条件評価の結果を表したものの。

スーパークラス (* superclass)

別のクラスによって継承されるクラス。サブクラス (*subclass*) も参照。

数字 (* numeric character)

次のような数字に属する文字。0、1、2、3、4、5、6、7、8、9。

数字 (digit)

0 から 9 までの任意の数字。COBOL では、この用語を用いて他の記号を参照することはない。

数字関数 (* numeric function)

クラスとカテゴリーは数字だが、考えられる評価のいくつかにおいて整数関数の要件を満たさないような関数。

数字編集データ項目 (numeric-edited data item)

印刷出力の際に使用するのに適したフォーマットの数値データを含むデータ項目。データ項目は、外部 10 進数字の 0 から 9 の数字、小数点、コンマ、通貨符号、符号制御文字、その他の編集記号から構成される。数字編集項目は、USAGE DISPLAY または USAGE NATIONAL のいずれかで表すことができる。

数値データ項目 (numeric data item)

(1) 記述により内容が数字 0 から 9 より選ばれた文字で表される値に制限されるデータ項目。符号付きである場合、この項目は +、-、または他の表記の演算符号も含むことができます。(2) カテゴリー数値、内部浮動小数点、または外部浮動小数点のデータ項目。数値データ項目は、USAGE DISPLAY、NATIONAL、PACKED-DECIMAL、BINARY、COMP、COMP-1、COMP-2、COMP-3、COMP-4、または COMP-5 を持つことができます。

数値リテラル (* numeric literal)

1 つ以上の数字から構成されるリテラルで、小数点または代数符号あるいはその両方を含むことができる。小数点は右端の文字であってはならない。代数符号がある場合には、それが左端の文字でなければならない。

ステートメント (* statement)

COBOL ソース・プログラムに書かれる、動詞を冒頭に置いた、ワード、リテラル、および区切り記号の構文的に正しい組み合わせ。

スレッド (thread)

プロセスの制御下にあるコンピューター命

令のストリーム (プロセス内のアプリケーションによって開始される)。

世紀ウィンドウ (century window)

2 桁年号が固有に決まる 100 年間のこと。COBOL プログラマーが使用できる世紀ウィンドウには、いくつかのタイプがある。

- ウィンドウ表示日付フィールドについては、YEARWINDOW コンパイラー・オプションを使用する。
- ウィンドウ操作組み込み関数 DATE-TO-YYYYMMDD、DAY-TO-YYYYDDD、および YEAR-TO-YYYY については、引数-2 (argument-2) によって世紀ウィンドウを指定する。

整数 (* integer)

(1) 小数点の右側に桁位置がない数値リテラル。(2) DATA DIVISION に定義される数値データ項目であり、小数点の右側に桁位置を含まないもの。(3) 関数の起こりうるすべての評価の戻り値で、小数点の右側の桁がすべてゼロであることが定義されている数字関数。

整数関数 (integer function)

カテゴリーが数字であり、小数点の右側の桁位置が定義に入っていない関数。

セクション (* section)

ゼロ、1 つ、または複数の段落またはエンティティ (セクション本体と呼ばれる) と、その最初のもの前にセクション・ヘッダーが付いているもの。各セクションは、セクション・ヘッダーとそれに関連するセクション本体から構成される。

セクション・ヘッダー (* section header)

後ろに分離文字ピリオドが付いたワードの組み合わせであり、ENVIRONMENT、DATA、またはPROCEDURE の各部において、セクションの始まりを示すもの。ENVIRONMENT DIVISION および DATA DIVISION では、セクション・ヘッダーは、予約語の後に分離文字ピリオドを続けたものから構成される。ENVIRONMENT DIVISION で許可されているセクション・ヘッダーは次のとおり。CONFIGURATION SECTION.INPUT-OUTPUT SECTION.

DATA DIVISION で許可されているセクション・ヘッダーは次のとおり。

FILE SECTION.WORKING-STORAGE
SECTION.LOCAL-STORAGE
SECTION.LINKAGE SECTION.

PROCEDURE DIVISION では、セクション・ヘッダーは、セクション名、その後続く予約語SECTION、およびその後の分離文字ピリオドから構成される。

セクション名 (* section-name)

PROCEDURE DIVISION 中にあるセクションに名前を付けるユーザー定義語。

節 (* clause)

記入項目の属性を指定するという目的で順番に並べられた連続する COBOL 文字ストリング。

宣言部分 (* declaratives)

PROCEDURE DIVISION の先頭に書き込まれた 1 つ以上の特殊目的セクションの集合であり、その先頭にはキーワード DECLARATIVE が付き、その最後にはキーワード END DECLARATIVES が続いている。宣言部分は、セクション・ヘッダー、USE コンパイラー指示文、および 0 個、1 個、または複数個の関連する段落で構成される。

宣言文 (* declarative sentence)

区切り記号のピリオドによって終了する 1 つの USE ステートメントから構成されるコンパイラー指示文。

選択構造 (selection structure)

条件が真であるか偽であるかに応じて、ある一連のステートメントか、または別の一連のステートメントが実行されるというプログラムの処理ロジック。

ソース・プログラム (source program)

ソース・プログラムは、他の形式や記号を使用して表現することができるが、本書では、構文的に正しい COBOL ステートメントの集合を常に指している。COBOL ソース・プログラムは、IDENTIFICATION DIVISIONまたは COPY ステートメントで開始され、指定された場合はプログラム終了マーカーで終了するか、または追加のソース・プログラム行なしで終了する。

ソース項目 (* source item)

SOURCE 節によって指定される ID で、印刷可能な項目の値を提供する。

ソース単位 (source unit)

COBOL ソース・コードの 1 単位で、個別にコンパイルできる。プログラムまたはクラス定義。コンパイル単位 とも呼ばれる。

ソート・ファイル (* sort file)

SORT ステートメントによってソートされるレコードの集まり。ソート・ファイルは、ソート機能によってのみ作成され使用される。

ソート・マージ・ファイル記述記入項目 (* sort-merge file description entry)

DATA DIVISION の FILE SECTION 中にある記入項目。レベル標識 SD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

ゾーン 10 進数データ項目 (zoned decimal data item)

暗黙的または明示的に USAGE DISPLAY として記述されており、PICTURE の記号 9、S、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。ゾーン 10 進数データ項目の内容は、文字 0 から 9 で表され、必要に応じて符号が付きます。PICTURE ストリングが符号を指定しており、SIGN IS SEPARATE 節が指定されている場合、符号は文字 + または - として表されます。SIGN IS SEPARATE が指定されていない場合、符号は、符号位置の最初の 4 ビットをオーバーレイする 1 つの 16 進数字です (先行または末尾)。

相互参照リスト (cross-reference listing)

コンパイラー・リストの一部であり、プログラム内においてファイル、フィールド、および標識が定義、参照、および変更される場所に関する情報が入る。

相対キー (* relative key)

相対ファイルの中の論理レコードを識別するための内容を持つキー。

相対ファイル (* relative file)

相対編成のファイル。

相対編成 (* relative organization)

各レコードが、レコードのファイル内にお

ける論理的順序位置を指定する 0 より大きい整数値によって、固有なものとして識別される永続的な論理ファイル構造。

相対レコード番号 (* relative record number)

相対編成ファイル内でのレコードの序数。この番号は、整数の数値リテラルとして扱われる。

想定小数点 (* assumed decimal point)

データ項目の中に実際には小数点のための文字が入っていない小数点位置。想定小数点には、論理的な意味があり、物理的には表現されない。

添え字 (* subscript)

整数、(オプションで演算子 + または - 付きの整数が後ろにある) データ名、あるいは (オプションで演算子 + または - 付きの整数が後ろにある) 索引名のいずれかによって表されるオカレンス番号。これによりテーブル内の特定のエレメントを識別する。可変数の引数を認める関数では、添え字付き ID を関数引数として使用する場合は、添え字に ALL を使用することができる。

添え字付きデータ名 (* subscripted data-name)

データ名とその後の括弧で囲まれた 1 つ以上の添え字から構成される ID。

タ

代替レコード・キー (* alternate record key)

基本レコード・キー以外のキーであり、その内容が索引付きファイル内のレコードを識別する。

大容量記憶 (* mass storage)

データを順次と非順次の 2 つの方法で編成して保管しておくことができるストレージ・メディア。

大容量記憶装置 (* mass storage device)

磁気ディスクなど、大きな記憶容量を持つ装置。

大容量記憶ファイル (* mass storage file)

大容量記憶メディアに格納されたレコードの集合。

多重定義 (overload)

同じクラスで使用可能な別のメソッドと同一の名前を使い (ただし、異なるシグニチャー

ャーを使用して)、メソッドを定義すること。シグニチャー (*signature*) も参照。

単項演算子 (* unary operator)

正符号 (+) または負符号 (-)。算術式の変数や算術式の左括弧の前に置き、それぞれ +1 または -1 を式に乗算する。

単純条件 (* simple condition)

以下のセットから選択される任意の単一条件。

- 比較条件
- クラス条件
- 条件名条件
- 切り替え状況条件
- 符号条件

条件 (*condition*) および 単純否定条件 (*negated simple condition*) も参照。

単純否定条件 (* negated simple condition)

論理演算子 NOT とその直後に単純条件を続けたもの。条件 (*condition*) および 単純条件 (*simple condition*) も参照。

段落 (* paragraph)

PROCEDURE DIVISION では、段落名の後に分離文字ピリオドが続き、その後に 0 個以上の文が続く。IDENTIFICATION DIVISION および ENVIRONMENT DIVISION では、段落ヘッダーの後に 0 個以上の記入項目が続く。

段落ヘッダー (* paragraph header)

予約語の後に分離文字ピリオドが付いたもので、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION において段落の始まりを示すもの。IDENTIFICATION DIVISION で許可されている段落ヘッダーは次のとおり。

PROGRAM-ID. (Program IDENTIFICATION DIVISION)CLASS-ID. (Class IDENTIFICATION DIVISION)METHOD-ID. (Method IDENTIFICATION DIVISION)AUTHOR.INSTALLATION.DATE-WRITTEN.DATE-COMPILED.SECURITY.

ENVIRONMENT DIVISION で許可されている段落ヘッダーは次のとおり。

SOURCE-COMPUTER.OBJECT-COMPUTER.SPECIAL-NAMES.REPOSITORY. (Program or Class CONFIGURATION SECTION)FILE-CONTROL.I-O-CONTROL.

段落名 (* paragraph-name)

PROCEDURE DIVISION 中の段落を識別し開始するユーザー定義語。

チェックポイント (checkpoint)

ジョブ・ステップを後で再始動することができるように、ジョブとシステムの状態に関する情報を記録しておくことができるポイント。

置換文字 (substitution character)

ソース・コード・ページからターゲット・コード・ページへの変換の際に、ターゲット・コード・ページで定義されていない文字を表すのに使用される文字。

逐次探索 (serial search)

最初のメンバーから始めて最後のメンバーで終わるように、ある集合のメンバーが連続的に検査される探査方法。

中間結果 (intermediate result)

連続して行われる算術演算の結果を収める中間フィールド。

直接アクセス (* direct access)

プロセスが、以前にアクセスされたデータへの参照ではなく、そのデータの位置のみ依存する方法で、ストレージ・デバイスからデータを入力したり、ストレージ・デバイスにデータを入力したりする機能。

通貨記号 (currency symbol)

数字編集項目内の通貨記号値の部分を示すために、PICTURE 節で使用される文字。通貨記号は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 (\$) がデフォルトの通貨記号値および通貨記号として使用される。通貨記号と通貨符号値は複数定義可能。通貨記号値 (currency sign value) も参照。

通貨記号値 (currency-sign value)

数字編集項目に保管される通貨単位を識別する文字ストリング。典型的な例としては、\$, USD, EUR などがある。通貨記号値は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION

の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 (\$) がデフォルトの通貨記号値として使用される。通貨記号 (currency symbol) も参照。

次の実行可能なステートメント (* next executable statement)

現在のステートメントの実行完了後に制御が移される次のステートメント。

次の実行可能な文 (* next executable sentence)

現在のステートメントの実行完了後に制御が移される次の文。

次のレコード (* next record)

ファイルの現在のレコードに論理的に続くレコード。

データ記述記入項目 (* data description entry)

COBOL プログラムの DATA DIVISION 内の記入項目であり、レベル番号の後に必要に応じてデータ名が続き、その後に必要に応じて一連のデータ節で構成されるもの。

データ項目 (* data item)

COBOL プログラムによってまたは関数演算の規則によって定義されるデータ単位 (リテラルを除く)。

データ節 (* data clause)

COBOL プログラムの DATA DIVISION のデータ記述記入項目に現れる節で、データ項目の特定の属性を記述する情報を提供する。

データ名 (* data-name)

データ記述記入項目で記述されたデータ項目に名前を割り当てるユーザー定義語。一般形式で使用された場合、データ名は、その形式の規則で特に許可されていない限り、参照変更、添え字付け、または修飾してはならないワードを表す。

テーブル (* table)

DATA DIVISION の中で OCCURS 節によって定義される、論理的に連続するデータ項目の集合。

テーブル・エレメント (* table element)

テーブルを構成する反復項目の集合に属するデータ項目。

停止点 (breakpoint)

通常は命令によって指定されるコンピューター・プログラムの場所であり、プログラムの実行は外部からの介入またはモニター・プログラムによって割り込まれる場合がある。

テキスト・ワード (* text word)

以下のいずれかの文字から成る COBOL ライブラリー、ソース・プログラム、または疑似テキスト内のマージン A およびマージン R の間の、1 文字または連続した文字のシーケンス。

- スペース以外の区切り記号、疑似テキスト区切り文字、英数字リテラルの開始と終了の区切り文字。ライブラリー、ソース・プログラム、または疑似テキスト内のコンテキストに関係なく、右括弧文字と左括弧文字は常にテキスト・ワードと見なされる。
- 英数字リテラルの場合には、リテラルを囲む左引用符と右引用符を含むリテラル。
- コメント行および区切り記号によって囲まれたワード COPY を除く、その他の連続する一連の COBOL 文字で、区切り記号でもリテラルでもないもの。

テキスト名 (* text-name)

ライブラリー・テキストを識別するユーザー一定義語。

デバッグ・セクション (* debugging section)

USE FOR DEBUGGING ステートメントが含まれているセクション。

デバッグ行 (* debugging line)

行の標識区域に文字 D がある行のこと。

トークン (token)

COBOL エディターでは、プログラムにおける意味の単位。トークンには、データ、言語キーワード、ID、またはその他の言語構文の一部を含めることができる。

動詞 (* verb)

COBOL コンパイラーまたはオブジェクト・プログラムによってとられる処置を表すワード。

動的 CALL (dynamic CALL)

DYNAM オプションを使用してコンパイルさ

れたプログラム内の CALL *literal* ステートメント、またはプログラム内の CALL *identifier* ステートメント。

動的アクセス (* dynamic access)

1 つの OPEN ステートメントの実行範囲内において、特定の論理レコードを、大容量記憶ファイルからは順次アクセス以外の方法で取り出したりそのファイルに入れたりでき、またファイルからは順次アクセスの方法で取り出せるアクセス・モード。

特殊名記入項目 (* special names entry)

ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の記入項目。この記入項目は、通貨記号を指定したり、小数点を選択したり、シンボリック文字を指定したり、インプリメントする人の名前をユーザー指定の簡略名と関連付けたり、英字名を文字セットまたは照合シーケンスと関連付けたり、クラス名を一連の文字と関連付けたりするための手段を提供する。

特殊文字 (* special character)

以下のセットに属する文字。

文字	意味
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
\$	通貨記号
,	コンマ (小数点)
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符
(左括弧
)	右括弧
>	より大記号
<	より小記号
:	コロン

特殊レジスター (* special registers)

特定のコンパイラー生成ストレージ域のことで、その基本的な使用法は、具体的な COBOL 機能を使用したときに作り出される情報を記憶することである。

独立項目 (* noncontiguous items)

WORKING-STORAGE SECTION および LINKAGE SECTION 内の基本データ項目で、他のデータ項目と階層上の関係を持たないもの。

トップダウン開発 (top-down development)

構造化プログラミング (*structured programming*) を参照。

トップダウン設計 (top-down design)

関連付けられた諸機能が、構造の各レベルで実行されるようにする階層構造を使ったコンピューター・プログラムの設計。

トラブルシューティング (troubleshoot)

コンピューター・ソフトウェアの使用中に問題を検出し、突き止め、除去すること。

トレーラー・ラベル (trailer-label)

(1) 記録メディア・ユニットのデータ・レコードの後にある、ファイルまたはデータ・セットのラベル。(2) 「ファイル終わりラベル (*end-of-file label*)」の同義語。

ナ

内部 10 進数データ項目 (internal decimal data item)

USAGE PACKED-DECIMAL または USAGE COMP-3 として記述されており、項目を数値として定義する PICTURE 文字ストリング (記号 9、S、P、または V の有効な組み合わせ) を持っている、データ項目。「パック 10 進数データ項目 (*packed-decimal data item*)」と同義。

内部データ (* internal data)

プログラムの中で記述されるデータで、すべての外部データ項目および外部ファイル結合子を除いたもの。プログラムの LINKAGE SECTION で記述された項目は、内部データとして扱われる。

内部データ項目 (* internal data item)

実行単位内の 1 つのプログラムの中で記述されるデータ項目。内部データ項目は、グローバル名を持つことができる。

内部ファイル結合子 (* internal file connector)

実行単位内にあるただ 1 つのオブジェクト・プログラムのみがアクセスできるファイル結合子。

内部浮動小数点データ項目 (internal floating-point data item)

USAGE COMP-1 または USAGE COMP-2 として記述されているデータ項目。COMP-1 は、単精度浮動小数点データ項目を定義します。COMP-2 は、倍精度浮動小数点デー

タ項目を定義します。内部浮動小数点データ項目に関連した PICTURE 節はありません。

名前 (name)

COBOL オペランドを定義する 30 文字を超えないで構成されたワード。

名前空間 (namespace)

XML 名前空間 (*XML namespace*) を参照。

二分探索 (binary search)

二分探索では、探索の各ステップで、一連のデータ・エレメントの集合が 2 つに分割される。エレメントの数が奇数の場合には、何らかの適切なアクションが取られる。

入出力状況 (* I-O status)

入出力操作の結果としての状況を示す 2 文字の値を収める概念上のエンティティ。この値は、そのファイルについてのファイル制御記入項目で FILE STATUS 節を使用することによって、プログラムに使用可能にされる。

入出力ステートメント (* input-output statement)

個々のレコードに対して操作を行うことにより、またはファイルを 1 つの単位として操作することにより、ファイルの処理を行うステートメント。入出力ステートメントには、ACCEPT (ID 句付き)、CLOSE、DELETE、DISPLAY、OPEN、READ、REWRITE、SET (TO ON または TOOFF 句付き)、START、および WRITE がある。

入出力ファイル (* input-output file)

I-O モードでオープンされるファイル。

入出力モード (* I-O mode)

ファイルに対する I-O 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

入力ファイル (* input file)

入力モードでオープンされるファイル。

入力プロシージャ (* input procedure)

ソートすべき特定のレコードの解放を制御する目的で、SORT ステートメントの実行時に制御が渡されるステートメントの集合。

入力モード (* input mode)

ファイルに対する INPUT 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

ヌル (null)

無効なアドレスの値をポインター・データ項目に割り当てるために使用される形象数字。NULL を使えるところならばどこでも、NULLS を使用できる。

ネイティブ・メソッド (native method)

COBOL などの別のプログラム言語で記述されたインプリメンテーションを備える Java メソッド。

ネイティブ英数字データ項目 (native alphanumeric data item)

(XML 文書の場合) NATIVE 句で記述されたカテゴリ英数字データ項目、または CHAR(NATIVE) オプションを有効にしてコンパイルされたカテゴリ英数字データ項目。データ項目のエンコードは、有効なランタイム・ロケールの ASCII または UTF-8 コード・ページである。

ネストされたプログラム (nested program)

他のプログラムの中に直接的に含まれているプログラム。

年フィールド拡張 (year field expansion)

2 桁の年の日付フィールドを、ファイルおよびデータベースの中で完全な 4 桁の年になるように明示的に拡張した後、そのフィールドをプログラムの中で拡張形式で使用する。これは、2 桁の年を使用していたアプリケーションに対して確実に信頼できる日付処理を行う唯一の方法である。

ハ

バイト (byte)

特定の数のビット (通常 8 ビット) から成るストリングであり、1 つの単位として処理され、1 つの文字または制御機能を表す。

バイトコード (bytecode)

Java コンパイラによって生成され、Java インタープリターによって実行される、マシンから独立したコード。(Sun)

バイナリー項目 (binary item)

2 進表記 (基数 2 の数体系) で表される数値データ項目。等価の 10 進数は、10 進数字 0 から 9 に演算符号を加えたもので構成される。項目の左端ビットは演算符号。

配列 (array)

データ・オブジェクトで構成される集合体。それぞれのオブジェクトは添え字付けによって一意的に参照できる。配列は、COBOL ではテーブルに類似する。

パック 10 進数データ項目 (packed-decimal data item) 内部 10 進数データ項目 (*internal decimal data item*) を参照。

パッケージ (package)

関連する Java クラスの集まり。個々に、または全体としてインポートすることができる。

バッファ (buffer)

入力データまたは出力データを一時的に保持するために使用されるストレージの一部分。

パラメーター (parameter)

(1) 呼び出し側プログラムと呼び出し先プログラム間で受け渡されるデータ。(2) メソッド呼び出しの USING 句内のデータ・エレメント。引数によって、呼び出されたメソッドが要求された操作を実行するために使用できる追加情報を与える。

範囲区切りステートメント (* delimited scope statement)

明示的範囲終了符号を含んでいるステートメント。

範囲終了符号 (scope terminator)

PROCEDURE DIVISION の特定のステートメントの終わりを示す COBOL 予約語。これは明示的なもの (例えば、END-ADD など) であることもあれば、暗黙のもの (分離文字ピリオド) であることもある。

反復構造 (iteration structure)

ある条件が真である間、あるいはある条件が真になるまで、一連のステートメントが繰り返して実行されるプログラムの処理ロジック。

汎用オブジェクト参照 (universal object reference)

どのクラスのオブジェクトでも参照できるデータ名。

比較条件 (* relation condition)

ある算術式、データ項目、英数字リテラル、または索引名の値が、他の算術式、データ項目、英数字リテラル、または索引名の値と特定の関係があるという命題 (それに対して真理値を判別する)。関係演算子 (*relational operator*) も参照。

比較文字 (* relation character)

以下のセットに属する文字。

文字	意味
>	より大きい
<	より小さい
=	に等しい

引数 (argument)

(1) ID、リテラル、算術式、または関数 ID で、これにより関数の評価に使用する値を指定する。(2) CALL または INVOKE ステートメントの USING 句のオペランドであり、呼び出されたプログラムまたは起動されたメソッドに値を渡すのに使用されます。

ビッグ・エンディアン (big-endian)

メインフレームおよび AIX ワークステーションが 2 進データおよび UTF-16 文字を保管する際に使用するデフォルトの形式。この形式では、2 進数データ項目の最下位バイトが最上位のアドレスになり、UTF-16 文字の最下位バイトが最上位のアドレスになる。リトル・エンディアン (*little-endian*) と比較。

日付形式 (date format)

次のいずれかの方法で指定される、日付フィールドの日付パターン。

- DATE FORMAT 節または DATEVAL 組み込み関数 argument-2 によって明示的に。
- 日付フィールドを戻すステートメントまたは組み込み関数によって暗黙的に。詳細については、日付フィールド (*COBOL for AIX 言語解説書*) を参照してください。

日付フィールド (date field)

次のうちのいずれか。

- データ記述記入項目に DATE FORMAT 節が含まれているデータ項目。
- 次の組み込み関数の 1 つで戻される値。

DATE-OF-INTEGERR
DATE-TO-YYYYMMDD
DATEVAL
DAY-OF-INTEGERR
DAY-TO-YYYYDDDD
YEAR-TO-YYYYY
YEARWINDOW

- ACCEPT ステートメントの概念上のデータ項目 DATE、DATE YYYYMMDD、DAY、および DAY YYYYDDDD。
- ある種の算術演算の結果。詳細については、『日付フィールドを使用する算術計算』 (*COBOL for AIX 言語解説書*) を参照してください。

「日付フィールド (*date field*)」という用語は、「拡張日付フィールド (*expanded date field*)」と「ウィンドウ表示日付フィールド (*windowed date field*)」の両方を指す。非日付 (*nondate*) も参照。

非日付 (nondate)

次の項目のいずれか。

- 日付記述記入項目に DATE FORMAT 節が含まれていないデータ項目
- リテラル
- UNDATE 関数を使用して変換された日付フィールド
- 参照変更された日付フィールド
- 日付フィールド・オペランドを含む特定の算術演算の結果。例えば、2 つの互換日付フィールドの差

表示浮動小数点データ項目 (display floating-point data item)

暗黙的または明示的に USAGE DISPLAY として記述されており、外部浮動小数点データ項目を記述する PICTURE 文字ストリングを持っている、データ項目。

標準 COBOL 85 (Standard COBOL 85)

以下の標準によって定義された COBOL 言語。

- 「ANSI INCITS 23-1985, *Programming languages -COBOL*」は「ANSI INCITS 23a-1989, *Programming Languages -COBOL - Intrinsic Function Module for COBOL*」に改訂されました。
- 「ISO 1989:1985, *Programming languages -COBOL*」は「ISO/IEC 1989/AMD1:1992, *Programming languages -COBOL: Intrinsic function module*」に改訂されました。

部 (* division)

部の本体と呼ばれる、0 個、1 個、または複数個のセクションまたは段落の集合であり、特定の規則に従って形成および結合されたもの。それぞれの部は、部のヘッダーおよび関連した部の本体で構成される。COBOL プログラムには、見出し部、環境部、データ部、および手続き部の 4 つの部がある。

ファイル (* file)

論理レコードの集合。

ファイル・システム (file system)

データ・レコードおよびファイル記述プロトコルの特定のセットに準拠するファイルの集合、およびこれらのファイルを管理する一連のプログラム。

ファイル位置標識 (file position indicator)

概念的エンティティであり、索引付きファイルの場合は参照キー内の現行キーの値、順次ファイルの場合は現行レコードのレコード番号、相対ファイルの場合は現行レコードの相対レコード番号が入っている。あるいは、次の論理レコードが存在しないことを示すか、オプションの入力ファイルが使用可能でないことを示すか、AT END 条件が既に存在していることを示すか、もしくは有効な次のレコードが設定されていないことを示す。

ファイル記述記入項目 (* file description entry)

DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 FD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

ファイル結合子 (* file connector)

ファイルに関する情報が入っており、ファイル名と物理ファイルの間のリンケージとして、さらにファイル名とその関連レコード域の間のリンケージとして使用されるストレージ域。

ファイル制御記入項目 (* file control entry)

SELECT 節と、ファイルの関連物理属性を宣言するすべての従属節。

ファイル節 (* file clause)

DATA DIVISION の記入項目であるファイル記述項目 (FD 記入項目) およびソート・マージ・ファイル記述項目 (SD 記入項目) のいずれかの一部として現れる節。

ファイル属性対立条件 (* file attribute conflict condition)

ファイルに入出力操作の実行を試みて失敗した場合に、プログラムの中でそのファイルに対して指定されたファイル属性が、そのファイルの固定属性と一致しないこと。

ファイル編成 (* file organization)

ファイルの作成時に確立される永続論理ファイル構造。

ファイル名 (* file-name)

DATA DIVISION の FILE SECTION の中のファイル記述項目またはソート・マージ・ファイル記述項目で記述されるファイル結合子に名前を付けるユーザー定義語。

ファクトリー・データ (factory data)

いったんクラスに割り振られ、クラスのすべてのインスタンスに共用されるデータ。ファクトリー・データは、クラス定義の FACTORY 段落の DATA DIVISION の WORKING-STORAGE SECTION 内に宣言される。Java private 静的データと同義。

ファクトリー・メソッド (factory method)

オブジェクト・インスタンスとは無関係に、クラスによってサポートされるメソッド。ファクトリー・メソッドは、クラス定義の FACTORY 段落に宣言される。Java public 静的メソッドと同義。これらは通常オブジェクトの作成をカスタマイズすることに使用される。

フォーマット (* format)

一連のデータの特定の配置。

複合 ODO (complex ODO)

次のような OCCURS DEPENDING ON 節の特定の形式。

- 可変位置項目またはグループ:
DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、非従属データ項目またはグループが続く。グループは英数字グループでも国別グループでも構いません。
- 可変位置テーブル: DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、OCCURS 節によって記述された非従属データ項目が続く。
- 可変長エレメントを持つテーブル:
OCCURS 節によって記述されたデータ項目に、DEPENDING ON オプションを指定した OCCURS 節によって記述された従属データ項目が含まれている。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

複合条件 (* combined condition)

2 つ以上の条件を AND または OR 論理演算子で結合した結果生じる条件。条件 (condition) および 複合否定条件 (negated combined condition) も参照。

複合条件 (* complex condition)

1 つ以上の論理演算子が 1 つ以上の条件に基づいて作動する条件。条件 (condition)、単純否定条件 (negated simple condition)、および 複合否定条件 (negated combined condition) も参照。

複合否定条件 (* negated combined condition)

論理演算子 NOT とその直後に括弧で囲んだ複合条件を続けたもの。条件 (condition) および 複合条件 (combined condition) も参照。

含まれているプログラム (contained program)

別の COBOL プログラムにネストされている COBOL プログラム。

符号条件 (* sign condition)

データ項目や算術式の代数值が、0 より小

さいか、大きいか、または等しいかという命題で、それに関して真理値が判別できる。

物理レコード (* physical record)

ブロック (block) を参照。

浮動小数点 (floating point)

実数を 1 対の数表示で表す、数を表記するための形式。浮動小数点表記では、固定小数点部分 (最初の数表示) と、暗黙浮動小数点の底を指数で表される数だけ累乗して得られる値 (2 番目の数表示) との積が、実数になります。例えば、数値 0.0001234 の浮動小数点表記は 0.1234 -3 です (ここで、0.1234 は小数部であり、-3 は指数です)。

浮動小数点データ項目 (floating-point data item)

小数部と指数が入っている数値データ項目。その値は、小数部に、指数で指定されただけ累乗された数字データ項目の底を乗算することによって得られる。

部の見出し (* division header)

ワードとその後に続く、部の先頭を示す分離文字ピリオドの組み合わせ。部のヘッダーは次のとおり。

```
IDENTIFICATION DIVISION.ENVIRONMENT  
DIVISION.DATA DIVISION.PROCEDURE DIVISION.
```

古くなったエレメント (* obsolete element)

標準 COBOL 85 の COBOL 言語エレメントのうち、標準 COBOL 2002 から削除されたもの。

プログラム (program)

(1) コンピューターで処理するのに適した一連の命令。処理には、コンパイラーを使用してプログラムの実行準備をすることやランタイム環境を使用してプログラムを実行することが含まれます。(2) 1 つ以上の相互に関係のあるモジュールの論理アセンブリー。同じプログラムの複数のコピーを異なるプロセスで実行することができる。

プログラム識別記入項目 (* program identification entry)

IDENTIFICATION DIVISION の PROGRAM-ID 段落内の記入項目であり、プログラム名を指定し、選択されたプログラム属性をプログラムに割り当てる節が入っている。

プログラム終了マーカー (* end program marker)
語の組み合わせに分離文字ピリオドが続いたもので、COBOL ソース・プログラムの終わりを示す。プログラム終了マーカーは、次のように記述する。

END PROGRAM *program-name*.

プログラム名 (* program-name)

IDENTIFICATION DIVISION およびプログラム終了マーカーにおいて、COBOL ソース・プログラムを識別するユーザー定義語または英数字リテラル。

プロシージャ (* procedure)

PROCEDURE DIVISION 内にある 1 つの段落または論理的に連続する段落のグループ、あるいは 1 つのセクションまたは論理的に連続するセクションのグループ。

プロシージャ・ブランチ・ステートメント (* procedure branching statement)

ソース・コードの中にステートメントが書かれている順番どおりに次の実行可能ステートメントに制御の移動をせず、別のステートメントに明示的に制御の移動を引き起こすステートメント。プロシージャ分岐ステートメントは次のとおり。ALTER、CALL、EXIT、EXIT PROGRAM、GO TO、MERGE (OUTPUTPROCEDURE 句付き)、PERFORM および SORT (INPUTPROCEDURE または OUTPUT PROCEDURE 句付き)、XML PARSE。

プロシージャ・ポインター・データ項目 (procedure-pointer data item)

入り口点を指すポインターを保管できるデータ項目。USAGE IS PROCEDURE-POINTER 節で定義されるデータ項目には、プロシージャ入り口点のアドレスが入っている。一般的に、COBOL のプログラムと通信するために使用される。

プロシージャ統合 (procedure integration)

COBOL 最適化プログラムの機能の 1 つであり、実行されるプロシージャまたは含まれているプログラムへの呼び出しを単純化する。

PERFORM プロシージャ統合とは、PERFORM ステートメントが、実行されるプロシージャによって置き換えられるプロセスのこと。含まれているプログラムのプロシージャ統合とは、含まれているプロ

グラムへの呼び出しがプログラム・コードによって置き換えられるプロセスのこと。

プロシージャ名 (* procedure-name)

PROCEDURE DIVISION 中にある段落またはセクションに名前を付けるために使用されるユーザー定義語。プロシージャ名は、段落名 (これは修飾することができる) またはセクション名から構成される。

プロジェクト (project)

ダイナミック・リンク・ライブラリー (DLL) や他の実行可能ファイル (EXE) などのターゲットを作成するのに必要な、データおよびアクションの完全セット。

プロセス (process)

プログラムの全部または一部の実行中に発生する一連のイベント。複数のプロセスを並行して実行することができ、1 つのプロセス内で実行されるプログラムはリソースを共用することができる。

ブロック (* block)

通常は 1 つ以上の論理レコードで構成される物理的データ単位。大容量記憶ファイルの場合、ある論理レコードの一部がブロックに入ることがある。ブロックのサイズは、そのブロックが含まれているファイルのサイズと直接関係はなく、そのブロックに含まれているか、そのブロックにオーバーラップしている論理レコードのサイズとも直接関係はない。「物理レコード (*physical record*)」と同義。

文 (* sentence)

1 つ以上のステートメントの並びで、その最後のものは、分離文字ピリオドで終了する。

文書タイプ定義 (document type definition (DTD))

XML 文書のクラスの文法。XML タイプ定義 (*XML type definition*) を参照。

分離文字 (* separator)

文字ストリングを区切るために使用される、1 文字または連続する 2 文字。

分離文字コンマ (* separator comma)

文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのコンマ (,)。

分離文字セミコロン (* separator semicolon)
文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのセミコロン (;)。

分離文字ピリオド (* separator period)
ピリオド (.) 文字ストリングを区切るために使用されるもの。

ページ (page)
データの物理的分離を表す、出力データの垂直分割。分離は、内部論理要件または出力メディアの外部特性、あるいはその両方に基づいて行われる。

ページ本体 (* page body)
行を記述できる、または行送りすることができる (またはその両方ができる) 論理ページの部分。

米国規格協会 (American National Standards Institute: ANSI)
米国で認定された組織が自発的工業規格を作成して維持する手順を設定する組織であり、製造業者、消費者、および一般の利害関係者で構成される。

別個にコンパイルされたプログラム (* separately compiled program)
あるプログラムが、その中に含まれているプログラムと一緒に、それ以外のすべてのプログラムとは別個にコンパイルされること。

ヘッダー・ラベル (header label)
(1) 記録メディア・ユニットのデータ・レコードの前にあるファイル・ラベルまたはデータ・セット・ラベル。(2) 「ファイル開始ラベル (*beginning-of-file label*)」の同義語。

編集解除 (* de-edit)
項目の編集解除された数値を判別するために、数字編集データ項目からすべての編集文字を論理的に除去すること。

編集済みデータ項目 (edited data item)
0 の抑止または編集文字の挿入、あるいはその両方を行うことによって変更されたデータ項目。

編集用文字 (* editing character)
次に示す集合に属する 1 文字、または 2 文字で構成される固定した組み合わせ。

文字	意味
0	スペース
+	ゼロ
-	正符号
CR	負符号
DB	貸方
Z	借方
*	ゼロの抑止
\$	チェック・プロテクト
,	通貨記号
.	コンマ (小数点)
/	ピリオド (小数点)
/	斜線 (スラッシュ)

変数 (* variable)
オブジェクト・プログラムの実行によって変更を受ける可能性のある値を持つデータ項目。算術式で使われる変数は、数字基本項目でなければならない。

ポインター・データ項目 (pointer data item)
アドレス値を保管できるデータ項目。これらのデータ項目は、USAGE IS POINTER 節を使用してポインターとして明示的に定義される。ADDRESS OF 特殊レジスターは、ポインター・データ項目として暗黙的に定義されている。ポインター・データ項目は、他のポインター・データ項目と等価かどうか比較したり、他のポインター・データ項目に内容を移動したりできる。

ホスト英数字データ項目 (host alphanumeric data item) (XML 文書の場合) NATIVE 句が含まれないデータ記述記入項目を持つカテゴリ英数字データ項目のうち、CHAR(EBCDIC) オプションを有効にしてコンパイルされたもの。データ項目のエンコードは、有効な EBCDIC コード・ページである。このコード・ページは、EBCDIC_CODEPAGE 環境変数が設定されている場合はこの環境変数から決定され、設定されていない場合は、ランタイム・ロケールに関連付けられたデフォルトのコード・ページから決定される。

ボリューム (volume)
外部ストレージのモジュール。テープ装置の場合はリール、直接アクセス装置の場合はユニット。

マ

マージ・ファイル (* merge file)

MERGE ステートメントによってマージされるレコードの集まり。マージ・ファイルは、マージ機能により作成され、マージ機能によってのみ使用できる。

マルチスレッド化 (multithreading)

コンピューター内で複数のパスを使用して実行を行う並行操作。「マルチプロセッシング (multiprocessing)」と同義。

マルチタスキング (multitasking)

2 つ以上のタスクの並行実行またはインターリーブ実行を可能にする操作モード。

マルチバイト文字セット (MBCS) (multibyte character set (MBCS))

さまざまなバイト数で表された文字からなるコード化文字セット。例えば、EUC (拡張 UNIX コード)、UTF-8、1 バイト文字および 2 バイト EBCDIC 文字または ASCII 文字の混合物からなる文字セットなど。

無効キー条件 (* invalid key condition)

索引付きファイルまたは相対ファイルに関連するキーの特定値が無効であると判別された場合に生じる、実行時の条件。

明示的範囲終了符号 (* explicit scope terminator)

特定の PROCEDURE DIVISION ステートメントの有効範囲を終わらせる予約語。

命令ステートメント (* imperative statement)

命令の動詞で開始して、行うべき無条件の処置を指定するステートメント。または明示範囲終了符号によって区切られた条件ステートメント (範囲区切りステートメント)。1 つの命令ステートメントは、一連の命令ステートメントから構成することができる。

メインプログラム (main program)

プログラムとサブルーチンからなる階層において、プロセス内でプログラムが実行されたときに最初に制御を受け取るプログラム。

メガバイト、MB (* megabyte (MB))

1 メガバイトは 1,048,576 バイトに相当する。

メソッド (method)

オブジェクトによってサポートされる操作

の 1 つを定義し、そのオブジェクトに対する INVOKE ステートメントによって実行されるプロシージャ・コード。

メソッド定義 (* method definition)

メソッドを定義する COBOL ソース・コード。

メソッドの起動 (method invocation)

あるオブジェクトから別のオブジェクトへの通信で、受信オブジェクトにメソッドを実行するように要求するもの。

メソッド見出し記入項目 (* method identification entry)

IDENTIFICATION DIVISION の METHOD-ID 段落内の記入項目。この記入項目には、メソッド名を指定する節が入っている。

メソッド名 (method-name)

オブジェクト指向操作の名前。メソッドを起動するのに使用する場合、名前は、英数字リテラル、国別リテラル、カテゴリー英数字データ項目、またはカテゴリー国別データ項目にすることができます。メソッドを定義する METHOD-ID 段落で使用する場合、名前は英数字リテラルまたは国別リテラルにする必要があります。

文字 (* character)

言語のそれ以上分割できない基本単位。

文字 (* letter)

以下の 2 つのセットのいずれかに属する文字。

1. 英大文字: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
2. 英小文字: a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

文字位置 (character position)

1 文字を保持または表示するために必要な物理ストレージまたは表示スペースの量。この用語はどのような文字のクラスにも適用される。文字の特定のクラスについては、以下の用語が適用される。

- 英数字文字位置。USAGE DISPLAY を使用して表される DBCS 文字。
- DBCS 文字位置。USAGE DISPLAY-1 を使用して表される DBCS 文字。

- **国別文字位置**。USAGE NATIONAL を使用して表される文字。UTF-16 の文字エンコード・ユニット と同義。

文字エンコード・ユニット (character encoding unit)

コード化文字セット内の 1 つのコード・ポイントに相当するデータの単位。1 つ以上の文字エンコード・ユニットを使用して、コード化文字セットの文字が表現される。エンコード・ユニット と呼ばれる。

USAGE NATIONAL の場合、文字エンコード・ユニットは、UTF-16 の 2 バイト・コード・ポイントに対応している。

USAGE DISPLAY の場合、文字エンコード・ユニットは、1 つのバイトに対応している。

USAGE DISPLAY-1 の場合、文字エンコード・ユニットは、DBCS 文字セットの 2 バイト・コード・ポイントに対応している。

文字ストリング (character string)

COBOL ワード、リテラル、PICTURE 文字ストリング、またはコメント記入項目を形成する一連の連続した文字。文字ストリングは区切り文字で区切らなければならない。

文字セット (character set)

テキスト情報を表すために使用されるエレメントの集合。ただし、コード化表現は想定されていない。コード化文字セット (coded character set) も参照。

モジュール定義ファイル (module definition file)

ロード・モジュール内のコード・セグメントを記述するファイル。

ヤ

ユーザー定義語 (* user-defined word)

節やステートメントの形式を満たすためにユーザーが提供する必要のある COBOL ワード。

優先順位番号 (* priority-number)

セグメンテーションの目的で、PROCEDURE DIVISION 内のセクションを分類するユーザー定義語。セグメント番号には 0 から

9 までの文字だけしか使用できない。セグメント番号は 1 桁または 2 桁として表すことができる。

ユニット (unit)

直接アクセスのモジュール。その大きさは IBM によって規定されている。

呼び出し可能サービス (callable services)

言語環境プログラムでは、従来の言語環境プログラム定義の呼び出しインターフェースを使用して COBOL プログラムが呼び出すことができる一連のサービス。言語環境プログラムの規則を共有するすべてのプログラムがこれらのサービスを使用できる。

呼び出し側プログラム (* calling program)

別のプログラムへの CALL を実行するプログラム。

呼び出し先プログラム (called program)

CALL ステートメントの対象となるプログラム。呼び出し先プログラムと呼び出し側プログラムが実行時に結合されて、1 つの実行単位が作成される。

予約語 (* reserved word)

COBOL ソース・プログラムの中で使用することができるが、ユーザー定義語またはシステム名としてプログラムの中で使用されてはならないワードのリスト中に挙げられている COBOL ワード。

ラ

ライブラリー・テキスト (* library text)

COBOL ライブラリー内の一連のテキスト・ワード、コメント行、区切りスペース、または区切りの疑似テキスト区切り文字。

ライブラリー名 (* library-name)

COBOL ライブラリーの名前を表すユーザー定義語。与えられたソース・プログラムをコンパイルするためにコンパイラーが使用するライブラリーを識別する。

ラッパー (wrapper)

オブジェクト指向コードとプロシージャ指向コード間のインターフェースを提供するオブジェクト。ラッパーを使用すると、

他のシステムがプログラムを再利用したり、プログラムにアクセスしたりできるようになる。

ランタイム環境 (runtime environment)

COBOL プログラムが実行される環境。

ランダム・アクセス (* random access)

キー・データ項目のプログラム指定値を使用して、相対ファイルまたは索引付きファイルから取り出したり、削除したり、またはそこに代入したりする論理レコードを識別するアクセス・モード。

リール (reel)

ストレージ・メディアの個別部分。その大きさはインプリメントする人によって決定され、1 つのファイルの一部、1 つのファイルの全部、または任意の個数のファイルが収容される。「ユニット (unit)」および「ボリューム (volume)」と同義。

リソース (* resource)

オペレーティング・システムの制御下に置かれており、実行中のプログラムによって使用できる機能またはサービス。

リテラル (literal)

文字列を構成するために配列された文字によって、または形象定数を使用することによって、その値が決める文字ストリング。

リトル・エンディアン (little-endian)

Intel プロセッサが 2 進データおよび UTF-16 文字を保管する際に使用するデフォルトの形式。この形式では、2 進数データ項目の最上位バイトが最上位のアドレスになり、UTF-16 文字の最上位バイトが最上位のアドレスになる。ビッグ・エンディアン (big-endian) と比較。

リリアン日 (Lilian date)

グレゴリオ暦の開始以降の日数。第 1 日は 1582 年 10 月 15 日、金曜日。リリアン日フォーマットは、グレゴリオ暦の考案者であるルイジ・リリオにちなんだ名称。

リンク (link)

(1) リンク接続 (伝送メディア) と、それぞれがリンク接続の終端にある 2 つのリンク・ステーションの組み合わせ。1 つのリンクは、マルチポイントまたはトークン

リング構成において、複数のリンク間で共用できる。(2) データ項目あるいは 1 つ以上のコンピューター・プログラムの部分を相互接続すること。例えば、リンケージ・エディターによってオブジェクト・プログラムをリンクして共用ライブラリーを作成すること。

ルーチン (routine)

コンピューターに操作または一連の関連操作を実行させる、COBOL プログラム内の一連のステートメント。

ルーチン名 (* routine-name)

COBOL 以外の言語で記述されたプロシージャを識別するユーザー定義語。

レコード (* record)

論理レコード (logical record) を参照。

レコード・キー (record key)

索引付きファイル内のレコードを識別する内容を持つキー。

レコード域 (* record area)

DATA DIVISION の FILE SECTION 内のレコード記述項目で記述されるレコードを処理する目的で割り振られるストレージ域。FILE SECTION では、レコード域の現行の文字位置の数は、明示または暗黙の RECORD 節によって決められる。

レコード記述 (* record description)

レコード記述項目 (record description entry) を参照。

レコード記述項目 (* record description entry)

特定のレコードに関連したデータ記述記入項目全体。「レコード記述 (record description)」と同義。

レコード内データ構造 (* intrarecord data structure)

連続したデータ記述記入項目のサブセットによって定義される、1 つの論理レコードから得られるグループ・データ項目および基本データ項目の集合全体。これらのデータ記述記入項目には、レコード内データ構造を記述している最初のデータ記述記入項目のレベル番号より大きいレベル番号を持つすべての記入項目が含まれる。

レコード番号 (* record number)

編成が順次であるファイル内のレコードの順序数。

レコード名 (* record-name)

COBOL プログラムの DATA DIVISION 内のレコード記述項目で記述されるレコードに名前を付けるユーザー定義語。

列 (* column)

印刷行または参照形式行におけるバイト位置。列は、行の左端の位置から始めて行の右端の位置まで、1 から 1 ずつ増やして番号が付けられる。列は 1 つの 1 バイト文字を保持する。

レベル番号 (* level-number)

階層構造におけるデータ項目の位置を示すか、またはデータ記述記入項目の特性を示す、2 桁の数字で表されたユーザー定義語。1 から 49 のレベル番号は、論理レコードの階層構造におけるデータ項目の位置を示す。1 から 9 のレベル番号は、1 桁の数字として書くことも、0 の後に有効数字を付けて書くこともできる。レベル番号 66、77、および 88 は、データ記述記入項目の特性を識別する。

レベル標識 (* level indicator)

特定のタイプのファイルを識別するか、または階層での位置を識別する 2 つの英字。DATA DIVISION 内のレベル標識には、CD、FD、および SD がある。

連続項目 (* contiguous items)

DATA DIVISION 内の連続する記入項目によって記述され、相互に一定の階層関係を持っている項目。

ロケール (locale)

プログラム実行環境の一連の属性であり、文化的に重要な考慮事項を示す。例えば、文字コード・ページ、照合シーケンス、日時形式、通貨表記、数値表記、または言語など。

論理演算子 (* logical operator)

予約語 AND、OR、または NOT のいずれか。条件の形成において、AND または OR、あるいはその両方を論理連結語として使用できる。NOT は論理否定に使用できる。

論理レコード (* logical record)

最も包括的なデータ項目。レコードのレベル番号は 01。レコードは、基本項目またはグループ項目のどちらでもよい。「レコード (record)」と同義。

ワ

ワークステーション (workstation)

エンド・ユーザーが使用するコンピューターの総称 (パーソナル・コンピューター、3270 端末、インテリジェント・ワークステーション、および UNIX 端末を含む)。ワークステーションはメインフレームまたはネットワークに接続されることがよくある。

ワード (* word)

ユーザー定義語、システム名、予約語、または関数名を形成する、30 文字を超えない文字ストリング。

割り当て名 (assignment-name)

COBOL ファイルの編成を識別する名前前で、システムがこれを認識する際に使用する。

数字

1 バイト文字セット (single-byte character set (SBCS))

各文字が 1 バイトで表現される文字のセット。ASCII および EBCDIC (拡張 2 進化 10 進コード) (EBCDIC (Extended Binary-Coded Decimal Interchange Code)) も参照。

2 バイト文字セット (double-byte character set (DBCS))

それぞれの文字が 2 バイトで表現される 1 組の文字。256 個のコード・ポイントで表現される記号より多くの記号を含んでいる言語 (日本語、中国語、および韓国語など) は、2 バイト文字セットを必要とする。各文字に 2 バイトが必要なため、DBCS 文字の入力、表示、および印刷には、DBCS を受け入れ可能なハードウェアおよびサポートされるソフトウェアが必要。

77 レベル記述記入項目 (77-level-description-entry)

レベル番号 77 を持つ不連続データ項目を記述するデータ記述記入項目。

A

ASCII

情報交換用米国標準コード。7 ビットのコード化文字をベースとする 1 つのコード化文字セット (パリティ・チェックを含む 8 ビット) を使用する標準コードであり、データ処理システム、データ通信システム、および関連装置の間での情報交換に使用される。ASCII セットは、制御文字と図形文字から構成されている。

IBM は、ASCII に対する拡張 (文字 128 から 255) を定義している。

ASCII ベース・マルチバイト・コード・ページ (ASCII-based multibyte code page)

UTF-8、EUC、または ASCII DBCS コード・ページ。各 ASCII ベース・マルチバイト・コード・ページは、1 バイト文字とマルチバイト文字の両方を含む。1 バイト文字のエンコードは ASCII エンコードである。

AT END 条件 (AT END condition)

次のような特定の条件のもとで、READ、RETURN、または SEARCH ステートメントを実行した場合に引き起こされる条件。

- 順次アクセス・ファイルに対して READ ステートメントを実行中に、そのファイル内に次の論理レコードが存在しない場合、または相対レコード番号中の有効数字の桁数が相対キー・データ項目のサイズより大きい場合、またはオプションの入力ファイルが使用可能でない場合。
- RETURN ステートメントの実行中に、関連するソート・ファイルまたはマージ・ファイルについての次の論理レコードが存在しない場合。
- SEARCH ステートメントの実行中に、関連する WHEN 句のいずれかで指定された条件を満足することなく、検索操作が終了した場合。

B

byte order mark (BOM)

UTF-16 または UTF-32 テキストの先頭に使用して、後続テキストのバイト・オーダーを示す Unicode 文字。バイト・オーダーには、「ビッグ・エンディアン (big-endian)」または「リトル・エンディアン (little-endian)」がある。

C

CCSID

コード化文字セット ID (coded character set identifier) を参照。

COBOL 文字セット (* COBOL character set)

COBOL 構文を作成する際に使用される文字セット。完全な COBOL 文字セットは、以下にリストする文字で構成される。

文字	意味
0,1, . . . ,9	数字
A,B, . . . ,Z	英大文字
a,b, . . . ,z	英小文字
	スペース
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
\$	通貨記号
,	コンマ (小数点)
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符
(左括弧
)	右括弧
>	より大記号
<	より小記号
:	コロン

COBOL ワード (* COBOL word)

ワード (word) を参照。

* CONFIGURATION SECTION

ENVIRONMENT DIVISION のセクションであり、ソース・プログラムとオブジェクト・プログラムの全体的な仕様およびクラス定義を記述する。

CONSOLE

オペレーター・コンソールと関連する COBOL 環境名。

D

DATA DIVISION

COBOL プログラムまたはメソッドの 1 つの部。使用するファイルおよびファイルに含まれるレコード、必要となる内部作業用ストレージ・レコード、COBOL 実行単位内の複数のプログラムで使用可能なデータを記述する。

DBCS

2 バイト文字セット (*double-byte character set (DBCS)*) を参照

DBCS データ項目 (DBCS data item)

少なくとも 1 つの記号 G または少なくとも 1 つの記号 N (NSYMBOL (DBCS) コンパイラー・オプションが有効なとき) を含んでいる PICTURE 文字ストリングで記述されたデータ項目。DBCS データ項目は USAGE DISPLAY-1 を持っています。

DBCS 文字 (DBCS character)

IBM の 2 バイト文字セットで定義された任意の文字。

DBCS 文字位置 (DBCS character position)

文字位置 (*character position*) を参照。

Do 構造 (do construct)

構造化プログラミングでは、DO ステートメントを使えば、プロシーチャー内の複数のステートメントをグループ化できる。

COBOL では、インライン PERFORM ステートメントが同様に機能する。

do-until

構造化プログラミングにおいて、do-until ループは、少なくとも 1 回は実行され、所定の条件が真になるまで実行される。COBOL では、TEST AFTER 句を PERFORM ステートメントで使用すれば、同様に機能する。

do-while

構造化プログラミングにおいて、do-while ループは、所定の条件が真である場合、および真である間に実行される。COBOL では、TEST BEFORE 句を PERFORM ステートメントで使用すれば、同様に機能する。

E

EBCDIC (拡張 2 進化 10 進コード) (* EBCDIC (Extended Binary-Coded Decimal Interchange

Code))

8 ビット・コード化文字をベースとするコード化文字セット。

EBCDIC 文字 (EBCDIC character)

EBCDIC (拡張 2 進化 10 進コード) セットに含まれているいずれかの記号。

ENVIRONMENT DIVISION

COBOL プログラム、クラス定義、またはメソッド定義の 4 つの主コンポーネントの 1 つ。ENVIRONMENT DIVISION では、ソース・プログラムがコンパイルされるコンピューターと、オブジェクト・プログラムが実行されるコンピューターを記述する。この部では、ファイルの論理概念とそのレコードの間のリンケージ、およびファイルが保管される装置の物理的局を提供する。

Extensible Markup Language

XML を参照。

F

* FILE SECTION

DATA DIVISION のセクションであり、ファイル記述項目、ソート・マージ・ファイル記述項目、および関連するレコード記述が入っている。

FILE-CONTROL 段落 (FILE-CONTROL paragraph)

ENVIRONMENT DIVISION 内の段落であり、この中では、特定のソース単位で使用されるデータ・ファイルが宣言される。

I

* I-O-CONTROL

ENVIRONMENT DIVISION 段落の名前。この段落では、再実行開始点についてのオブジェクト・プログラム要件、複数データ・ファイルによる同じ区域の共用、および単一入出力装置上の複数のファイル・ストレージが指定される。

I-O-CONTROL 記入項目 (* I-O-CONTROL entry)

ENVIRONMENT DIVISION の I-O-CONTROL 段落内の記入項目であり、プログラム実行中に指定のファイルへのデータの伝送と処理を行うために必要な情報を提供する節が入っている。

IBM COBOL 拡張部分 (IBM COBOL extension)

COBOL 85 標準で記述されるもの以外で、IBM コンパイラーでサポートされる COBOL 構文とセマンティクス。

ICU *International Components for Unicode (ICU)* を参照。

ID (* identifier)

データ項目に名前を付けるための文字ストリングと区切り文字の構文的に正しい組み合わせ。関数ではないデータ項目を参照するときは、ID は、データ名と、修飾子、添え字、または参照修飾子 (一意的に参照するために必要な場合) から構成される。関数であるデータ項目を参照する際には、関数 ID が使われる。

IDENTIFICATION DIVISION

COBOL プログラム、クラス定義、またはメソッド定義の 4 つの主コンポーネントの 1 つ。IDENTIFICATION DIVISION では、プログラム、クラス、またはメソッドを識別する。IDENTIFICATION DIVISION には、作成者名、インストール、または日付を含めることができる。

IGZCBSO

COBOL for AIX のブートストラップ・ルーチン。このルーチンは、COBOL for AIX プログラムが含まれているモジュールとリンク・エディットしなければならない。

* INPUT-OUTPUT SECTION

ENVIRONMENT DIVISION のセクションであり、オブジェクト・プログラムまたはメソッドに必要なファイルおよび外部メディアに名前を付け、実行時にデータの伝送および処理に必要な情報を提供する。

International Components for Unicode (ICU)

IBM が後援、サポート、および使用するオープン・ソース開発プロジェクト。ICU ライブラリーは、AIX をはじめする、さまざまなプラットフォーム上で、強力かつフル機能の Unicode サービスを提供する。

is-a 継承階層におけるクラスおよびサブクラスの特徴を表す関係。あるクラスに対して is-a 関係を持つサブクラスは、そのクラスから継承する。

J

J2EE *Java 2 Platform, Enterprise Edition (J2EE)* を参照。

Java 2 Platform, Enterprise Edition (J2EE)

Sun Microsystems, Inc. が定義する、エンタープライズ・アプリケーションの開発とデプロイメントのための環境。J2EE プラットフォームは、マルチスレッドの Web ベース・アプリケーションを開発するための機能を提供する、一群のサービス、アプリケーション・プログラミング・インターフェース (API)、およびプロトコルで構成される。(Sun)

Java Native Interface (JNI)

Java 仮想マシン (JVM) 内で実行される Java コードが、他のプログラム言語で記述されたアプリケーションおよびライブラリーと連携できるようにするプログラミング・インターフェース。

Java 仮想マシン (JVM) (Java virtual machine (JVM))

コンパイル済みの Java プログラムを実行する中央演算処理装置のソフトウェア・インプリメンテーション。

JVM *Java 仮想マシン (JVM) (Java virtual machine (JVM))* を参照。

K

K 記憶容量に関連して使われるときは、2 の 10 乗。10 進表記では 1024。

L

* LINAGE-COUNTER

ページ本体内の現在位置を指す値を収めた特殊レジスター。

LINKAGE SECTION

呼び出し先のプログラムまたはメソッドの DATA DIVISION 内のセクションであり、呼び出し側プログラムまたはメソッドから使用可能なデータ項目が記述される。これらのデータ項目は、呼び出し側プログラムまたはメソッドおよび呼び出し先プログラムまたはメソッドの両方から参照できる。

* LOCAL-STORAGE SECTION

DATA DIVISION のセクションであり、

VALUE 節で割り当てられた値に応じて、呼び出し単位で割り振りまたは解放が行われるストレージを定義する。

M

Make ファイル (makefile)

アプリケーションに必要なファイルのリストが収められたテキスト・ファイル。make ユーティリティはこのファイルを使用して、ターゲット・ファイルを最新の変更で更新する。

MBCS マルチバイト文字セット (MBCS) (*multibyte character set (MBCS)*) を参照。

O

* OBJECT-COMPUTER

ENVIRONMENT DIVISION にある段落の名前であり、ここではオブジェクト・プログラムが実行されるコンピューター環境が記述される。

ODBC *Open Database Connectivity (ODBC)* を参照。

ODO オブジェクト (ODO object)

次の例では、X が OCCURS DEPENDING ON 節のオブジェクト (ODO オブジェクト) である。

```
WORKING-STORAGE SECTION01 TABLE-1.  
05 X          PICS9. 05  
Y OCCURS 3 TIMES          DEPENDING  
ON X          PIC X.
```

ODO オブジェクトの値によって、テーブル内の ODO サブジェクトの数が決まる。

ODO サブジェクト (ODO subject)

上記の例では、Y が OCCURS DEPENDING ON 節のサブジェクト (ODO サブジェクト) である。テーブル内の ODO サブジェクトの数である Y の値は、X の値によって決まる。

Open Database Connectivity (ODBC)

さまざまなデータベースおよびファイル・システムのデータへのアクセスを可能にするアプリケーション・プログラミング・インターフェース (API) の仕様。

P

private

ファクトリー・データまたはインスタンス・データに適用されるため、そのデータを定義するクラスのメソッドだけがアクセス可能である。

PROCEDURE DIVISION

COBOL の部の 1 つで、問題を解決するための命令を記述する。

PROCEDURE DIVISION の終わり (* end of PROCEDURE DIVISION)

COBOL ソース・プログラムにおいて、それ以後にはプロシーチャーが存在しない物理的な位置。

R

RSD ファイル・システム (RSD file system)

RSD (レコード順次区切り) ファイル・システムは、順次ファイルをサポートするワークステーション・ファイル・システムである。RSD ファイルは、固定長レコードのすべての COBOL データ・タイプをサポートし、ほとんどのファイル・エディターで編集可能であり、他の言語で作成されたプログラムによって読み取ることができる。

S

SBCS 1 バイト文字セット (SBCS) (*single-byte character set (SBCS)*) を参照。

SFS ファイル・システム (SFS file system)

Encina の SFS (Structured File Server) ファイル・システムは、順次ファイル・アクセスおよびキー索引付きファイル・アクセスをサポートするレコード単位ファイル・システムである。SFS ファイルは PL/I アプリケーションと共用できる。

* SOURCE-COMPUTER

ENVIRONMENT DIVISION にある段落の名前であり、ここではソース・プログラムがコンパイルされるコンピューター環境が記述される。

SPECIAL-NAMES

ENVIRONMENT DIVISION にある段落の名前。この段落では、環境名がユーザー指定の簡略名と関連付けられる。

STL ファイル・システム (STL file system)

STL (標準言語) ファイル・システムは、COBOL および PL/I 用のネイティブ・ワークステーション・ファイル・システムである。このシステムは、順次ファイル、相対ファイル、および索引付きファイルをサポートする。

U

Unicode

現代世界の各国の言語で記述されるテキストの交換、処理、表示をサポートする汎用文字エンコード標準。

UTF-8、UTF-16、UTF-32 など、Unicode を表現する複数のエンコード・スキームがある。COBOL for AIX では、国別データ・タイプの表記としてビッグ・エンディアン・フォーマットの UTF-16 を使用して Unicode をサポートしている。

UPSI スイッチ (UPSI switch)

ハードウェア・スイッチの機能を実行するプログラム・スイッチ。UPSI-0 から UPSI-7 の 8 つのスイッチがある。

| **URI** リソースを一意に指す文字のシーケンスの
| ことで、COBOL for AIX では、名前空間
| の ID。URI 構文は、文書「*Uniform*
| *Resource Identifier (URI): Generic Syntax*」
| で定義されています。

| **URI** *URI* を参照。

V

VSAM ファイル・システム (VSAM file system)

COBOL の順次編成、相対編成、および索引編成をサポートするファイル・システム。

W

Web サービス (Web service)

特定のタスクを実行し、HTTP や SOAP といったオープン・プロトコルを介してアクセス可能なモジュラー・アプリケーション。

* WORKING-STORAGE SECTION

独立項目または作業用ストレージ・レコー

ドあるいはその両方から構成される、作業用ストレージ・データ項目を記述する DATA DIVISION。

X

x PICTURE 節内の記号であり、コンピュータの有する文字セットの任意の文字を含めることができる。

XML Extensible Markup Language。マークアップ言語を定義するための標準メタ言語。SGML から派生した、SGML のサブセットである。XML では、SGML の複雑で使用頻度の低い部分が省略され、文書タイプを扱うアプリケーションの作成、構造化情報の作成および管理、異種コンピューター・システム間での構造化情報の伝送および共有はるかに容易になっている。XML を使用するとき、SGML で必要とされるような堅固なアプリケーションや処理は不要である。XML は、World Wide Web Consortium (W3C) の主導で開発された。

XML 宣言 (XML declaration)

使用している XML のバージョンや文書のエンコードなど、XML 文書の特性を指定する XML テキスト。

XML タイプ定義 (XML type definition)

あるクラスの文書に対する文法を規定するマークアップ宣言を含む、または指示する XML エlement。この文法は、文書タイプ定義または DTD と呼ばれる。

XML データ (XML data)

XML エlementを持つ階層構造に編成されたデータ。データ定義は XML エlement・タイプ宣言で定義される。

| XML 名前空間 (XML namespace)

| エlement名および属性名のコレクション
| のスコープを制限する、W3C XML 名前
| 空間仕様によって定義されたメカニズム。
| 一意的に選択された XML 名前空間によ
| って、複数の XML 文書または XML 文
| 書内の複数のコンテキストでElement名
| または属性名が一意的に識別されます。

XML 文書 (XML document)

W3C XML 仕様で定義される形式のデータ・オブジェクト。

資料名リスト

COBOL for AIX

インストール・ガイド (GC88-5745)

言語解説書 (SC88-5743)

プログラミング・ガイド (SC88-5744)

VSAM ファイル・システム 応答メッセージ (SC88-5746)

サポート

COBOL for AIX のご使用の際に問題がある場合は、サイト: www.ibm.com/software/awdtools/cobol/aix/support/ を参照してください。そこでは最新のサポート情報が提供されています。

関連資料

AIX

コマンド・リファレンス第 1 巻 (a から c) (SC88-4583)

コマンド・リファレンス第 2 巻 (d から h) (SC88-4582)

コマンド・リファレンス第 3 巻 (i から m) (SC88-4581)

コマンド・リファレンス第 4 巻 (n から r) (SC88-4580)

コマンド・リファレンス第 5 巻 (s から u) (SC88-4579)

コマンド・リファレンス第 6 巻 (v から z) (SC88-4578)

プログラミングの一般概念: プログラムの作成およびデバッグ (SC88-4577)

XL C/C++ for AIX

最適化およびプログラミング・ガイド (SC88-4922)

DB2 for Linux、UNIX、および Windows

コマンド・リファレンス (SC88-4432)

SQL リファレンス第 1 巻 (SC88-4436)

SQL リファレンス第 2 巻 (SC88-4437)

インフォメーション・センター (publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp) も参照してください。

Java

IBM SDK for Java - Tools Documentation (publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp)

Java 2 SDK, Standard Edition Documentation (java.sun.com/j2se/1.4.2/docs/)

The Java Language Specification, Third Edition (Gosling 他著)、java.sun.com/docs/books/jls/

The Java Native Interface、java.sun.com/j2se/1.4.2/docs/guide/jni/

Java Technology Edition SDK User Guides (www.ibm.com/developerworks/java/jdk/aix/service.html)

TXSeries for Multiplatforms

CICS Administration Reference (SC34-6641)

CICS Application Programming Guide (SC34-6634)

CICS Application Programming Reference (SC34-6640)

CICS Problem Determination Guide (SC34-6636)

Encina 管理ガイド 第2巻: サーバー管理 (SD88-7403)

Encina SFS プログラミング・ガイド (SD88-7412)

Unicode および文字表現

Unicode, www.unicode.org/

International Components for Unicode: Converter Explorer (demo.icu-project.org/icu-bin/convexp/)

Character Data Representation Architecture: Reference and Registry (SC09-2190)

XML

Extensible Markup Language (XML),
www.w3.org/XML/

Namespaces in XML 1.0, www.w3.org/TR/xml-names/

Namespaces in XML 1.1, www.w3.org/TR/xml-names11/

XML specification, www.w3.org/TR/xml/

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

アクセシビリティ

キーボード・ナビゲーション xxii

支援テクノロジー xxii

本書の xxii

COBOL for AIX xxi

アクティブ・ロケール 209

アセンブラー

プログラム

リスト 291, 595

LIST オプションから 595

アセンブラー言語プログラム

デバッグ 354

遊びバイト, 定義 617

値の割り当て 27

アドレス

入り口点アドレスを渡す 526

増分 523

プログラム間での受け渡し 523

NULL 値 523

アドレスを増分する 523

アプリケーション, 移植

言語の違い 493

プラットフォームの違い 493

メインフレームとワークステーション間

コンパイラー・オプションの選択

493

ワークステーション上でのメインフレーム・アプリケーションの実行

494

ワークステーションとメインフレーム間

ワークステーション専用の言語機能

498

ワークステーション専用の名前

499

COPY を使用したプラットフォーム固有のコードの分離

493

Windows と AIX 間 499

アプリケーションの移植

概要 493

環境の違い 497

言語の違い 493

アプリケーションの移植 (続き)

ファイル状況キー 497

プラットフォーム間の違い 493

分離符号の影響 42

マルチタスキング 499

マルチバイト 496

メインフレームとワークステーション間

コンパイラー・オプションの選択

493

ワークステーション上でのメインフレーム・アプリケーションの実行

494

ワークステーションとメインフレーム間

ワークステーション専用の言語機能

498

ワークステーション専用のコンパイラー・オプション

498

ワークステーション専用の名前

499

CICS 364

COPY を使用したプラットフォーム固有のコードの分離

493

SBCS 495

Windows と AIX 間 499

暗黙の範囲終了符号 21

移行の考慮事項

概要 615

32 ビット・モードから 64 ビット・モードへ 617

異常終了, ERRCOUNT ランタイム・オプションを使用した誘発

326

一時作業ファイルの場所

TMP による指定 228

一括表示表現 588

移動, 制御権の

ネストされたプログラム 503

メインプログラムとサブプログラム

501

呼び出し側プログラム 501

呼び出し先プログラム 501

COBOL プログラム相互間の

入り口点

アドレスの受け渡し 526

関数ポインター・データ項目 526

代替, ENTRY ステートメントの

定義 535

プロシージャー・ポインター・データ

項目 526

インスタンス

削除 457

作成 455

定義 427

インスタンス・データ

初期化 455

それをアクセス可能にする 442

定義 427, 434, 460

private 435

インスタンス・メソッド

オーバーライド 440

多重定義 441

定義 427, 436, 461

呼び出し, オーバーライドした 454

インスタンス・メソッドの多重定義 441

インポート・ファイル 251

インライン PERFORM

概要 94

例 95

ウィンドウ表示日付フィールド

契約 584

受け渡し, プログラム間でのデータの

アドレス 523

呼び出し側プログラムの引数 519

呼び出し先プログラムのパラメーター

519

BY CONTENT 517

BY REFERENCE 517

BY VALUE

概要 517

制限 519

EXTERNAL データ 527

Java との 482

JNI サービス 478

OMITTED 引数 520

RETURN-CODE 特殊レジスターでの

526

英字データ

国別との比較 204

これを伴う MOVE ステートメント

33

英数字グループ項目

定義 24

GROUP-USAGE NATIONAL なしのグループ

25

英数字データ

これを伴う MOVE ステートメント

33

比較する

国別と 204

照合シーケンスの影響 217

- 英数字データ (続き)
 - 比較する (続き)
 - ZWB の影響 318
 - 変換
 - MOVE による国別への 198
 - NATIONAL-OF による国別への 198
- 英数字の日付フィールドの縮小 584
- 英数字比較 90
- 英数字編集データ
 - これを伴う MOVE ステートメント 33
 - 初期化
 - 例 29
 - INITIALIZE の使用 73
- 英数字リテラル
 - 制御文字 26
 - 説明 26
 - マルチバイトの内容での 206
- エクスポート・ファイル 251
- エラー
 - コンパイラー・オプションの競合 265
 - 算術 170
 - 実行時のフラグ設定 325
 - 処理 169
 - メッセージ・テーブル
 - 指標付けを使用した例 76
 - 添え字付けを使用した例 75
- エラー検査、実行時のフラグ設定 325
- エラー・メッセージ
 - 各国語の設定 228
 - コンパイラー
 - 形式 244
 - 作成する重大度レベルの判別 286
 - 重大度レベル 242
 - ソースの訂正 242
 - ソース・リストへの組み込み 338
 - フラグを立てる重大度の選択 338
 - リストにおける位置 244
 - リストの生成 243
 - コンパイラーの指示 243
 - ランタイム
 - 形式 781
 - 不完全または省略 254
 - リスト 781
 - dspeat コマンドを使用したリスト表示 781
- エンコード
 - 言語文字 188
 - 構文解析対象の XML 文書 390
 - 生成された XML 出力での制御 411
 - 説明 196
 - XML 文書での矛盾 404
 - XML 文書の 397, 398
- エンコード宣言
 - 指定 400
- エンコード宣言 (続き)
 - 省略を推奨 399
- オーバーフロー条件
 - ストリングの結合および分割 169
- CALL 178
- UNSTRING 102
- オーバーライド
 - インスタンス・メソッド 440
 - ファクトリー・メソッド 466
- 大文字への変換 112
- お客様のサポート 871
- オブジェクト
 - 削除 457
 - 作成 455
 - 定義 427
- オブジェクト参照
 - 型式化 448
 - サイズは ADDR に依存 267
 - 設定 449
 - 汎用 448
 - 比較する 449
 - 引き渡しの例 452
 - ローカルからグローバルへの変換 455
 - 32 ビット・モードおよび 64 ビット・モード 617
- オブジェクト指向 COBOL
 - アプリケーションの準備 258
 - オブジェクト指向プログラムの書き込み 427
 - 互換 615
 - コンパイル 257
 - 実行
 - 概要 260
 - Java SDK サポート 261
 - 制限
 - CICS 427
 - EXEC CICS ステートメント 427
 - EXEC SQL ステートメント 427
 - SQL コンパイラー・オプション 427
 - リンク
 - 概要 258
 - 例 259
 - Java との通信 483
 - オブジェクト・インスタンス、定義 427
 - オブジェクト・インスタンスの解放 457
 - オブジェクト・コード
 - 互換 615
 - 生成 276
 - オプション
 - リンカーに対する指定 248
- 外部 10 進数データ
 - 国別 46
 - ゾーン 46
- 外部クラス名 433, 447
- 外部コード・ページ、定義 399
- 外部ファイル 528
- 外部浮動小数点データ
 - 国別 46
 - 表示 46
- カウント
 - 生成される XML 文字 408
 - 文字 (INSPECT) 110
- 拡張モード 42, 631
- カスタマイズ
 - 環境変数の設定 225
- 型式化オブジェクト参照 448
- 各国語サポート
 - メッセージ 213
- 各国語サポート (NLS)
 - 照合シーケンス 216
 - データ処理 183
 - ロケール (locale) 209
 - ロケールおよびコード・ページ値へのアクセス 220
 - ロケールおよびコード・ページの指定 228
 - ロケールの設定 209
 - ロケール・ベースの照合 216
 - DBCS 205
- 仮定による世紀ウィンドウ、非日付用の 574
- 可変位置グループ 644
- 可変位置データ項目 644
- 可変長テーブル
 - 値の割り当て 80
 - オーバーレイを防ぐ 646
 - 作成 77
 - 例 79
 - ロードの例 79
- 可変長レコード
 - OCCURS DEPENDING ON (ODO) 節 593
- 環境、事前初期設定
 - 概要 555
 - 例 557
 - C/C++ プログラム 509
- 環境の違い、zSeries およびワークステーション 497
- 環境変数
 - アクセス 226
 - コンパイラー 226
 - 設定
 - 概要 225
 - コマンド・シェル内 225
 - プログラム内 226
 - ロケール (locale) 212

[力行]

- ガーベッジ・コレクション 457
- 回避、コーディング・エラーの 587

環境変数 (続き)
 設定 (続き)
 .profile 内 225
 設定およびアクセスの例 234
 定義 225
 ファイルの割り当て 228
 優先順位 226
 ランタイム 228
 割り当て名 228
 CLASSPATH
 説明 228
 Java クラスの場所の指定 260
 COBCPYEXT 226
 COBJVMINIOPTIONS
 説明 228
 JVM オプションの指定 261
 COBLSTDIR 226
 COBMSGs 228
 COBOPT 226
 COBPATH
 説明 228
 CICS 動的呼び出し 366
 COBRTOPT 228
 DB2DBDFT 226
 EBCDIC_CODEPAGE 228
 ENCINA_CDS_ROOT 228
 ENCINA_SFS_DATA_VOLUME 228
 ENCINA_SFS_INDEX_VOLUME 228
 ENCINA_VSAM_AUTO_FLUSH 228
 ENCINA_VSAM_CACHE 228
 LANG 228
 LC_ALL 211, 228
 LC_COLLATE 211, 228
 LC_CTYPE 211, 228
 LC_MESSAGES 211, 228
 LC_TIME 211, 228
 LIBPATH
 説明 228
 COBOL クラスの場所の指定 260
 library-name 226, 319
 LOCPATH 228
 NLSPATH 228
 PATH
 説明 228
 SYSIN、SYSIPT、SYSOUT、
 SYSLIST、SYSLST、CONSOLE、
 SYSPUNCH、SYSPCH 228
 SYSLIB 226
 JNI.cpy の場所の指定 257
 text-name 226, 319
 TMP 228
 TZ 228
 環境変数を使用したファイルへのアクセス
 228
 環境名 7
 漢字データのテスト 207

漢字比較 90
 関数ポインター・データ項目
 サイズは ADDR に依存 267
 定義 526
 呼び出し可能サービスへのパラメータ
 の受け渡し 526
 32 ビット・モードおよび 64 ビット・
 モード 617
 JNI サービスのアドレッシング 705
 簡略名
 SPECIAL-NAMES 段落 7
 完了コード
 ソート 164
 マージ 164
 キー
 許容データ型
 MERGE ステートメントでの 162
 OCCURS 節の 66
 SORT ステートメントでの 162
 ソート用の
 概要 156
 定義 162
 デフォルト 218
 テーブル・エレメントの順序を指定す
 るための 66
 二分探索用の 82
 マージ用の
 概要 156
 定義 162
 デフォルト 218
 キーボード・ナビゲーション xxii
 記述、コンピューター 7
 基本ファイル、Encina SFS 122
 逆方向ブランチの回避 588
 キャッシュ 151
 キャッシュ、クライアント側
 環境変数 228
 説明 151
 挿入キャッシュ 151
 読み取りキャッシュ 151
 行、テーブルの 67
 行順次ファイル
 ファイル・アクセス・モード 134
 編成 134
 強制操作
 環境変数 228
 説明 153
 抑制 153
 行番号 345
 共用
 データ
 概要 517
 再帰的またはマルチスレッド化され
 たプログラムの 17
 名前の有効範囲 505

共用 (続き)
 データ (続き)
 パラメーター受け渡しの仕組み
 517
 別のプログラムからの 16
 別々にコンパイルされたプログラム
 間での 527
 別々にコンパイルされたプログラムの
 16
 メソッドから値を戻す 454
 メソッドへの引数の引き渡し 451
 coding the LINKAGE
 SECTION 520
 Java との 482
 PROCEDURE DIVISION ヘッダー
 521
 RETURN-CODE 特殊レジスター
 526
 ファイル
 名前の有効範囲 505
 EXTERNAL 節の使用 13, 528
 GLOBAL 節の使用 13
 共用ライブラリー
 オブジェクト指向のための作成 258
 オブジェクト指向プログラムでの使用
 260
 概要 536
 構築 535
 例 537
 サブプログラムおよび最外部プログラ
 ム 535
 参照の解決 536
 定義 535
 ディレクトリー・パスの設定 228
 目的 536
 利点と欠点 536
 CICS についての考慮事項 366
 Java とのインターオペラビリティで
 の使用 260
 Java とのインターオペラビリティの
 ための 258
 切り替え状況条件 90
 国別 10 進数データ (USAGE
 NATIONAL)
 形式 46
 初期化の例 30
 定義 192
 例 41
 国別グループ項目
 英数字グループと比べた場合の利点
 193
 英数字リテラルを伴う VALUE 節の例
 75
 概要 192
 基本項目として扱われる
 ほとんどの場合 25, 192

国別グループ項目 (続き)

基本項目として扱われる (続き)

MOVE の例 34

国別データのみを含むことができる
25, 194

グループ項目として扱われる

要約 195

INITIALIZE で 32

INITIALIZE の例 195

MOVE CORRESPONDING で 34

これを伴う MOVE ステートメント
34

使用

概要 194

基本項目として 195

初期化

INITIALIZE の使用 32, 73

VALUE 節の使用 75

生成された XML 文書の 408

テーブルの定義 66

定義 194

引数として渡す 522

例 25

Java との通信 483

LENGTH 組み込み関数および 118

USAGE NATIONAL グループとの対比
25

国別データ

英数字リテラルを伴う VALUE 節の例
117

外部 10 進数 46

外部浮動小数点 46

キーの

MERGE ステートメントでの 162

OCCURS 節の 66

SORT ステートメントでの 162

組み込み関数を使用した評価 114

形象定数 191

検査 (INSPECT) 110

これを伴う MOVE ステートメント
33, 197

最小項目または最大項目の検出 115

指定 189

条件式の 202, 203

初期化の例 29

生成された XML 文書の 408

その参照変更 107

定義 189

比較する

英字、英数字、または DBCS と
204

英数字グループと 205

概要 202

照合シーケンスの影響 219

数値との 204

2 つのオペランド 203

国別データ (続き)

比較する (続き)

NCOLLSEQ の影響 203

分割 (UNSTRING) 102

変換

大文字または小文字への 112

概要 197

ギリシャ語の英数字との間、例
200

中国語 GB 18030 との間の 201

DISPLAY-OF による英数字への
199

INSPECT による 110

MOVE による英数字、DBCS、ま
たは整数からの 198

NATIONAL-OF による英数字また
は DBCS からの 198

NUMVAL、NUMVAL-C による数
値への 113

UTF-8 との間の 201

文字の逆順 112

リテラル

使用 190

連結 (STRING) 100

ACCEPT による入力 36

DATE FORMAT 節と一緒に使用で
きない 562

DISPLAY による出力 37

Java との通信 483

LENGTH OF 特殊レジスター 118

LENGTH 組み込み関数および 118

USAGE 節がない場合の NSYMBOL

コンパイラ・オプション 190

XML 文書のエンコード 398

国別比較

国別浮動小数点データ (USAGE

NATIONAL)

定義 46, 192

国別編集データ

これを伴う MOVE ステートメント
33

初期化

例 30

INITIALIZE の使用 73

定義 189

編集記号 189

PICTURE 節 189

国別リテラル

使用 190

説明 26

国/地域別情報の定義 209

組み込み SQL

利点 371

ODBC との比較 371

組み込みエラー・メッセージ 338

組み込み関数

英数字データ項目の変換 111

国別データ項目の変換 111

コンパイルの日付の検索 118

最大または最小項目の検出 115

参照修飾子としての 109

紹介 37

照合シーケンスの影響 219

数字関数

整数、浮動小数点、混合 56

ネストされた 57

引数としてのテーブル・エレメント
57

引数としての特殊レジスター 57

用途 56

例 56

中間結果 637, 640

データ項目の長さの検出 118

データ項目の評価 114

テーブル・エレメントの処理 83

ネスト 38

日付および時刻 650

例

ANNUITY 59

CHAR 115

CURRENT-DATE 58

DISPLAY-OF 200

INTEGER 109

INTEGER-OF-DATE 58

LENGTH 58, 116, 118

LOG 59

LOWER-CASE 112

MAX 58, 84, 115, 116

MEAN 59

MEDIAN 59, 84

MIN 109

NATIONAL-OF 200

NUMVAL 113

NUMVAL-C 58, 113

ORD 115

ORD-MAX 84, 116

PRESENT-VALUE 58

RANGE 59, 84

REM 59

REVERSE 112

SQRT 59

SUM 84

UPPER-CASE 112

WHEN-COMPILED 118

CEELOCT との互換性 676

DATEVAL

使用 579

例 580

UNDATE

使用 580

例 580

- 組み込み相互参照
 - 説明 342
 - 例 351
- 組み込みの CICS 変換プログラム
 - 概要 369
 - 利点 369
 - cicstcl -p コマンド 369
- 組み込みマップ要約 341, 346
- クライアント
 - 定義 445
- クライアント側のキャッシュ
 - 環境変数 228
 - 説明 151
 - 挿入キャッシュ 151
 - 読み取りキャッシュ 151
- クラス
 - インスタンス化
 - COBOL 456
 - Java 455
 - インスタンス・データ 434
 - オブジェクト、JNI による参照の取得 478
 - 定義 427, 430
 - ファクトリー・データ 463
 - ユーザー定義 10
 - name
 - 外部 433, 447
 - プログラムにおいて 432
- クラス条件
 - データの妥当性検査 333
 - テスト
 - 概要 90
 - 漢字の 207
 - 数値の 54
 - DBCS の 208
- クラスター・ファイル 126
- グリニッジ標準時 (GMT)
 - 現地時間までのオフセットの取得 (CEEGMTO) 671
 - リリアン日付およびリリアン秒の戻り (CEEGMT) 669
- グループ移動と基本移動の対比 34, 195
- グループ項目
 - 可変位置 644
 - 国別グループ内で英数字グループを従属させることはできない 194
 - 国別データと比較 205
 - グループ移動と基本移動の対比 34, 195
 - グループ項目として扱われる
 - INITIALIZE で 32
 - INITIALIZE の例 72
 - これを伴う MOVE ステートメント 34
 - 初期化
 - INITIALIZE の使用 31, 72
- グループ項目 (続き)
 - 初期化 (続き)
 - VALUE 節の使用 74
 - テーブルの定義 65
 - 定義 24
 - 引数として渡す 522
 - 32 ビット・モードおよび 64 ビット・モード 617
- グレゴリオ文字ストリング
 - 現地時間の戻り (CEELOCT) 676
 - 例 677
- グローバル名 506
- ケース構造、EVALUATE ステートメント 87
- 計算
 - 算術データ項目 590
 - 指標の 70
 - 添え字の 593
 - 重複 589
 - 定数データ項目 589
- 形象定数
 - 国別文字 191
 - 定義 27
 - HIGH-VALUE の制約事項 191
- 継承の階層の定義 429
- 継続
 - 構文検査 276
 - プログラム 171
- 言語間通信
 - 制約事項 618
 - COBOL および Java 間の 477
 - COBOL と C/C++ の間
 - 概要 508
 - 制約事項 508
- 検査、有効データの
 - 条件式 90
- 検索規則、リンカーの 252
- 現地時間
 - 取得 (CEELOCT) 676
- コーディング
 - インスタンス・メソッド 436, 461
 - オブジェクト指向プログラム
 - 概要 427
 - 技法 12, 587
 - クライアント 445
 - クラス定義 430
 - 決定 85
 - 効率的 587
 - コンストラクター・メソッド 464
 - サブクラス
 - 概要 458
 - 例 461
 - 条件テスト 91
 - 単純化 601
 - テーブル 65
 - テスト条件 91
- コーディング (続き)
 - 手続き部 17
 - 入出力
 - 概要 137
 - 例 138
 - ファイルに対する
 - 概要 137
 - 例 138
 - ファイル入出力 132
 - ファクトリー定義 462
 - ファクトリー・メソッド 464
 - ループ 94
 - CICS での制約事項 364
 - CICS のもとで実行されるプログラム 363
 - 概要 364
 - システム日付の取得 365
 - DATA DIVISION 12
 - DB2 のもとで実行されるプログラム
 - 概要 357
 - ENVIRONMENT DIVISION 7
 - EVALUATE ステートメント 87
 - IDENTIFICATION DIVISION 5
 - IF ステートメント 85
 - Java との相互運用が可能なデータ型 483
 - SFS ファイル、例 149
 - SQL ステートメント 358
- コード
 - コピー 601
 - 最適化 596
- コード化文字セット
 - 定義 188
 - XML 文書における 397
- コード・ページ
 - アクセス 221
 - 英字データ項目の 210
 - 英数字データ項目の 210
 - オーバーライド 199
 - 国別データ項目の 210
 - システム・デフォルト 212
 - 指定 399
 - 照会 221
 - 定義 188
 - 特殊文字の 16 進値 400
 - 文字の使用 210
 - ユーロ通貨サポート 63
 - 有効 213
 - ASCII 211
 - DBCS データ項目の 210
 - EBCDIC 211
 - EUC 211
 - UTF-8 211
 - XML 文書での矛盾 404
- コード・ポイント、定義 188

- 構成ファイル
 - スタンザ 241
 - 調整 240
 - デフォルト 239
 - 変更 239
- 構造、INITIALIZE による初期化 31
- 構造化プログラミング 588
- 構文エラー
 - NOCOMPILE コンパイラー・オプションによる検出 336
- 構文解析データ項目、定義 390
- 構文図の読み方 xvi
- 項目のシステム名への関連付け 7
- 効率、コーディングの 587
- 互換
 - オブジェクト指向構文 615
 - オブジェクト・コード 615
 - 小容量形式ファイル 131
 - データ項目サイズ 616
 - 日付
 - 比較における 570
 - DATA DIVISION 570
 - PROCEDURE DIVISION 570
 - 32 ビット・モードおよび 64 ビット・モード 617
 - FILEMODE 131
 - Java および COBOL 615
 - SFS (Encina) ファイル 616
 - XML 生成 616
- 互換モード 42, 631
- 固定小数点演算
 - 指数 634
 - 比較 61
 - 評価 60
 - 例の評価 61
- 固定小数点データ
 - 外部 10 進数 46
 - 固定小数点と浮動小数点との間の変換 52
 - 使用計画 590
 - 中間結果 633
 - パック 10 進数 48
 - 変換と精度 52
 - 2 進数 47
- 固定世紀ウィンドウ 565
- コピーブック
 - 検索規則 319
 - 使用 601
 - 相互参照 350
 - 探索 245
 - library-name 環境変数 226
 - ODBC を使用する場合 376
 - odbc3
 - 説明 376
 - 例 378
- コピーブック (続き)
 - odbc3d
 - 説明 376
 - ソース 382
 - 例 378
 - odbc3p
 - 説明 376
 - ソース 379
 - 例 378
 - SYSLIB による検索パスの指定 226
 - コピーブック相互参照、記述 340
 - コピー・コード、ユーザー提供モジュールからの取得 282
 - コピー・ライブラリー
 - 例 602
 - コマンド行の引数
 - 使用 532
 - host オプションを使用しない例 532
 - host オプションを使用する例 533
 - コマンド・プロンプト、環境変数の定義 225
 - 小文字への変換 112
 - コンパイラー
 - エラー・メッセージのリストの生成 243
 - 制限
 - DATA DIVISION 12
 - 中間結果の計算 632
 - 日付関連のメッセージ、分析 581
 - メッセージ
 - 作成する重大度レベルの判別 286
 - 重大度レベル 242
 - ソース・リストへの組み込み 338
 - フラグを立てる重大度の選択 338
 - 呼び出し 236
 - コンパイラー指示ステートメント
 - 概要 21
 - 説明 319
 - コンパイラー・オプション
 - 移植性のための 493
 - コンパイラー呼び出しでの 344
 - 指定
 - 環境変数 226
 - シェル・スクリプト内 237
 - cob2 コマンド 236
 - cob2_j コマンド 236
 - cob2_r コマンド 236
 - COBOPT の使用 226
 - 状況 344
 - 省略形 263
 - その表 263
 - デバッグ用
 - 概要 336
 - TEST に関する制約事項 334
 - THREAD に関する制約事項 334
 - 矛盾する 265
- コンパイラー・オプション (続き)
 - ADATA 266
 - ADDR 266
 - APOST 301
 - ARITH 268
 - BINARY 269
 - CALLINT 270
 - CHAR 271
 - CICS 273
 - CICS の場合 368
 - COLLSEQ 274
 - COMPILE 276
 - CURRENCY 277
 - DATEPROC 278
 - DIAGTRUNC 279
 - DYNAM 280, 597
 - ENTRYINT 281
 - EXIT 281
 - FLAG 286, 338
 - FLAGSTD 288
 - FLOAT 289
 - LIB 290
 - LINECOUNT 291
 - LIST 291, 342
 - LSTFILE 292
 - MAP 293, 341, 342
 - MAXMEM 294
 - MDECK 295
 - NCOLLSEQ 296
 - NOCOMPILE 336
 - NSYMBOL 296
 - NUMBER 297, 343
 - OPTIMIZE 298, 595
 - パフォーマンスの考慮事項 597
 - PGMNAME 299
 - PROCESS (CBL) による指定 238
 - PROFILE 301
 - QUOTE 301
 - SEPOBJ 302
 - SEQUENCE 303
 - SIZE 304
 - SOSI 305
 - SOURCE 306, 342
 - SPACE 307
 - SPILL 308
 - SQL
 - コーディング・サブオプション 360
 - 説明 308
 - SSRANGE 309, 337
 - パフォーマンスの考慮事項 597
 - TERMINAL 310
 - TEST
 - 説明 310
 - デバッグに使用 341
 - パフォーマンスの考慮事項 597

コンパイラー・オプション (続き)

- THREAD
 - 説明 311
 - デバッグに関する制約事項 334
 - パフォーマンスの考慮事項 597
- TRUNC 312
 - パフォーマンスの考慮事項 597
- VBREF 315, 342
- WSCLEAR 315
- XREF 316, 340
- YEARWINDOW 317
- ZWB 318

コンパイラー・リスト

- 出力ディレクトリーの指定 226
- 入手 342

コンパイル

- オブジェクト指向アプリケーション
 - 例 259
 - cob2_j コマンド 257
- 構成ファイルの調整 240
- 調整 240
- 統計 344
- プログラム 235
- SFS ファイル・システム用 238

コンパイルおよびリンク

- オブジェクト指向アプリケーション
 - 例 259
 - cob2_j コマンド 258

コンパイル時についての考慮事項

- エラー・メッセージ
 - 作成する重大度レベルの判別 286
 - 重大度レベル 242
- コンパイラーの指示エラー 243
- コンパイルおよびリンク・ステップの表示 247
- コンパイル済みプログラム 245
- デフォルト以外の構成ファイルの使用 247
- 表示後のコンパイルおよびリンク・ステップの実行 247
- リンクなしでのコンパイル 245
- cob2 ヘルプの表示 247

[サ行]

再帰呼び出し

- コーディング 515
- 識別 6
- LINKAGE SECTION 17

最後に使われた状態

- EXIT PROGRAM または GOBACK でのサブプログラム 502

最大または最小項目、検出 115

最適化

- 一括表示表現 588
- 一貫性のあるデータ 591

最適化 (続き)

- 逆方向分岐の回避 588
- 構造化プログラミング 588
- コンパイラー・オプションの影響 597
- 指標計算 593
- 指標付け 592
- 添え字計算 593
- 添え字付け 592
- 重複計算 589
- テーブル・エレメント 592
- 定数計算 589
- 定数データ項目 589
- トップダウン・プログラミング 588
- ネストされたプログラムの統合 596
- バック 10 進数 データ項目 591
- パフォーマンスにおける意味 593
- パフォーマンスへの影響 588
- パラメーター引き渡しの影響 520
- 含まれているプログラムの統合 596
- 未使用データ項目 298
- ライン外の PERFORM 588
- ALTER ステートメントの回避 588
- BINARY データ項目 590
- MAXMEM 294
- OCCURS DEPENDING ON 593

最適化プログラム

- 概要 596
- 再入可能コード 619
- サイン表記 53
- 索引
 - キーの誤りの検出 177
 - Encina SFS ファイル 126
- 索引付きファイル
 - ファイル・アクセス・モード 134
 - Encina SFS ファイル 126
- 索引付きファイル編成 134
- 削除、ファイルからのレコードの作成 146
- 作成
 - オブジェクト 455
 - 可変長テーブル 77
 - 代替索引ファイル 148
 - SFS ファイル
 - 環境変数 148
 - fsadmin コマンド 150
- 作成、互換コードの 498

サブクラス

- インスタンス・データ 460
- コーディング
 - 概要 458
 - 例 461

サブストリング

- その参照変更 106
- テーブル・エレメントの 107

サブプログラム

- 共用ライブラリー内 535
- 使用 501

サブプログラム (続き)

- 説明 501
- 定義 517
- メインプログラム 501
- リンケージ
 - 共通データ項目 519
- PROCEDURE DIVISION 521
- サポート 871
- 参考文献 871
- 算術
 - エラー処理 170
 - 組み込み関数を使用した 56
 - コーディングが容易な COMPUTE ステートメント 54
 - 日付の計算
 - 現在のグリニッジ標準時の取得 (CEEGMT) 669
 - タイム・スタンプから秒数への変換 (CEESECS) 685
 - 日付から COBOL 整数形式への変換 (CEECLDY) 651
 - 日付からリリアン形式への変換 (CEEDAYS) 663

算術演算

- MLE を使用した 575, 576

算術式

- 括弧で囲まれた 55
- 参照修飾子としての 109
- 説明 55
- 非算術ステートメントでの 640
- MLE を使用した 576

算術比較 61

算術評価

- 固定小数点と浮動小数点の対比 60
- 精度 631
- 中間結果 631
- データ形式の変換 51
- パフォーマンスに関するヒント 590
- 変換と精度 52
- 優先順位 55, 633
- 例 60, 61

参照修飾子

- 組み込み関数、例 109
- としての算術式 109
- としての変数 107

参照変更

- 国別データ 107
- 組み込み関数 106
- 生成された XML 文書 408
- テーブル 69, 107
- 範囲外の値 107
- 例 108
- SSRANGE による検査式 309
- UTF-8 文書 201

- シェル・スクリプトを使用したコンパイル 237

- 支援テクノロジー xxii
- 時間、現地の取得 (CEELOCT) 676
- 時間帯情報
 - TZ による指定 228
- シグニチャー
 - 固有でなければならない 437
 - 定義 437
- 時刻情報、形式 228
- 指数
 - 固定小数点演算で評価される 634
 - パフォーマンスに関するヒント 591
 - 浮動小数点演算で評価される 639
- システム日付
 - CICS のもとで 365
- システム名 7
- 実行、計算の
 - 日時サービス 604
- 実行時
 - パフォーマンスの考慮事項 587
 - 引数 532
 - ファイル名の変更 11
 - プラットフォーム間の違い 494
 - メッセージ 781
- 実行する、オブジェクト指向アプリケーションを
 - 概要 260
 - Java SDK サポート 261
- 実行単位
 - 終了 510
 - マルチスレッド化での役割 545
- 指標
 - 値を割り当てる 70
 - エレメントの変位の計算例 68
 - 初期化 71
 - 増分または減分 71
 - 他のテーブルの参照 70
 - 定義 69
 - 範囲検査 337
 - OCCURS INDEXED BY 節による作成 70
- 指標付け
 - エレメントの変位の計算例 68
 - 添え字付けより望ましい 592
 - テーブル 70
 - 定義 69
 - 例 76
- 指標データ項目
 - サイズは ADDR に依存 267
 - 添え字や指標として使用できない 71
 - 32 ビット・モードおよび 64 ビット・モード 617
 - USAGE IS INDEX 節による作成 70
- 縮小、英数字日付の 584
- 出力
 - 概要 132
 - ファイルへ 121
- 出力プロシージャ
 - コーディング 160
 - 制限 160
 - 例 163
 - RETURN または RETURN INTO が必要 160
- 順次ファイル
 - ファイル・アクセス・モード 133
 - 編成 133
- 状況コード、ファイル
 - 概要 175
 - 例 176
- 条件式
 - EVALUATE ステートメント 85
 - IF ステートメント 85
 - PERFORM ステートメント 96
- 条件処理
 - 日時サービスおよび 604
 - ERRCOUNT の影響 326
- 条件ステートメント
 - オブジェクト参照 449
 - 概要 19
 - NOT 句を指定した 20
- 条件の検査 91
- 条件名 573
- 照合シーケンス
 - 移植性に関する考慮事項 495
 - 英数字 217
 - 英数字および DBCS オペランドへの COLLSEQ の影響 274
 - 国別 219
 - 国別オペランドへの NCOLLSEQ の影響 296
 - 国別キーのバイナリー 162
 - 国別ソートまたはマージ・キーのバイナリー 219
 - 組み込み関数と 219
 - 指定 8
 - シンボリック文字 10
 - 制御 216
 - 代替
 - 選択 163
 - 例 9
 - 非数値比較 8
 - 文字の序数位置 115
 - ASCII 9
 - DBCS 218
 - EBCDIC 8
 - HIGH-VALUE 8
 - ISO 7 ビット・コード 9
 - LOW-VALUE 8
 - MERGE 8, 163
 - NATIVE 8
 - SEARCH ALL 8
 - SORT 8, 163
 - STANDARD-1 8
- 照合シーケンス (続き)
 - STANDARD-2 8
- 商標 833
- 初期化
 - インスタンス・データ 455
 - 可変長グループ 80
 - 国別グループ項目
 - INITIALIZE の使用 32, 73
 - VALUE 節の使用 75
 - グループ項目
 - INITIALIZE の使用 31, 72
 - VALUE 節の使用 74
 - 構造、INITIALIZE による 31
 - テーブル
 - エレメントのすべての出現 75
 - グループ・レベルの 74
 - それぞれの項目の個別の 74
 - INITIALIZE の使用 72
 - PERFORM VARYING の使用 96
 - ランタイム環境
 - 概要 555
 - 例 557
 - 例 28
- 処理
 - チェーン・リスト
 - 概要 523
 - 例 523
 - テーブル
 - 指標付けを使用した例 76
 - 添え字付けを使用した例 75
- 資料名リスト 871
- 診断、プログラムの 344
- シンボリック定数 589
- スイッチおよびフラグ
 - スイッチをオフに設定する例 93
 - スイッチをオンに設定する例 93
 - 説明 91
 - 単一値のテストの例 92
 - 定義 91
 - 複数値のテストの例 92
 - リセット 92
- 数学
 - 組み込み関数 56, 59
- 数字組み込み関数
 - 整数、浮動小数点、混合 56
 - ネストされた 57
 - 引数としてのテーブル・エレメント 57
 - 引数としての特殊レジスター 57
 - 用途 56
 - 例
 - ANNUITY 59
 - CURRENT-DATE 58
 - INTEGER 109
 - INTEGER-OF-DATE 58
 - LENGTH 58, 116

- 数字組み込み関数 (続き)
 - 例 (続き)
 - LOG 59
 - MAX 58, 84, 115, 116
 - MEAN 59
 - MEDIAN 59, 84
 - MIN 109
 - NUMVAL 113
 - NUMVAL-C 58, 113
 - ORD 115
 - ORD-MAX 84
 - PRESENT-VALUE 58
 - RANGE 59, 84
 - REM 59
 - SQRT 59
 - SUM 84
- 数字編集データ
 - 初期化
 - 例 30
 - INITIALIZE の使用 73
 - 定義 190
 - 編集記号 43
 - BLANK WHEN ZERO 節
 - コーディング、数値データでの 190
 - 例 43
 - PICTURE 節 43
 - USAGE DISPLAY
 - 初期化の例 30
 - 表示 43
 - USAGE NATIONAL
 - 初期化の例 31
 - 表示 43
- 数値データ
 - 外部 10 進数
 - USAGE DISPLAY 46
 - USAGE NATIONAL 46
 - 外部浮動小数点
 - USAGE DISPLAY 46
 - USAGE NATIONAL 46
 - 国別 10 進数 (USAGE NATIONAL) 46
 - 国別との比較 204
 - 国別浮動小数点 (USAGE NATIONAL) 46
 - ストレージ形式 44
 - ゾーン 10 進数 (USAGE DISPLAY) 形式 46
 - サイン表記 53
 - 定義 41
 - 内部浮動小数点
 - USAGE COMPUTATIONAL-1 (COMP-1) 48
 - USAGE COMPUTATIONAL-2 (COMP-2) 48
- 数値データ (続き)
 - パック 10 進数
 - サイン表記 53
 - USAGE COMPUTATIONAL-3 (COMP-3) 48
 - USAGE PACKED-DECIMAL 48
 - 表示浮動小数点 (USAGE DISPLAY) 46
 - 変換
 - 固定小数点と浮動小数点との間の 51
 - 精度 52
 - MOVE による国別への 198
 - 編集記号 43
 - 2 進数
 - USAGE BINARY 47
 - USAGE COMPUTATIONAL (COMP) 47
 - USAGE COMPUTATIONAL-4 (COMP-4) 47
 - USAGE COMPUTATIONAL-5 (COMP-5) 47
 - PICTURE 節 41, 43
 - USAGE DISPLAY 41
 - USAGE NATIONAL 41
 - USAGE とは無関係に代数値の比較が可能 204
- 数値のクラス・テスト
 - 検査、有効データの 54
- 数値比較 90
- 数値リテラルの説明 26
- スタック・スペース、cob2 コマンドを使用した割り振り 249
- スタック・スペースの割り振り 249
- スタック・フレーム、縮小 510
- スタティック・リンク
 - 概要 535
 - 欠点 536
 - 定義 506
 - 利点 536
- スタンプ
 - 構成ファイル内の属性 241
 - 説明 240
 - 追加 240
 - cob2 240
 - cob2_j 239
 - cob2_r 240
- ステートメント
 - 暗黙の範囲終了符号 21
 - コンパイラ指示 21
 - 条件付き 19
 - 定義 18
 - 範囲区切り 20
 - 明示範囲終了符号 21
 - 命令ステートメント 19
- ステートメント・ネスト・レベル 345
- スライド、テーブル 593
- ストリング
 - 処理 99
 - ヌル終了 522
- Java
 - 宣言する 484
 - 取り扱い 488
- ストレージ
 - 引数のための 519
 - マッピング 342
 - 文字データ 196
 - 割り振りは ADDR に依存 267
- スライディング世紀ウィンドウ 565
- スレッド化
 - および事前初期設定 549
 - 制御転送 549
 - プログラムの終了 550
- 世紀ウィンドウ
 - 概要 610
 - 固定 565
 - 照会および変更の例 610
 - スライディング 565
 - 非日付用の仮定による 574
 - CEECBLDY 653
 - CEEDAYS 665
 - CEEQCEN 678
 - CEESCEN 679
 - CEESECS 687
- 制御
 - 移動 501
 - ネストされたプログラム内の 503
 - プログラムのフロー 85
- 制限
 - オブジェクト指向プログラム 427
 - 添え字付け 69
 - 入出力プロシージャ 160
 - CICS
 - 概要 364
 - 分離型の変換プログラム 369
- 制限事項、コンパイラの
 - データ・サイズの互換性 616
 - ユーザー・データ 12
 - DATA DIVISION 12
- 整数
 - リリアン秒の変換 (CEESECI) 681
- 静的データ、定義 427
- 静的メソッド
 - 定義 427
 - 呼び出し 466
- 製品サポート 871
- セクション
 - グループ化 97
 - 説明 18
 - 宣言 22
- 設定
 - 指標 71

設定 (続き)

- 指標データ項目 70
- スイッチおよびフラグ 92
- リンカー・オプション 248

ゼロ比較 (符号条件を参照) 576

ゼロ抑制

- BLANK WHEN ZERO 節の例 43
- PICTURE 記号 Z 43

宣言型プロシージャ

- EXCEPTION/ERROR 173
- USE FOR DEBUGGING 334

全日付フィールド拡張の利点 564

ソース・コード

- 行番号 345, 346, 349
- リストの説明 342

ソート

- エラー番号
- 考えられる値のリスト 165
- iwzGetSortErrno での取得 165

完了コード 164

キー

- 概要 156
- 定義 162
- デフォルト 218

基準 162

終了 168

出力プロシージャ

- コーディング 160
- 例 163

診断メッセージ 164

正常終了の判別 164

説明 155

代替照合シーケンス 163

入出力プロシージャに関する制約事項 160

入力プロシージャ

- コーディング 158
- 例 163

ファイルの説明 157

プロセス 156

ゾーン 10 進数データ (USAGE

DISPLAY)

英数字との比較に対する ZWB の影響 318

形式 46

サイン表記 53

例 41

相互参照

- 組み込み 342
- コピーブック 342
- データおよびプロシージャ名 340
- テキスト名およびファイル名 340
- 動詞 342
- 動詞リスト 315
- 特殊定義記号 351
- プログラム名 350

相互参照 (続き)

- リスト 316
- COPY/BASIS 350
- COPY/BASIS ステートメント 342

相対ファイル

- ファイル・アクセス・モード 134
- 編成 134

挿入キャッシュ 151

添え字

- 計算 594
- 定義 69
- 範囲検査 337
- 変数、例 68
- リテラル、例 68

添え字付け

- 参照変更 69
- 制限 69
- 相対 69
- データ名またはリテラルを使用 69
- 定義 69
- 変数、例 68
- リテラル、例 68
- 例 75

属性メソッド 442

[夕行]

ダーティ読み取り 152

代替索引

追加 150

代替索引、定義 134

代替索引の追加 150

代替索引ファイル

- データ・ボリュームの指定 148
- ファイル名の指定 123

代替照合シーケンス

- 選択 163
- 例 9

代替ファイル・システム

- 環境変数を使用 228
- ファイル・システム ID 228

ダイナミック・リンク

- 共用ライブラリーの参照の解決 536
- 定義 506

タイム・スタンプ 659, 685

多重継承、許可されていない 430, 459

探索

テーブル

- 概要 80
- 逐次探索 81
- 二分探索 82
- パフォーマンス 81
- 名前の宣言の 506

短縮リストの例 343

ダンプ、TRAP(OFF) の副作用 329

端末へのメッセージの送信 310

段落

概要 18

グループ化 97

チェーン・リスト処理

概要 523

例 523

遅延書き込み

環境変数 228

使用可能化 153

違い、ホスト COBOL との 619

置換

- データ項目 (INSPECT) 110
- ファイル内のレコード 145

置換文字 191

逐次探索

説明 81

例 81

チューニング考慮事項、パフォーマンス

597, 598

中間結果 631

中国語 GB 18030 データ

処理 201

重複計算のグループ化 589

直接アクセス

直接指標付け 70

通貨記号

使用 62

複数の文字 62

ユーロ 63

16 進数リテラル 63

データ

受け渡し 517

グループ化 522

形式、数値タイプ 44

形式の変換 51

効率的な実行 587

数値 41

妥当性検査 54

非互換 54

分割 (UNSTRING) 102

命名 13

レコード・サイズ 13

連結 (STRING) 99

データ域、動的 280

データおよびプロシージャ名相互参照の

記述 340

データ型、COBOL と C/C++ の間の対応

511

データ記述記入項目 12

データ項目

可変位置 644

基本、定義 24

共通、サブプログラム・リンケージの

519

組み込み関数を使用した評価 114

組み込み関数を使用した変換 111

- データ項目 (続き)
 - グループ、定義 24
 - 最小項目または最大項目の検出 115
 - サイズ互換性 616
 - サブストリングの参照 106
 - 参照変更 106
 - 指標によるテーブル・エレメントの参照 69
 - 初期化の例 28
 - 数値 41
 - 調整は ADDR に依存 267
 - 分割 (UNSTRING) 102
 - 変換、大文字または小文字への 112
 - 未使用 298
 - 文字から数値への変換 113
 - 文字のカウント (INSPECT) 110
 - 文字の逆順 112
 - 文字の置換 (INSPECT) 110
 - 文字の変換 (INSPECT) 110
 - 連結 (STRING) 99
 - Java の型のコーディング 482
- データ項目の比較
 - 英数字
 - 照合シーケンスの影響 217
 - COLLSEQ の影響 274
 - オブジェクト参照 449
 - 国別
 - 英字、英数字、または DBCS と 204
 - 英数字グループと 205
 - 概要 202
 - 照合シーケンスの影響 219
 - 数値との 204
 - 2 つのオペランド 203
 - NCOLLSEQ の影響 203
 - ゾーン 10 進数および英数字、ZWB の影響 318
 - 日付フィールド 570
 - DBCS
 - 英数字グループと 219
 - 国別と 219
 - 照合シーケンスの影響 218
 - リテラル 207
 - COLLSEQ の影響 274
- データ項目の分割 (UNSTRING) 102
- データ項目の連結 (STRING) 99
- データ操作
 - 文字データ 99
- データ定義 346
- データ定義属性コード 346
- データの検査 (INSPECT) 110
- データ表現
 - 移植性 494
 - 影響を与えるコンパイラー・オプション 269
- データ名
 - 相互参照 349
 - MAP リスト内の 346
- テーブル
 - 値のロード 71
 - 値の割り当て 73
 - エレメント 65
 - エレメントのサブストリングの参照 107
 - エレメントの参照 69
 - 可変長
 - オーバーレイを防ぐ 646
 - 作成 77
 - 初期化 80
 - ロードの例 79
 - 行 67
 - 組み込み関数による処理 83
 - 効率のよいコーディング 592, 593
 - 参照変更 69
 - 指標、定義 69
 - 指標による参照の例 68
 - 初期化
 - エレメントのすべての出現 75
 - グループ・レベルの 74
 - それぞれの項目の個別の 74
 - INITIALIZE の使用 72
 - PERFORM VARYING の使用 96
 - ストライド計算 593
 - 説明 38
 - 添え字、定義 69
 - 添え字による参照の例 68
 - 多次元 66
 - 探索
 - 概要 80
 - 逐次 81
 - パフォーマンス 81
 - 2 進数 82
 - 定義 65
 - 同一エレメント仕様 592
 - 動的ロード 72
 - 配列との比較 38
 - 深さ 67
 - ループ 96
 - レコードの再定義 74
 - 列 65
 - 1 次元 65
 - 2 次元 67
 - 3 次元 67
 - OCCURS 節による定義 65
- テーブルの動的なロード 72
- 定義
 - ファイル
 - 概要 137
 - 例 138
 - SFS ファイル、例 149
- 定数
 - 計算 589
 - 形象、定義 27
 - データ項目 589
 - 定義 26
- ディレクトリー
 - エラー・リスト・ファイル 243
 - 共有ライブラリーのパスの指定 228
 - パスの追加 245
- 適合要件
 - INVOKE でのオブジェクト参照引き渡しの例 452
 - INVOKE の RETURNING 句 454
 - INVOKE の USING 句 451
- テキスト名相互参照、記述 340
- 出口モジュール
 - デバッグ 353
 - ライブラリー名の代わりに使用される場合 284
 - ロードおよび呼び出し 284
 - SYSADATA データ・セットのために呼び出される 286
 - SYSLIB の代わりに使用される場合 284
 - SYSPRINT の代わりに使用される場合 286
- テスト
 - 条件 95
 - 数値オペランド 90
 - データ 90
 - UPSI スイッチ 90
- 手続き部
 - インスタンス・メソッド 439
 - クライアント 446
 - サブプログラムでの 521
 - ステートメント
 - コンパイラー指示 21
 - 条件付き 19
 - 範囲区切り 20
 - 命令ステートメント 19
- 説明 17
- 用語 17
- RETURNING
 - パラメーターの戻し 18
 - メソッドでの使用 527
- USING
 - パラメーターの受け取り 18, 519
 - BY VALUE 521
- 鉄道線路構文図の読み方 xvi
- デバッグ
 - アセンブラー 354
 - 概要 331
 - コンパイラー・オプション
 - 概要 336
 - TEST に関する制約事項 334
 - THREAD に関する制約事項 334

デバッグ (続き)
シンボリック情報の生成 247
デバッガーの使用 341
バッチ機能のアクティブ化 326
メッセージ・オフセット情報を含む
353
ユーザー出口 353
ランタイム・オプション 334
CICS プログラム 369
COBOL 言語機能の使用 331
idebug コマンド 354
デバッグ、言語機能の
クラス・テスト 333
宣言 334
デバッグ行 334
デバッグ・ステートメント 334
範囲終了符号 332
ファイル状況キー 332
DISPLAY ステートメント 332
INITIALIZE ステートメント 333
SET ステートメント 333
WITH DEBUGGING MODE 節 334
統計組み込み関数 59
動詞、プログラムで使用される 342
動詞相互参照リスト
説明 342
動的呼び出し
CICS のもとで 366
DB2 API には使用できない 357
動的ロード、要件 228
特殊機能指定 7
特殊レジスター
組み込み関数の引数 57
ADDRESS OF
サイズは ADDR に依存 267
32 ビット・モードおよび 64 ビット
モード 617
CALL ステートメントでの使用
518
JNIEnvPtr
サイズは ADDR に依存 267
32 ビット・モードおよび 64 ビット
モード 617
JNI 呼び出し可能サービスのための
使用 478
LENGTH OF 118, 518
RETURN-CODE 526
SORT-RETURN
ソートまたはマージの終了 168
ソートまたはマージの成功の判断
164
WHEN-COMPILED 118
XML-CODE 391, 393
XML-EVENT 391, 392
XML-NTEXT 391, 393
XML-TEXT 391, 393

特記事項 831
トップダウン・プログラミング
回避するための構成 588

[ナ行]

内部浮動小数点データ
(COMP-1、COMP-2) 48
内部ブリッジ
日付処理用 566
利点 564
例 567
内部ブリッジによる日付処理の利点 564
長さを検出する、データ項目の 118
名前宣言
探索 506
名前の有効範囲
グローバル 506
実行単位 547
プログラム起動インスタンス 548
マルチスレッド化の影響 547
要約表 548
ローカル 506
二分探索
説明 82
例 83
入出力
エラー後のロジック・フロー 171
エラーの検査 174
概要 121
コーディング
概要 137
例 138
処理エラー
VSAM ファイル 171
GUI アプリケーション 228
入出力コーディング
誤った索引キーの検出 177
エラー処理技法 171
状況コードの検査
概要 175
例 176
正常操作の検査 174
AT END (ファイルの終わり) 句 172
EXCEPTION/ERROR 宣言 173
入力
概要 132
ファイルから 121
入出力プロシージャ
コーディング 158
制限 160
例 163
RELEASE または RELEASE FROM
が必要 159
ヌル終了ストリング
処理 522

ヌル終了ストリング (続き)
取り扱い 105
例 106
ネイティブ形式
メソッドの引数用 619
CHAR オプション 271
FLOAT オプション 289
JNI サービスの引数用 619
-host オプションのコマンド行引数への
影響 532
ネストされた COPY ステートメント
285, 601
ネストされた IF ステートメント
コーディング 86
望ましい EVALUATE ステートメント
86
CONTINUE ステートメント 86
NULL ブランチを伴う 85
ネストされた区切り範囲ステートメント
22
ネストされた組み込み関数 57
ネストされたプログラム
移動、制御の 503
指針 503
説明 503
名前の有効範囲 505
マップ 342, 348
呼び出し 503
ネストされたプログラムの統合 596
ネストされたプログラム・マップ
説明 342
例 348
ネスト・レベル
ステートメント 345
プログラム 345, 349
年先行型日付フィールド 570
年単独型日付フィールド 570
年のウィンドウ操作
サポートされない場合 572
制御方法 579
利点 564
MLE アプローチ 565
年フィールド拡張 567
年末尾型日付フィールド 570

[ハ行]

バイト・オーダー・マークが生成されない
412
配列
COBOL 38
Java
宣言する 484
取り扱い 485

パス名
 カタログおよびヘルプ・ファイルの指定 228
 コピーブックの検索に使用 245, 319
 実行可能プログラムに対する指定 228
 複数、指定 226, 319
 優先順位 226
 ライブラリー・テキスト 226, 319
 ランタイム・ライブラリーに対する指定 228
 ロケール情報データベースの指定 228
 LIB コンパイラー・オプションでの指定 291
 パック 10 進数データ項目
 効率的な使用 48, 591
 サイン表記 53
 説明 48
 同義語 45
 日付フィールドの起こりうる問題 583
 バッチ・コンパイル 302
 バッチ・デバッグ、アクティブ化 326
 パフォーマンス
 一貫性のあるデータ型 591
 コーディング 587
 コンパイラー・オプション
 DYNAM 597
 FLOAT 496
 OPTIMIZE 595, 597
 SSRANGE 597
 TEST 597
 THREAD 312, 597
 TRUNC 312, 597
 コンパイラー・オプションの影響 597
 最適化プログラム
 概要 596
 算術式 591
 算術評価 590
 指数 591
 実行時の考慮事項 587
 データの使用 590
 テーブル探索
 逐次探索の改善 81
 二分と逐次の比較 81
 テーブルのコーディング 592
 テーブルの処理 593
 トレース・ファイルの作成 246
 プログラミング・スタイル 588
 変数添え字のデータ形式 69
 ライン外 PERFORM とインラインの比較 95
 ワークシート 600
 CICS 環境 587
 EVALUATE 内の WHEN 句の順序 88
 OCCURS DEPENDING ON 593
 SFS に対する環境変数 228

パフォーマンス (続き)
 SFS ファイルのための向上
 強制操作の抑制 153
 クライアント側のキャッシュの使用
 可能化 151
 変更の保存回数の削減 153
 パラメーター
 メインプログラムにおける 532
 呼び出し先プログラムの中での記述 519
 パラメーター・リスト
 ADEXIT 用 286
 INEXIT を使用した場合のアドレス 284
 PRTEXIT 用 286
 範囲区切りステートメント
 説明 20
 ネストされた 22
 範囲終了符号
 暗黙的な 21
 デバッグでの補助 332
 明示的 20, 21
 範囲終了符号としてのピリオド 21
 汎用オブジェクト参照 448
 ヒープ・スペース、cob2 コマンドを使用した割り振り 249
 ヒープ・スペースの割り振り 249
 比較条件 90
 指数
 メインプログラムに対する 532
 呼び出し側プログラムでの記述 519
 BY VALUE で渡す 519
 COBOL から C への受け渡し、例 511
 COBOL から C++ への受け渡し、例 514
 COBOL と C の間の受け渡し 513
 COBOL と C の間の受け渡し、例 512
 OMITTED の指定 520
 OMITTED 指数に関するテスト 520
 指数として渡すデータのグループ化 522
 指数を省略するための OMITTED 句 520
 ピクチャー・ストリング
 日時サービスおよび 607
 ビッグ・エンディアン
 国別文字の表現 499
 整数の表現 499
 ビッグ・エンディアン、リトル・エンディアンへの変換 188
 日付演算
 組み込み関数 38
 日付および時刻
 組み込み関数 650

日付および時刻 (続き)
 形式
 整数から秒への変換 (CEEISEC) 673
 タイム・スタンプから秒数への変換 (CEESECS) 685
 秒から整数への変換 (CEESECI) 681
 秒から文字タイム・スタンプへの変換 (CEEDATM) 659
 文字形式から COBOL 整数形式への変換 (CEECBLDY) 651
 文字形式からリリアン形式への変換 (CEEDAYS) 663
 リリアン形式から文字形式への変換 (CEEDATE) 655
 サービス
 概要 649
 計算の実行 604
 条件処理 604
 条件フィードバック 606
 ピクチャー・ストリング 607
 フィードバック・コード 604
 戻りコード 604
 リスト 649
 例 605
 CALL ステートメントを使用した呼び出し 603
 CEECBLDY: 日付から COBOL 整数形式への変換 651
 CEEDATE: リリアン日付から文字形式への変換 655
 CEEDATM: 秒から文字タイム・スタンプへの変換 659
 CEEDAYS: 日付からリリアン形式への変換 663
 CEEDYWK: リリアン日付からの曜日の計算 666
 CEEGMTO: グリニッジ標準時からのオフセットの取得 671
 CEEGMT: 現在のグリニッジ標準時の取得 669
 CEEISEC: 整数から秒への変換 673
 CEEOCT: 現在の現地時間の取得 676
 CEEQCEN: 世紀ウィンドウの照会 678
 CEESCEN: 世紀ウィンドウの設定 679
 CEESECI: 秒から整数への変換 681
 CEESECS: タイム・スタンプから秒数への変換 685
 CEEUTC: 協定世界時の取得 689

- 日付および時刻 (続き)
 - サービス (続き)
 - RETURN-CODE 特殊レジスタ— 604
 - 日時の取得 (CEELOCT) 676
 - syntax 685
 - 日付算術演算 577
 - 日付情報、形式 228
 - 日付操作
 - コンパイルの日付の検索 118
 - 日付のウィンドウ操作
 - サポートされない場合 572
 - 制御方法 579
 - 利点 564
 - 例 566, 572
 - MLE アプローチ 565
 - 日付の比較 570
 - 日付フィールド拡張
 - 説明 567
 - 利点 564
 - 日付フィールドの潜在的な問題 583
 - 評価、データ項目の内容の
 - 組み込み関数 114
 - クラス・テスト
 - 概要 90
 - 数値の 54
 - INSPECT ステートメント 110
 - 評価の順序
 - コンパイラー・オプション 265
 - 算術演算子 55, 633
 - 表現
 - データ 54
 - 符号 53
 - 表示浮動小数点データ (USAGE DISPLAY) 46
 - 標準 COBOL 85
 - 定義 xv
 - ファイル
 - オープン 140
 - オペレーティング・システムに対する
 - 識別 10
 - 外部 528
 - 環境変数を使用したアクセス 228
 - クラスター 126
 - 識別 121
 - 使用法の説明 11
 - 小容量形式
 - 互換 131
 - 定義 131
 - 処理
 - Encina SFS ファイル 124
 - RSD ファイル 124
 - SFS (Encina) ファイル 124
 - STL ファイル 124
 - VSAM ファイル 124
 - 説明 12
- ファイル (続き)
 - 大容量形式 130
 - 名前の変更 11
 - ファイル位置標識 140, 143
 - ファイル編成の比較 133
 - 複数、コンパイル 236
 - ブランクによるレコードの埋め込み 130
 - プログラム・ファイルを外部ファイルに関連付ける 7
 - 要約 131
 - リンカー
 - インポート 251
 - エクスポート 251
 - library 251
 - レコードの更新 147
 - レコードの削除 146
 - レコードの置換 145
 - レコードの追加 145
 - レコードの読み取り 143
 - 32 ビット・モードおよび 64 ビット・モード 618
 - COBOL コーディング
 - 概要 137
 - 例 138
 - Encina SFS
 - アクセス 367
 - 識別 121
 - Encina SFS へのアクセス 148
 - FILEMODE ランタイム・オプション、影響 327
 - FILESYS ランタイム・オプション、影響 328
 - RSD 121
 - SFS (Encina)
 - アクセス 367
 - 識別 121
 - STL
 - 識別 121
 - 小容量形式 131
 - 大容量形式 130
 - FILEMODE によって影響を受けるサイズ 327
 - TRAP ランタイム・オプションの影響 329
 - VSAM
 - 識別 121
 - 小容量形式 131
 - 大容量形式 130
 - FILEMODE によって影響を受けるサイズ 327
 - ファイル位置標識 140, 143
 - ファイル記述 (FD) 記入項目 13
 - ファイル終了句 (AT END) 172
 - ファイル状況キー
 - エラー処理 332
- ファイル状況キー (続き)
 - 状況コードと一緒に使用
 - 概要 175
 - 例 176
 - 正常 OPEN かどうかの検査 174, 175
 - 設定 136
 - 入出力エラーの検査 174
 - ファイル状況コード
 - 使用 171
 - 02 144
 - 39 176
 - 92 146
 - ファイルのオープン
 - 概要 140
 - 環境変数を使用 228
 - STL 形式と VSAM 形式の影響 130
 - ファイルへのレコードの追加
 - 概要 145
 - 順次 145
 - ランダムまたは動的に 145
 - STL 形式と VSAM 形式の影響 131
 - ファイル変換
 - 2000 年言語拡張での 568
 - ファイル編成
 - 概要 132
 - 行順次 134
 - 索引付き 134
 - 順次 133
 - 相対 134
 - ファイル・アクセス・モード
 - 行順次ファイル用 134
 - 索引付きファイル用 134
 - 順次 135
 - 順次ファイル用 133
 - 相対ファイル用 134
 - 動的 135
 - 要約テーブル 133
 - ランダム 135
 - ファイル・サフィックス
 - エラー・メッセージ・リスト 243
 - ファイル・システム・サポート
 - 要約 131
 - Encina SFS 328
 - FILEMODE ランタイム・オプションの使用 327
 - FILESYS ランタイム・オプションの使用 328
 - RSD 328
 - SFS (Encina) 328
 - STL 328
 - VSAM 328
 - ファクトリー定義、コーディング 462
 - ファクトリー・セクション、定義 462
 - ファクトリー・データ
 - それをアクセス可能にする 464
 - 定義 427, 463

- ファクトリー・データ (続き)
 - private 464
- ファクトリー・メソッド
 - 隠蔽 466
 - 定義 427, 464
 - プロシージャー・プログラムのラップ
に使用 473
 - 呼び出し 466
- ファクトリー・メソッドの隠蔽 466
- フィードバック・トークン
 - 日時サービスおよび 606
- 深さ、テーブルの 67
- 複合 OCCURS DEPENDING ON
 - 可変位置グループ 644
 - 可変位置データ項目 644
 - 基本形式 643
 - 複合 ODO 項目 643
- 複数の通貨記号
 - 使用 62
 - 例 63
- 含まれているプログラムの統合 596
- 符号条件
 - 数値オペランドの符号のテスト 90
 - 日付処理での使用 576
- 物理レコード 13
- 浮動小数点演算
 - 指数 639
 - 比較 61
 - 評価 60
 - 例の評価 61
- 浮動小数点データ
 - 移植性 496
 - 外部 46
 - 固定小数点と浮動小数点との間の変換
52
 - 使用計画 590
 - 中間結果 639
 - 内部
 - 形式 48
 - パフォーマンスに関するヒント
591
 - パフォーマンスの考慮事項 496
 - 変換と精度 52
 - フラグおよびスイッチ 91
 - プラットフォームの違い 497
 - ブランクによるレコードの埋め込み 130
 - ブランチ、暗黙の 94
 - プリンター・ファイル 134
 - プログラム
 - 決定
 - スイッチおよびフラグ 91
 - ループ 95
 - EVALUATE ステートメント 85
 - IF ステートメント 85
 - PERFORM ステートメント 95
 - 構造体 5
- プログラム (続き)
 - サブプログラム 501
 - 診断 344
 - 制約 587
 - 属性コード 349
 - 統計 344
 - ネスト・レベル 345
 - メイン 501
- プログラム、実行 254
- プログラムの実行 254
- プログラムのドキュメンテーション 7
- プログラム名
 - 指定 5
 - 相互参照 350
 - 大/小文字の処理 299
- プロシージャーおよびデータ名相互参照の
記述 340
- プロシージャー・ポインター・データ項目
 - サイズは ADDR に依存 267
 - 使用 526
 - 定義 526
 - 呼び出し可能サービスへのパラメータ
ーの受け渡し 526
 - 32 ビット・モードおよび 64 ビット・
モード 617
 - SET ステートメントと 526
- プロセス
 - 終了 510
 - 定義 545
- プロファイル作成のサポート、PROFILE
オプション 301
- プロファイル・ファイル、環境変数の設定
225
- 文、定義 18
- 文書エンコード宣言 399
- 分離型の CICS 変換プログラム
 - 制限 369
- 分離符号
 - 移植性 42
 - 印刷 42
 - 符号付き国別 10 進数に必要 42
- ベース・ロケーター 347
- ヘルプ・ファイル
 - 各国語の設定 228
 - パス名の指定 228
- 変換、データ項目の
 - 英数字へ
 - DISPLAY による 37
 - DISPLAY-OF を使用した 199
 - 大文字または小文字への
 - 組み込み関数を使用した 112
 - INSPECT による 111
 - 国別から UTF-8 への 201
 - 国別から中国語 GB 18030 へ 201
 - 国別と
 - 中国語 GB 18030 から 201
- 変換、データ項目の (続き)
 - 国別と (続き)
 - ACCEPT による 36
 - MOVE を使用した 198
 - NATIONAL-OF を使用した 198
 - UTF-8 から 201
 - 組み込み関数を使用した 111
 - コード・ページ間の 114
 - 精度 52
 - データ・フォーマット間の 51
 - 文字の逆順 112
 - INSPECT による 110
 - INTEGER、INTEGER-PART による整
数への 109
 - NUMVAL、NUMVAL-C による数値へ
の 113
- 変換、ファイルの拡張日付形式への例
568
- 変換、文字形式からリリアン日付への
(CEEDAYS) 663
- 変換、リリアン日付から文字形式への
(CEEDATE) 655
- 変換、CICS を COBOL へ 363
- 変換、COBOL データから XML への
概要 407
- 例 413
- 変更
 - ソース・リストのタイトル 7
 - ファイル名 11
 - 文字から数値への 113
- 変数
 - 参照修飾子としての 107
 - 定義 23
- 変数、環境
 - アクセス 226
 - コンパイラー 226
 - 設定
 - 概要 225
 - コマンド・シェル内 225
 - プログラム内 226
 - ロケール (locale) 212
 - .profile 内 225
 - 設定およびアクセスの例 234
 - 定義 225
 - 優先順位 226
 - ランタイム 228
 - 割り当て名 228
 - CLASSPATH 228
 - COBCPYEXT 226
 - COBJVMINIOPTIONS 228
 - COBLSTDIR 226
 - COBMSGs 228
 - COBOPT 226
 - COBPATH
 - 説明 228
 - CICS 動的呼び出し 366

変数、環境 (続き)

COBRTOPT 228
EBCDIC_CODEPAGE 228
ENCINA_CDS_ROOT 228
ENCINA_SFS_DATA_VOLUME 228
ENCINA_SFS_INDEX_VOLUME 228
ENCINA_VSAM_AUTO_FLUSH 228
ENCINA_VSAM_CACHE 228
LANG 228
LC_ALL 228
LC_COLLATE 228
LC_CTYPE 228
LC_MESSAGES 228
LC_TIME 228
LIBPATH 228
library-name 226, 319
LOCPATH 228
NLSPATH 228
PATH 228
SYSIN, SYSIPT, SYSOUT,
SYSLIST, SYSLST, CONSOLE,
SYSPUNCH, SYSPCH 228
SYSLIB 226
text-name 226, 319
TMP 228
TZ 228

ポインター・データ項目

アドレスの受け渡し 523
アドレスの増分 523
サイズは ADDR に依存 267
説明 38
チェーン・リストの処理 523
チェーン・リストの処理に使用 523
32 ビット・モードおよび 64 ビット・
モード 617
NULL 値 523

[マ行]

マージ

エラー番号
考えられる値のリスト 165
iwzGetSortErrno での取得 165
完了コード 164
キー
概要 156
定義 162
デフォルト 218
基準 162
終了 168
診断メッセージ 164
正常終了の判別 164
説明 155
代替照合シーケンス 163
ファイルの説明 157
プロセス 156

マッピング、DATA DIVISION 項目の

342
マルチスレッド化
概要 545
言語エレメントの有効範囲
実行単位 547
プログラム起動インスタンス 548
要約表 548
再帰 549
事前初期設定 549
制御転送 549
制約 550
データ・セクションの選択
OO クライアントにおいて 448
プログラムの終了 550
戻りコードでの影響 516
用語 545
リソースへのアクセスの同期化 551
例 551
COBOL プログラムの準備 545
THREAD コンパイラー・オプション
いつ選択するか 549
制限 311
マルチスレッド環境での実行 311
マルチタスキング、定義 546
ミドルウェア・サブシステム、32 ビッ
ト・モードおよび 64 ビット・モード
618
矛盾するコンパイラー・オプション 265
明示範囲終了符号 21
命名
プログラム 5
命令ステートメントのリスト 19
メイン入り口
cob2 による指定 246
メインプログラム
サブプログラム 501
に対する引数 532
cob2 による指定 246
メソッド
インスタンス 436, 461
オーバーライド 440, 466
から値を戻す 440
コンストラクター 464
シグニチャー 437
スーパークラスの呼び出し 454
多重定義 441
ファクトリー 464
ファクトリーの隠蔽 466
呼び出し 450, 466
渡された引数の取得 440
Java アクセス制御 482
PROCEDURE DIVISION
RETURNING 527
メッセージ
オフセット情報 353

メッセージ (続き)

各国語サポート 213
各国語の設定 228
コンパイラー
作成する重大度レベルの判別 286
重大度レベル 242
ソース・リストへの組み込み 338
日付関連 581
フラグを立てる重大度の選択 338
リストの生成 243
2000 年言語拡張 581
コンパイラーの指示 243
ファイルの指定 228
ランタイム
形式 781
不完全または省略 254
リスト 781
dspcat コマンドを使用したリスト
表示 781
TRAP(OFF) の副次作用 329
メッセージ・カタログ
パス名の指定 228
目標、2000 年言語拡張の 563
文字セット、定義 188
文字タイム・スタンプ
リリアン日付の変換 (CEEDATM) 659
例 660
リリアン秒への変換 (CEESECS) 685
例 685
COBOL 整数形式への変換
(CEECBLDY) 651
例 653
文字の逆順 112
モジュール、出口
ロードおよび呼び出し 284
戻りコード
オペレーティング・システムに制御権
が戻される時 516
コンパイラー 242
日時サービスからのフィードバック・
コード 604
ファイル
概要 175
例 176
DB2 SQL ステートメントからの 360
RETURN-CODE 特殊レジスター 516,
527, 604

[ヤ行]

ユーザー定義の条件 90
ユーザー出口作業域 283
ユーロ通貨記号 63
優先順位
算術演算子 55, 633
用語集 835

曜日、CEEDYWK による計算 666
 呼び出し
 インスタンス・メソッド 450
 オーバーフロー条件 178
 コンパイラーおよびリンカー 236
 再帰的 515
 静的 506
 データの受け渡し 517
 動的 506
 日時サービス 603
 日時サービスに対する 603
 パラメーターの受け取り 519
 引数の受け渡し 519
 ファクトリーまたは静的メソッド 466
 例外条件 178
 CICS での COBOL と C/C++ の間
 367
 JNI サービスへの 477
 LINKAGE SECTION 520
 OMITTED 引数 520
 呼び出しインターフェース規約
 CALLINT で指示 270
 ODBC を使用する場合 384
 呼び出し可能サービス
 CEECBLDY: 日付から COBOL 整数形
 式への変換 651
 CEEDATE: リリアン日付から文字形式
 への変換 655
 CEEDATM: 秒から文字タイム・スタ
 ンプへの変換 659
 CEEDAYS: 日付からリリアン形式へ
 の変換 663
 CEEDYWK: リリアン日付からの曜日
 の計算 666
 CEEGMTO: グリニッジ標準時からの
 オフセットの取得 671
 CEEGMT: 現在のグリニッジ標準時の
 取得 669
 CEEISEC: 整数から秒への変換 673
 CEEOCT: 現在の現地時間の取得
 676
 CEEQCEN: 世紀ウィンドウの照会
 678
 CEESCEN: 世紀ウィンドウの設定
 679
 CEESECI: 秒から整数への変換 681
 CEESECS: タイム・スタンプから秒数
 への変換 685
 CEEUTC: 協定世界時の取得 689
 IZEDT4: 4 桁年号を使用した現在日
 付の取得 689
 _iwbzGetCCSID: コード・ページ ID か
 ら CCSID への変換 221
 _iwbzGetLocaleCP: ロケールおよび
 EBCDIC コード・ページ値の取得
 220

読み取り、ファイルからのレコードの
 順次 143
 動的 143
 ランダム 143
 読み取りキャッシュ 151

[ラ行]

ライブラリーのコンポーネント、再配布
 254
 ライブラリーのコンポーネントの再配布
 254
 ライブラリー名、使用されない場合 284
 ライブラリー・テキスト
 パスの指定 226, 319
 ライブラリー・ファイル 251
 ライン外の PERFORM 94
 ラッパー、定義 473
 ラッピング、プロシージャ指向プログラ
 ムの 473
 ランタイム環境、事前初期設定
 概要 555
 例 557
 ランタイム・オプション
 概要 325
 指定 228
 CHECK 325
 CHECK(OFF)
 パフォーマンスの考慮事項 597
 CICS の場合 368
 DEBUG 326, 334
 ERRCOUNT 326
 FILEMODE
 互換 131
 説明 327
 FILESYS 328
 TRAP
 説明 328
 ON SIZE ERROR 170
 UPSI 329
 ランタイム・メッセージ
 各国語の設定 228
 形式 781
 言語用のファイルの設定 228
 不完全または省略 254
 リスト 781
 dspcat コマンドを使用したリスト表示
 781
 ランタイム・ライブラリー、パス名の指定
 228
 ランタイム・ロード、制約事項 541
 リスト
 組み込みエラー・メッセージ 338
 短縮リストの生成 342
 データおよびプロシージャ名相互参
 照 340

リスト (続き)
 テキスト名相互参照 340
 テキスト名のソート済み相互参照 350
 プログラム名のソート済み相互参照
 350
 ユーザー提供の行番号 343
 MAP 出力で使用される用語 347
 リストのヘッダー 7
 リテラル
 英数字
 制御文字 26
 説明 26
 マルチバイトの内容での 206
 国別
 使用 190
 説明 26
 使用 26
 数値 26
 定義 26
 16 進数
 使用 190
 DBCS
 最大長 207
 使用 206
 説明 26
 リトル・エンディアン
 国別文字の表現 499
 整数の表現 499
 リトル・エンディアン、ビッグ・エンディ
 アンへの変換 188
 リリアン日
 現在の現地日時の取得
 (CEEOCT) 676
 日付から COBOL 整数形式への変換
 (CEECBLDY) 651
 日付の変換 (CEEDAYS) 663
 文字形式への変換 (CEEDATE) 655
 曜日の計算 (CEEDYWK) 666
 CEESECI への入力としての使用 683
 GMT の取得 (CEEGMT) 669
 output_seconds の変換 (CEEISEC) 673
 リンカー
 エラー 253
 オプションの指定 248
 共用ライブラリーへの参照の解決 536
 検索規則 252
 ファイル
 インポート 251
 エクスポート 251
 library 251
 ファイルのデフォルト 253
 呼び出し 236, 247
 リンク
 静的 535
 プログラム 247
 例 250

リンクする、オブジェクト指向アプリケーションを
例 259
cob2_j コマンド 258
リンク・リスト処理、例 523
リンケージ、データ 511
リンケージ規約
コンパイラー・オプション
CALLINT 270
コンパイラー・ディレクティブ
CALLINT 319
ループ
コーディング 94
条件付き 96
テーブル内 96
明示的に指定した回数だけ実行される
95
DO 95
例
CEECELDY: 日付から COBOL 整数形式への変換 653
CEEDATE: リリアン日付から文字形式への変換 656
CEEDATM: 秒から文字形式への変換 660
CEEDAYS: 日付からリリアン形式への変換 665
CEEDYWK: リリアン日付からの曜日の計算 667
CEEGMTO: グリニッジ標準時からのオフセットの取得 672
CEEGMT: 現在の GMT の取得 670
CEEISEC: 整数から秒への変換 675
CEELOCT: 現在の現地時間の取得 677
CEEQCEN: 世紀ウィンドウの照会 678
CEESCEN: 世紀ウィンドウの設定 680
CEESECI: 秒から整数への変換 683
CEESECS: タイム・スタンプから秒数への変換 687
IGZEDT4: 4 桁年号を使用した現在日付の取得 689
_iwezGetCCSID: コード・ページ ID から CCSID への変換 221
_iwezGetLocaleCP: ロケールおよび EBCDIC コード・ページ値の取得 221
例外、代行受信 328
例外条件
CALL 178
XML GENERATE 412
XML PARSE 403
例外処理
Java との 479

レコード
形式 132
説明 12
TRAP ランタイム・オプションの影響 329
レコードを埋め込むための awk コマンド 130
列、テーブルの 65
レベル 88 項目
ウィンドウ表示日付フィールド用 573
条件式 90
スイッチおよびフラグ 91
スイッチをオフに設定する例 93
スイッチをオンに設定する例 93
制約事項 573
単一値のテストの例 92
複数値のテストの例 92
レベル番号 346
ローカル参照、グローバルへの変換 455
ローカル名 506
ロケール (locale)
アクセス 220
値の構文 212
およびメッセージ 213
国/地域別情報の定義 209
サポートされる値 213
指定 228
照会 220
定義 209
デフォルト 212
リストに表示される 340, 344
ロケール・ベースの照合 216
COLLSEQ コンパイラー・オプションの影響 217
PROGRAM COLLATING SEQUENCE の影響 217
ロケール情報データベース
検索パス名の指定 228

[ワ行]

ワークステーションおよびワークステーション COBOL
ホストとの違い 619
割り当て名環境変数 228

[数字]

16 進数
移植性 496
16 進数リテラル
国別
使用 190
説明 26
通貨符号として 63

2 桁の年号
有効な値 607
100 年範囲内での照会 (CEEQCEN) 678
例 678
100 年範囲内の設定 (CEESCEN) 679
例 680
2 進数データ、データ表現 269
2 進数データ項目
一般的な説明 47
効率的な使用 47, 590
中間結果 636
同義語 45
2000 年言語拡張
概念 562
仮定による世紀ウィンドウ 574
原則 563
互換性のある日付 570
日付のウィンドウ操作 561
非日付 575
目標 563
DATEPROC コンパイラー・オプション 278
YEARWINDOW コンパイラー・オプション 317
3 桁の年号 607
4 桁の年号 607
64 ビット・モード
言語間通信 508
制限
概要 268
要約 618
64 ビット COBOL プログラムと 32 ビット COBOL プログラムは混用できない 507
CICS 364
DB2 618
SFS ファイル 125
ファイル・システム要約 131
プログラミングの要件 268
32 ビット・モードからの移行 617
ADDR コンパイラー・オプション 266
-q64 コンパイラー・オプション
説明 247
64 ビット Java SDK の選択 262

A

ACCEPT ステートメント
入力データの割り当て 35
CICS のもとで 365
GUI アプリケーションでの使用 228
ACCEPT ステートメント、使用される環境変数 228
ADATA コンパイラー・オプション 266

ADDR コンパイラー・オプション 266
ADDR に依存する調整 267
ADDRESS OF 特殊レジスター
サイズは ADDR に依存 267
32 ビット・モードおよび 64 ビット・
モード 617
CALL ステートメントでの使用 518
AIX、への移植 499
ALL 添え字
関数引数としてのテーブル・エレメン
ト 57
テーブル・エレメントの反復処理 84
例 84
ALPHABET 節による照合シーケンスの設
定 8
ANNUITY 組み込み関数 59
APOST コンパイラー・オプション 301
ARITH コンパイラー・オプション
説明 268
ASCII
マルチバイトの移植性 496
EBCDIC への変換 114
SBCS の移植性 495
XML 文書でサポートされるコード・
ページ 397
ASSIGN 節 10
AT END (ファイルの終わり) 172

B

Base クラス
java.lang.Object に相当 433
java.lang.Object のために使用 432
BINARY コンパイラー・オプション 269
BLANK WHEN ZERO 節
数字編集データを含んだ例 43
数値データ用にコーディングされる
190
BY CONTENT 517
BY REFERENCE 517
BY VALUE
制限 519
説明 517
有効なデータ型 519

C

C
COBOL から呼び出される関数、例
511
COBOL を呼び出す関数、例 512,
513
CALL ID
共有ライブラリーの呼び出し例 540

CALL ステートメント
エラー処理に関する 178
オーバーフロー条件 178
その中でのプログラム名の処理 299
日時サービスを呼び出すための 603
例外条件 178
BY CONTENT 517
BY REFERENCE 517
BY VALUE
制限 519
説明 517
CALL ID 507
CALL literal 507
CALLINT オプションの影響 270
DYNAM の場合 280
ON EXCEPTION を指定した 178
ON OVERFLOW を指定した 20, 178
RETURNING 527
USING 519
CALLINT コンパイラー・オプション
説明 270
CALLINT ステートメント
説明 319
CANCEL ステートメント
その中でのプログラム名の処理 299
CBL ステートメント
コンパイラー・オプションの指定 238
説明 319
CCSID
構文解析対象の XML 文書 390
定義 188
PARSE ステートメント 390
XML 文書での矛盾 404
XML 文書の 397, 398
CEECBLDY: 日付から COBOL 整数形式
への変換
例 651
syntax 651
CEEDATE: リリアン日付から文字形式へ
の変換
出力例の表 658
例 656
syntax 655
CEEDATM: 秒から文字タイム・スタンプ
への変換
出力例の表 662
例 660
CEESECI 681
syntax 659
CEEDAYS: 日付からリリアン形式への変
換
例 665
syntax 663
CEEDYWK: リリアン日付からの曜日の計
算
例 667

CEEDYWK: リリアン日付からの曜日の計
算 (続き)
syntax 666
CEEGMTO: グリニッジ標準時からのオフ
セットの取得
例 672
syntax 671
CEEGMT: 現在のグリニッジ標準時の取得
例 670
syntax 669
CEEISEC: 整数から秒への変換
例 675
syntax 673
CEELOCT: 現在の現地時間の取得
例 677
syntax 676
CEEQCEN: 世紀ウィンドウの照会
例 678
syntax 678
CEESCEN: 世紀ウィンドウの設定
例 680
syntax 679
CEESECI: 秒から整数への変換
例 683
syntax 681
CEESECS: タイム・スタンプから秒数へ
の変換
例 687
syntax 685
CHAR 組み込み関数の例 115
CHAR コンパイラー・オプション
説明 271
マルチバイトの移植性 496
SBCS の移植性 495
XML 文書エンコードに対する影響
398
CHECK ランタイム・オプション 325
参照変更 107
パフォーマンスの考慮事項 597
CICS
移植性に関する考慮事項 364
共用ライブラリー 366
組み込みの変換プログラム
概要 369
利点 369
コンパイラー・オプション 368
システム日付の取得 365
制限
アプリケーションは 32 ビットでな
ければならない 364
オブジェクト指向プログラム 364,
427
事前初期設定 555
ネストされたプログラム 364
分離型の変換プログラム 369

- CICS (続き)
 - 制限 (続き)
 - DYNAM コンパイラー・オプション 280
 - 動的呼び出し 366
 - 動的呼び出し用の DFHCOMMAREA パラメーター 366
 - 動的呼び出し用の DFHEIBLK パラメーター 366
 - プログラムのコーディング 364
 - プログラムのデバッグ 369
 - 分離型の変換プログラム
 - 制限 369
 - ホスト・データ・フォーマットはサポートされない 365
 - ランタイム・オプション 368
 - CICS 以外のアプリケーションからの CICS ファイルへのアクセス 367
 - COBOL と C/C++ の間での呼び出し 367
 - COBOL に関係のあるコマンド 363
 - COBOL プログラムの開発 363
 - TRAP ランタイム・オプションの影響 329
- CICS コンパイラー・オプション
 - 組み込みの変換プログラムを使用可能にする 369
 - サブオプションの指定 274
 - 説明 273
 - マルチオプションの相互作用 265
 - cicstcl -p コマンドの影響 369
- cicsmap コマンド 363
- cicstcl コマンド 363
 - 組み込みの変換プログラム用の -p フラグ 369
- cicsterm コマンド 363
- CLASSPATH 環境変数
 - 説明 228
 - Java クラスの場所の指定 260
- cob2 コマンド
 - オプション
 - 説明 245
 - デフォルトの変更 239
 - ADDR の -q 省略形 267
 - host 532
 - # 239
 - 構成ファイルの変更 239
 - 使用されるスタンザ 240
 - 使用例、コンパイル 237
 - 使用例、リンク 250
 - 説明 236
 - command-line argument format 532
- cob2 スタンザ 240
- cob2.cfg 構成ファイル 239
- cob2_j コマンド
 - オブジェクト指向アプリケーションで使用する 236
 - オブジェクト指向アプリケーションをコンパイルする場合 257
 - オブジェクト指向アプリケーションをリンクする場合 258
 - オプション
 - 説明 245
 - ADDR の -q 省略形 267
 - host 619
 - q32 は 32 ビット SDK を選択 262
 - q64 は 64 ビット SDK を選択 262
 - 使用されるスタンザ 239
 - 説明 236
 - command-line argument format 532
- cob2_j スタンザ 239
- cob2_r コマンド
 - オプション 245
 - 使用されるスタンザ 240
 - スレッド化の例 551
 - スレッド化または CICS アプリケーションのための 236
 - 説明 236
- cob2_r スタンザ 240
- COBCPYEXT 環境変数 226
- COBJVMINIOPTIONS 環境変数
 - 説明 228
 - JVM オプションの指定 261
- COBLSTDIR 環境変数 226
- COBMSGS 環境変数 228
- COBOL
 - オブジェクト指向
 - コンパイル 257
 - 実行 260
 - リンク 258
 - および C/C++ 508
 - データ型、C/C++ との対応 511
 - C 関数によって呼び出される、例 512, 513
 - C 関数の呼び出し、例 511
 - C++ 関数の呼び出し、例 514
 - Java
 - 間の通信 477
 - アプリケーションの構造化 473
 - 互換 615
 - コンパイル 257
 - 実行 260
 - リンク 258
- COBOL for AIX
 - アクセシビリティ xxi
 - 互換 615
 - ランタイム・メッセージ 781
- COBOL 環境の事前初期設定
 - 概要 555
 - 終了 556
 - 初期化 555
 - マルチスレッド化 549
 - 例 557
 - CICS での制約事項 555
 - C/C++ プログラム 509
- COBOL クライアント
 - オブジェクト参照引き渡しの例 452
 - 例 467
- COBOL 用語 23
- COBOPT 環境変数 226
- COBPATH 環境変数
 - 説明 228
 - CICS 動的呼び出し 366
- COBRTOPT 環境変数 228
- COLLATING SEQUENCE 句
 - 移植性に関する考慮事項 495
 - 国別キーに適用されない 162
 - ソートおよびマージ・キーへの影響 218
- PROGRAM COLLATING SEQUENCE 節のオーバーライド 8, 163
- SORT または MERGE での使用 163
- COLLSEQ コンパイラー・オプション
 - 移植性に関する考慮事項 495
 - 英数字照合シーケンスへの影響 216
 - 説明 274
 - DBCS 照合シーケンスへの影響 218
- COMMON 属性 6, 503
- COMP (COMPUTATIONAL) 47
- COMPILE コンパイラー・オプション
 - 構文エラーを見つけるための NOCOMPILE の使用 336
 - 説明 276
- COMPUTATIONAL (COMP) 47
- COMPUTATIONAL-1 (COMP-1)
 - 形式 48
 - パフォーマンスに関するヒント 591
- COMPUTATIONAL-2 (COMP-2)
 - 形式 48
 - パフォーマンスに関するヒント 591
- COMPUTATIONAL-3 (COMP-3)
 - 説明 48
 - 日付フィールドの起こりうる問題 583
- COMPUTATIONAL-4 (COMP-4) 47
- COMPUTATIONAL-5 (COMP-5) 47
- COMPUTE ステートメント
 - コーディングが容易 54
 - 算術結果の割り当て 35
- COMP-1 (COMPUTATIONAL-1)
 - 形式 48
 - パフォーマンスに関するヒント 591
- COMP-2 (COMPUTATIONAL-2)
 - 形式 48

COMP-2 (COMPUTATIONAL-2) (続き)
パフォーマンスに関するヒント 591

COMP-3 (COMPUTATIONAL-3) 48

COMP-4 (COMPUTATIONAL-4) 47

COMP-5 (COMPUTATIONAL-5) 47

CONFIGURATION SECTION 7

CONTINUE ステートメント 86

CONTROL ステートメント 319

CONVERTING 句 (INSPECT) の例 111
coprocessor、DB2
概要 357
SQL INCLUDE の使用 359

COPY ステートメント
移植性のために使用 493
検索規則 319
説明 319
ネストされた 285、601
例 602

COPY 名
検索されるファイル接尾部 226

COUNT IN 句
UNSTRING 102
XML GENERATE 413

CURRENCY コンパイラー・オプション
277

CURRENT-DATE 組み込み関数
例 58
CICS のもとで 365

C++
COBOL から呼び出される関数、例
514

C/C++
データ型、COBOL との対応 511
COBOL 508
COBOL との通信
概要 508
制約事項 508
COBOL プログラムに対する複数の呼
び出し 509

D

DATA DIVISION
インスタンス・データ 434、460
インスタンス・メソッド 438
クライアント 447
グループ・レベルの USAGE
NATIONAL 節 193
グループ・レベルの USAGE 節 25
コーディング 12
項目のマッピング 293、342
制限 12
説明 12
ファクトリー・データ 463
ファクトリー・メソッド 465
リスト 342

DATA DIVISION (続き)
FD 記入項目 12
FILE SECTION 12
GROUP-USAGE NATIONAL 節 66
LINKAGE SECTION 12、16
LOCAL-STORAGE SECTION 12
OCCURS DEPENDING ON (ODO) 節
77
OCCURS 節 65
REDEFINES 節 74
USAGE IS INDEX 節 70
WORKING-STORAGE SECTION 12

DATA RECORDS 節 13

DATE FORMAT 節
国別データと一緒に使用できない
562
自動日付認識に使用 561

DATEPROC コンパイラー・オプション
警告レベル・メッセージの分析 581
説明 278

DATEVAL 組み込み関数
使用 579
例 580

DATE-COMPILED 段落 5

DATE-OF-INTEGGER 組み込み関数 58

DB2
オプション 360
コーディングに関する考慮事項 357
バインド・ファイル名 361
パッケージ名 361
プリコンパイラーは NODYNAM を必
要とする 357
無視されるオプション 360
64 ビット・モードの考慮事項 618
coprocessor
概要 357
SQL INCLUDE の使用 359

SQL ステートメント
概要 357
コーディング 358
バイナリー・データの使用 359
戻りコード 360
SQL INCLUDE 359

DB2DBDFT 環境変数 226、360

DB2INCLUDE 環境変数 359

DBCS データ
エンコードおよびストレージ 196
これを伴う MOVE ステートメント
33
宣言する 206
テスト 207
比較する
英数字グループと 219
国別と 204、219
照合シーケンスの影響 218
リテラル 207

DBCS データ (続き)
変換
国別への、概要 208
リテラル
最大長 207
使用 206
説明 26
比較する 207

DBCS 比較 90

DEBUG ランタイム・オプション 326

DESC サブオプション、CALLINT コンパ
イラー・オプション 270

DESCRIPTOR サブオプション、CALLINT
コンパイラー・オプション 270

DFHCOMMAREA パラメーター
CICS 動的呼び出しで使用 366

DFHEIBLK パラメーター
CICS 動的呼び出しで使用 366

DIAGTRUNC コンパイラー・オプション
279

DISPLAY (USAGE IS)
エンコードおよびストレージ 196
外部 10 進数 46
浮動小数点 46

DISPLAY ステートメント
データ値の表示 37
デバッグでの使用 332
GUI アプリケーションでの使用 228

DISPLAY ステートメント、使用される環
境変数 228

DISPLAY-1 (USAGE IS)
エンコードおよびストレージ 196

DISPLAY-OF 組み込み関数
ギリシャ語データでの例 200
使用 199
中国語データでの例 202
UTF-8 データでの例 201
XML 文書での 399

DO ループ 95

do-until 96

do-while 96

dspscat コマンドを使用したリスト表示
781

DYNAM コンパイラー・オプション
説明 280
パフォーマンスの考慮事項 597
CALL literal への影響 507

E

E レベルのエラー・メッセージ 242、338

EBCDIC
マルチバイトの移植性 496
ASCII への変換 114
SBCS の移植性 495

EBCDIC (続き)
 XML 文書でサポートされるコード・
 ページ 397

EBCDIC_CODEPAGE 環境変数
 設定 228

Encina SFS サーバー
 完全修飾名 122
 サーバー名の指定 148

Encina SFS ファイル
 アクセス 367
 概要 148
 例 149
 完全修飾名 148
 基本ファイル名 122
 互換 616
 コンパイル 238
 識別 121
 使用可能なデータ・ボリュームの判別
 148
 処理 124
 代替索引の追加 150
 代替索引ファイルの作成 148
 代替索引ファイル名 122
 データ・ボリュームの指定 148
 パフォーマンスの向上 151
 非トランザクション・アクセス 126
 ファイル名 126
 編成 126
 1 次索引と副次索引 126
 COBOL コーディング例 149
 SFS ファイルの作成
 環境変数 148
 sfsadmin コマンド 150

Encina SFS ファイル・システム
 完全修飾ファイル名 126
 システム管理 126
 制約事項 125
 説明 125
 パフォーマンスの向上 151
 非階層型 126
 SFS ファイルへのアクセス
 概要 148
 例 149

ENCINA_CDS_ROOT 環境変数 228

ENCINA_SFS_DATA_VOLUME 環境変数
 228

ENCINA_SFS_INDEX_VOLUME 環境変数
 228

ENCINA_VSAM_AUTO_FLUSH 環境変数
 228

ENCINA_VSAM_CACHE 環境変数 228

ENTRY ステートメント
 その中でのプログラム名の処理 299
 代替入り口点の 526

ENTRYINT コンパイラー・オプション
 説明 281

ENVIRONMENT DIVISION
 インスタンス・メソッド 438
 クライアント 446
 クラス 433
 サブクラス 460
 照合シーケンスのコーディング 8
 説明 7

CONFIGURATION SECTION 7

INPUT-OUTPUT SECTION 7

ERRCOUNT ランタイム・オプション
 326

ERRMSG、エラー・メッセージのリスト
 の生成 243

EUC コード・ページ 211

EVALUATE ステートメント
 いくつかの条件をテストする例 89
 ケース構造 87
 コーディング 87
 構造化プログラミング 588
 ネストされた IF と対比 88, 89
 パフォーマンス 88
 複数値のテストの例 92, 93
 複数条件のテストに使用 85
 複数の WHEN 句の例 89
 THRU 句の例 88

EXCEPTION XML イベント 403

EXCEPTION/ERROR 宣言
 説明 173
 ファイル状況キー 175

EXIT PROGRAM ステートメント
 サブプログラムにおける 502
 メインプログラムにおける 502

EXIT コンパイラー・オプション
 説明 281
 文字ストリング形式 282

ADEXIT サブオプション 286

INEXIT サブオプション 284

LIBEXIT サブオプション 284

PRTEXIT サブオプション 286

export コマンド
 環境変数の定義 225
 パス検索順序 226

EXTERNAL 節
 データ項目の 527
 ファイルの共用 13, 528
 ファイルの場合の例 529

EXTERNAL データ
 共用 527

F

FACTORY 段落
 ファクトリー・データ 463
 ファクトリー・メソッド 464

FD (ファイル記述) 記入項目 13

FILE SECTION
 説明 12
 レコードの説明 12

DATA RECORDS 節 13

EXTERNAL 節 13

FD 記入項目 13

GLOBAL 節 13

RECORD CONTAINS 節 13

RECORD IS VARYING 13

RECORDING MODE 節 13

VALUE OF 13

FILE STATUS 節
 使用 174
 状況コードを持つ
 概要 175
 例 176
 ファイルのロード 144
 例 178

FILEMODE ランタイム・オプション
 互換 131
 説明 327

FILESYS ランタイム・オプション 328

FILE-CONTROL 段落、例 8

FLAG コンパイラー・オプション
 コンパイラー出力 339
 使用 338
 説明 286

FLAGSTD コンパイラー・オプション
 288

FLOAT コンパイラー・オプション 289

G

GB 18030 データ
 国別との間の変換 201
 処理 201

get メソッドおよび set メソッド 442

getenv()、環境変数へのアクセス 226

GETMAIN のアドレスの保管 283

GLOBAL 節、ファイルに対する 13, 17

GOBACK ステートメント
 サブプログラムにおける 502
 メインプログラムにおける 502

GROUP-USAGE NATIONAL 節
 国別グループの初期化 32
 国別グループの宣言の例 25
 国別グループの定義 194
 テーブルの定義 66
 Java との通信 483

I

I レベルのメッセージ 242, 338

idebug コマンド、例 354

IDENTIFICATION DIVISION

- クライアント 445
- クラス 432
- コーディング 5
- サブクラス 459
- 必要な段落 5
- メソッド 437
- リストのヘッダーの例 7
- CLASS-ID 段落 432, 459
- DATE-COMPILED 段落 5
- PROGRAM-ID 段落 5
- TITLE ステートメント 7

IEEE

- 移植性 496

IF ステートメント

- コーディング 85
- ネストされた 86
- 複数条件の場合に、代わりに
EVALUATE を使用 86
- NULL ブランチを伴う 85

IGZEDT4: 4 桁年号を使用した現在日付の取得 689

INITIAL 属性 502

- ネストされたプログラムへの影響 6
- プログラムを初期状態に設定 6

INITIALIZE ステートメント

- 国別グループ値のロード 32
- グループ値のロード 31
- テーブルの値のロード 72
- デバッグ用の使用 333
- 例 28
- REPLACING 句 72

INPUT-OUTPUT SECTION 7

INSPECT ステートメント

- 使用 110
- 例 110

INTEGER 組み込み関数の例 109

INTEGER-OF-DATE 組み込み関数 58

INTEGER-PART 組み込み関数 109

INVALID KEY 句

- 説明 177
- 例 178

INVOKE ステートメント

- オブジェクトの作成に使用 455
- メソッドの呼び出しに使用 450
- ON EXCEPTION を指定した 451, 467

PROCEDURE DIVISION RETURNING で 527

RETURNING 句 454

USING 句 451

iwzGetSortErrno、ソートまたはマージのエラー番号の取得 165

J

Java

- インターオペラビリティ 477
- オブジェクト配列 484
- クラス型 483
- グローバル参照
 - 受け渡し 480
 - オブジェクト 480
 - 管理 480
 - JNI サービス 481
- ストリング
 - 宣言する 484
 - 取り扱い 488
- ストリング配列 484
- 相互運用可能データ型、コーディング 483
- データ共用 482
- 長い配列 484
- 二重配列 485
- バイト配列 484
- 配列
 - 宣言する 484
 - 取り扱い 485
 - 例 487
- 配列クラス 482
- ブール配列 484
- 浮動配列 485
- 短い配列 484
- メソッド
 - アクセス制御 482
- 文字配列 484
- ライブラリー
 - 更新 257
 - cob2.cfg 内に指定 239
- 例
 - 配列の処理 487
 - 例外処理 479
- 例外
 - 処理 479
 - 例 479
 - catch 479
 - throw 479
- ローカル参照
 - 受け渡し 480
 - オブジェクト 480
 - 解放 481
 - 管理 480
 - 削除 481
 - 保管 481
 - マルチスレッド化ごと 481
 - JNI サービス 481
- boolean 型 483
- byte 型 483
- char 型 483

Java (続き)

COBOL

- 間の通信 477
- アプリケーションの構造化 473
- 互換 615
- コンパイル 257
- 実行 260
- リンク 258

COBOL での実行

- 概要 260
- SDK サポート 261

double 型 483

float 型 483

int 型 483

int 配列 484

jstring クラス 482

long 型 483

SDK サポート 261

short 型 483

Java 仮想マシン

- オブジェクト参照 480
- 初期化 260
- 例外 479

Java との相互運用が可能なデータ型 483

javac コマンド

- Java 5 または Java 6 用に再コンパイル 615

Java クラス定義のコンパイル 257

java.lang.Object

- Base として参照 432

JNI

- オブジェクト参照の比較 449
- クラス・オブジェクト参照の取得 478
- 構造環境 477
 - アドレス可能度の場合 478
- サービスへアクセス 477
- 使用した場合の制限 478
- ライブラリー
 - 更新 257
 - cob2.cfg 内に指定 239
- 例外取り扱いサービス 479
- ローカル参照からグローバルへの変換 455
- Java ストリング・サービス 488
- Java 配列サービス 485
- Unicode サービス 488
- UTF-8 サービス 488
- JNINativePtr 特殊レジスター
 - サイズは ADDR に依存 267
 - 32 ビット・モードおよび 64 ビット・モード 617
- JNI 呼び出し可能サービスのための使用 478
- JNINativeInterface
 - 構造環境 477
 - JNI.cpy 477

JNI.cpy
コンパイル用 257
リスト 705
JNINativeInterface の場合 477
jstring Java クラス 482

L

LANG 環境変数 228
LC_ALL 211
LC_ALL 環境変数 228
LC_COLLATE 211
LC_COLLATE 環境変数 228
LC_CTYPE 211
LC_CTYPE 環境変数 228
LC_MESSAGES 211
LC_MESSAGES 環境変数 228
LC_TIME 211
LC_TIME 環境変数 228
LENGTH OF 特殊レジスター
受け渡し 518
サイズは ADDR に依存 268
使用 118
32 ビット・モードおよび 64 ビット・
モード 617
LENGTH 組み込み関数
可変長の結果 116
国別データを伴う 118
結果サイズは ADDR に依存 268
使用 114
例 58, 118
32 ビット・モードおよび 64 ビット・
モード 617
LENGTH OF 特殊レジスターと比較
118
LIB コンパイラー・オプション 290
libjava.a 259
LIBPATH 環境変数
説明 228
COBOL クラスの場所の指定 260
library-name
指定されなかった場合の代替 245
ライブラリー・テキストのパスの指定
226, 319
LINECOUNT コンパイラー・オプション
291
LINKAGE SECTION
コーディング 520
再帰呼び出し 17
パラメーターを記述するための 519
THREAD オプションを指定した 17
LIST コンパイラー・オプション
出力の取得 342
説明 291
デバッグでの使用 353

LOCAL-STORAGE SECTION
クライアント 447, 448
WORKING-STORAGE との比較
概要 14
例 15
OO クライアント 448
LOCPATH 環境変数 228
LOG 組み込み関数 59
LOWER-CASE 組み込み関数 112
lst ファイル・サフィックス 243
LSTFILE コンパイラー・オプション 292

M

MAP コンパイラー・オプション
組み込みマップ要約 342
出力で使用される記号 347
出力で使用される用語 347
使用 341, 342
説明 293
ネストされたプログラム・マップ 342
例 348
例 346, 348
MAP 出力で使用される記号 347
MAP 出力で使用される用語 347
MAX 組み込み関数
関数の例 58
使用 115
テーブル計算の例 84
MAXMEM コンパイラー・オプション
294
MDECK コンパイラー・オプション
説明 295
マルチオプションの相互作用 265
MEAN 組み込み関数
テーブル計算の例 84
統計計算の例 59
MEDIAN 組み込み関数
テーブル計算の例 84
統計計算の例 59
MERGE 作業ファイル 228
MERGE ステートメント
概要 155
説明 161
ASCENDING/DSCENDING KEY 句
162
COLLATING SEQUENCE 句 8, 163
GIVING 句 161
USING 句 161
METHOD-ID 段落 437
MIN 組み込み関数
使用 115
例 109
MLE 562
MLE での非日付 575

MOVE ステートメント
基本受信項目を伴う 33
国別項目を伴う 33
国別データへの変換 198
グループ移動と基本移動の対比 34,
195
グループ受信項目を伴う 34
算術結果の割り当て 35
送信項目および受信項目の長さに対す
る ODO の影響 78
CORRESPONDING 34

N

N 区切り文字、国別または DBCS リテラ
ル用の 26
NATIONAL (USAGE IS)
外部 10 進数 46
浮動小数点 46
NATIONAL-OF 組み込み関数
ギリシャ語データでの例 200
使用 198
中国語データでの例 202
UTF-8 データでの例 201
XML 文書での 399
NCOLLSEQ コンパイラー・オプション
国別照合シーケンスへの影響 216,
219
国別比較の影響 203
説明 296
ソートおよびマージ・キーへの影響
162
NLSPATH 環境変数 228
NOCOMPILE コンパイラー・オプション
構文エラーの検出に使用 336
NODESC サブオプション、CALLINT コ
ンパイラー・オプション 270
NODESCRIPTOR サブオプション、
CALLINT コンパイラー・オプション
270
NOSSRANGE コンパイラー・オプション
エラー検査への影響 325
NSYMBOL コンパイラー・オプション
国別データ項目の 190
国別リテラルの 190
説明 296
DBCS リテラル用の 190
N リテラルへの影響 26
NULL ブランチ 85
NUMBER コンパイラー・オプション
説明 297
デバッグ使用 343
NUMVAL 組み込み関数
説明 113
NUMVAL-C 組み込み関数
説明 113

NUMVAL-C 組み込み関数 (続き)
例 58
NX 区切り文字、国別リテラル用の 26

O

OBJECT 段落
インスタンス・データ 434, 460
インスタンス・メソッド 436
OBJECT-COMPUTER 段落 7
OCCURS DEPENDING ON (ODO) 節
可変長テーブル作成用 77
最適化 593
単純 77
複合 643
ODO エレメントの初期化 80
ODO オブジェクト 77
ODO サブジェクト 77
OCCURS INDEXED BY 節による指標の
作成 70
OCCURS 節
指標作成用の INDEXED BY 句 70
多次元テーブルを作成するためにネス
トされた 66
テーブルの定義 65
テーブル・エレメントの定義 66
レベル 01 項目では使用できない 66
ASCENDING|DESCENDING KEY 句
テーブル・エレメントの順序の指定
66
二分探索に必要 82
例 83
ODBC
概要 371
組み込み SQL 371
組み込み SQL との比較 371
コピーブック
例 378
odbc3d 382
odbc3p 379
ドライバのインストールおよび構成
372
ドライバ・マネージャー 372
バックグラウンド 372
付属のコピーブック 376
戻り値へのアクセス 375
呼び出しインターフェース規約 384
利点 371
C データ型のマッピング 372
COBOL からの API の使用 372
COBOL ポインターの受け渡し 373
ODO オブジェクト 77
ODO サブジェクト 77
OMITTED パラメーター 604
ON EXCEPTION 句
INVOKE ステートメント 451, 467

ON SIZE ERROR
ウィンドウ表示日付フィールドでの
577
OO アプリケーションの構造化 473
OPEN ステートメント
ファイル状況キー 174
ファイルの可用性 140
OPEN 命令コード 284
OPTIMIZE コンパイラー・オプション
使用 595
説明 298
パフォーマンスの考慮事項 597
パフォーマンスへの影響 595
パラメーター引き渡しの影響 520
ORD 組み込み関数の例 115
ORD-MAX 組み込み関数
使用 116
テーブル計算の例 84
ORD-MIN 組み込み関数 116

P

PATH 環境変数
説明 228
PERFORM ステートメント
インライン 94
指標を変更するための 71
テーブル用の
指標付けを使用した例 76
添え字付けを使用した例 75
明示的に指定した回数だけ実行される
95
ライン外 94
ループのコーディング 94
TEST AFTER 96
TEST BEFORE 96
THRU 97
TIMES 95
UNTIL 96
VARYING 96
VARYING WITH TEST AFTER 96
WITH TEST AFTER . . . UNTIL 96
WITH TEST BEFORE . . .
UNTIL 96
PGMNAME コンパイラー・オプション
299
PICTURE 節
国別データを表す N 189
国別編集データ 189
使用される記号の判別 277
数字編集データ 190
数値データ 41
ゼロ抑制用の Z 43
内部浮動小数点に使用できない 42
非互換データ 54
PRESENT-VALUE 組み込み関数 58

PROCESS (CBL) ステートメント
コンパイラー・オプションの指定 238
説明 319
矛盾するオプション 265
PROFILE コンパイラー・オプション
301
PROGRAM COLLATING SEQUENCE 節
英数字比較への影響 218
国別比較に影響なし 219
国別または DBCS オペランドに影響
を与えない 9
照合シーケンスの設定 8
デフォルト照合シーケンスのオーバ
ライド 163
COLLATING SEQUENCE 句によるオ
ーバライド 8
COLLSEQ の相互作用 275
DBCS 比較に影響なし 218
PROGRAM-ID 段落
コーディング 5
COMMON 属性 6
INITIAL 属性 6
putenv()、環境変数の設定 226

Q

QUOTE コンパイラー・オプション 301

R

RANGE 組み込み関数
テーブル計算の例 84
統計計算の例 59
RECORD CONTAINS 節
FILE SECTION 記入項目 13
RECORDING MODE 節
QSAM ファイル 13
REDEFINES 節を使用して、レコードをテ
ーブルに作成 74
RELEASE FROM ステートメント
例 158
RELEASE との比較 159
RELEASE ステートメント
RELEASE FROM との比較 159
SORT での 158, 159
REM 組み込み関数 59
REPLACING 句 (INSPECT) の例 110
REPOSITORY 段落
クライアント 446
クラス 433
コーディング 7
サブクラス 460
RETURN ステートメント
出力プロシージャで必要 160
INTO 句を指定した 160

RETURNING 句
 メソッドでの使用 527
 CALL ステートメント 527
 INVOKE ステートメント 454
 PROCEDURE DIVISION ヘッダー
 440
 RETURN-CODE 特殊レジスター
 受け渡し、プログラム間でのデータの
 527
 オペレーティング・システムに制御権
 が戻される時 516
 スレッド化サポートの場合 516
 日時サービスへの呼び出し後の値 604
 非スレッド化サポートの場合 516
 プログラム間での戻りコードの共用
 526
 INVOKE で設定しない 451
 REVERSE 組み込み関数 112
 ROUNDED 句 632
 RSD ファイル
 識別 121
 処理 124
 RSD ファイル・システム 129

S

S レベルのエラー・メッセージ 242, 338
 SD (ソート記述) 項目の例 157
 SEARCH ALL ステートメント
 指標を変更するための 71
 テーブルは順序付けが必要 82
 二分探索 82
 例 83
 SEARCH ステートメント
 指標を変更するための 71
 逐次探索 81
 テーブルの複数のレベルを検索するた
 めのネスト 81
 例 81
 SELECT OPTIONAL 140
 SELECT 節
 入出力ファイルの変更 11
 SELF 449
 SEPOBJ コンパイラー・オプション 302
 SEQUENCE コンパイラー・オプション
 303
 SET 条件名 TO TRUE ステートメント
 スイッチおよびフラグ 93
 例 95, 96
 SET ステートメント
 オブジェクト参照用 449
 指標データ項目を変更するための 70
 指標を変更するための 71
 条件設定用の、例 93
 その中でのプログラム名の処理 299
 デバッグ用の使用 333
 SET ステートメント (続き)
 プロシージャ・ポインター・データ
 項目用 526
 SFS (Encina) サーバー
 完全修飾名 122
 サーバー名の指定 148
 SFS (Encina) ファイル
 アクセス 367
 概要 148
 例 149
 完全修飾名 148
 基本ファイル名 122
 互換 616
 コンパイル 238
 識別 121
 使用可能なデータ・ボリュームの判別
 148
 処理 124
 代替索引の追加 150
 代替索引ファイルの作成 148
 代替索引ファイル名 122
 データ・ボリュームの指定 148
 パフォーマンスの向上 151
 非トランザクション・アクセス 126
 ファイル名 126
 編成 126
 1 次索引と副次索引 126
 COBOL コーディング例 149
 SFS ファイルの作成
 環境変数 148
 sfsadmin コマンド 150
 SFS (Encina) ファイル・システム
 完全修飾ファイル名 126
 システム管理 126
 制約事項 125
 説明 125
 パフォーマンスの向上 151
 非階層型 126
 SFS ファイルへのアクセス
 概要 148
 例 149
 sfsadmin コマンド
 索引付きファイルの作成 150
 使用可能なデータ・ボリュームの判別
 148
 説明 126
 代替索引の追加 150
 SIGN IS SEPARATE 節
 移植性 42
 印刷 42
 符号付き国別 10 進数データに必要
 42
 SIZE コンパイラー・オプション 304
 SORT 作業ファイル 228
 SORT ステートメント
 概要 155
 SORT ステートメント (続き)
 説明 161
 ASCENDINGDESCENDING KEY 句
 162
 COLLATING SEQUENCE 句 8, 163
 GIVING 句 161
 USING 句 161
 SORT-RETURN 特殊レジスター
 ソートまたはマージの終了 168
 ソートまたはマージの成功の判断 164
 SOSI コンパイラー・オプション
 説明 305
 マルチバイトの移植性 496
 SOURCE および NUMBER 出力の例
 345
 SOURCE コンパイラー・オプション
 出力の取得 342
 説明 306
 SOURCE-COMPUTER 段落 7
 SPACE コンパイラー・オプション 307
 SPECIAL-NAMES 段落
 コーディング 7
 SPILL コンパイラー・オプション 308
 SQL コンパイラー・オプション
 オブジェクト指向プログラムの制約事
 項 427
 コーディング 360
 説明 308
 マルチオプションの相互作用 265
 SQL ステートメント
 概要 357
 コーディング 358
 バイナリー・データの使用 359
 戻りコード 360
 DB2 サービスのための使用 357
 SQL INCLUDE 359
 SQLCA
 DB2 からの戻りコード 360
 SQL ステートメントを使用するプログ
 ラムについて宣言 358
 SQRT 組み込み関数 59
 SSRANGE コンパイラー・オプション
 参照変更 107
 使用 337
 説明 309
 パフォーマンスの考慮事項 597
 CHECK(OFF) ランタイム・オプション
 を使用してオフにする 597
 STDCALL インターフェース規約
 CALLINT で指定 270
 STL ファイル
 互換 131
 識別 121
 小容量形式
 互換 131
 定義 131

STL ファイル (続き)
 処理 124
 大容量形式 130
 FILEMODE によって影響を受けるサ
 イズ 327
 STL ファイル・システム
 説明 127
 戻りコード 127
 STOP RUN ステートメント
 サブプログラムにおける 502
 メインプログラムにおける 502
 STRING ステートメント
 オーバーフロー条件 169
 使用 99
 例 100
 SUM 組み込み関数、テーブル計算の例
 84
 SUPER 455
 SYMBOLIC CHARACTERS 節 10
 SYNCHRONIZED 文節
 調整は ADDR に依存 267
 32 ビット・モードおよび 64 ビット・
 モード 618
 SYSADATA
 出力 266
 SYSADATA ファイル
 ファイル内容 711
 例 713
 レコード記述 714
 レコード・タイプ 712
 SYSADATA レコード
 モジュールの提供 281
 呼び出される出口モジュール 286
 SYSIN
 代替モジュールの提供 281
 SYSIN、SYSIPT、SYSOUT、
 SYSLIST、SYSLST、CONSOLE、
 SYSPUNCH、SYSPCH 環境変数 228
 SYSLIB
 使用されない場合 284
 代替モジュールの提供 281
 SYSLIB 環境変数 226
 JNL.cpy の場所の指定 257
 SYSPRINT
 使用されない場合 286
 代替モジュールの提供 281
 SYSTEM インターフェース規約
 CALLINT で指定 270
 SYSTEM サブオプション、CALLINT コ
 ンパイラー・オプション 270
 SYSTEM データ・セット
 メッセージの送信 310

T

TALLYING 句 (INSPECT)、例 110
 TERMINAL コンパイラー・オプション
 310
 TEST AFTER 96
 TEST BEFORE 96
 TEST コンパイラー・オプション
 説明 310
 デバッグに使用 341
 パフォーマンスの考慮事項 597
 マルチオプションの相互作用 265
 THREAD コンパイラー・オプション
 オブジェクト指向 COBOL のための
 257
 説明 311
 パフォーマンスの考慮事項 597
 cob2_j のデフォルト 312
 Java とのインターオペラビリティの
 ための 257
 LINKAGE SECTION 17
 TITLE ステートメント
 リストのヘッダーの制御 7
 TMP 環境変数 228
 TRAP ランタイム・オプション
 説明 328
 ON SIZE ERROR 170
 TRUNC コンパイラー・オプション
 説明 312
 パフォーマンスの考慮事項 597
 TZ 環境変数 228

U

U レベルのエラー・メッセージ 242, 338
 UNDATE 組み込み関数
 使用 580
 例 580
 Unicode
 エンコードおよびストレージ 196
 説明 188
 データ処理 183
 JNI サービス 488
 UNSTRING ステートメント
 オーバーフロー条件 169
 使用 102
 例 103
 UPPER-CASE 組み込み関数 112
 UPSI スイッチ、設定 329
 UPSI ランタイム・オプション 329
 USAGE 節
 グループ・レベルの 25
 グループ・レベルの NATIONAL 句
 193
 非互換データ 54

USAGE 節 (続き)

INDEX 句による指標データ項目の作
 成 70
 OBJECT REFERENCE 447
 USE FOR DEBUGGING 宣言
 概要 334
 DEBUG ランタイム・オプション 326
 USING 句
 INVOKE ステートメント 451
 PROCEDURE DIVISION ヘッダー
 440, 521
 UTF-16
 エンコード方式、国別データの 188
 定義 188
 UTF-8
 エンコードおよびストレージ 196
 国別との間の変換 201
 データ項目の処理 201
 定義 188
 ASCII インバリエント文字のエンコー
 ド方式 188
 JNI サービス 488
 XML 文書エンコード 398
 XML 文書で参照変更を回避 201
 XML 文書の構文解析 400
 XML 文書の生成例 409

V

VALUE IS NULL 523
 VALUE OF 節 13
 VALUE 節
 大きな、TRUNC(BIN) での 312
 外部浮動小数点に使用できない 47
 可変長グループへの割り当て 80
 国別グループでの英数字リテラル、例
 75
 国別データを持つ英数字リテラルの例
 117
 テーブルの値の割り当て
 エレメントのそれぞれの出現への
 75
 グループ・レベルの 74
 それぞれの項目に個別に 74
 内部浮動小数点リテラルの初期化 42
 COMP-5 を指定したラージ・リテラル
 48
 VBREF コンパイラー・オプション
 出力例 352
 使用 342
 説明 315
 VSAM ファイル
 エラー処理 171
 互換 131
 識別 121

VSAM ファイル (続き)
状況コード
概要 175
例 176
小容量形式
互換 131
定義 131
処理 124
大容量形式 130
マルチスレッド化環境において 550
FILEMODE によって影響を受けるサ
イズ 327
VSAM ファイル・システム 125

W

W レベルのメッセージ 242, 338
WHEN 句
EVALUATE ステートメント 87
SEARCH ALL ステートメント 82
SEARCH ステートメント 81
WHEN-COMPILED 組み込み関数 118
WHEN-COMPILED 特殊レジスター 118
WITH DEBUGGING MODE 節
デバッグ行 334
デバッグ・ステートメント 334
WITH POINTER 句
STRING 99
UNSTRING 102
wlist ファイル 292
WORKING-STORAGE SECTION
インスタンス・データ 434, 460
インスタンス・メソッド 438
クライアント 447, 448
初期化 315
ファクトリー・データ 463
マルチスレッド化の考慮事項 448
LOCAL-STORAGE との比較
概要 14
例 15
OO クライアント 448
WSCLEAR コンパイラー・オプション
315

X

X 区切り文字、英数字リテラル内の制御
文字としての 26
XML GENERATE ステートメント
COUNT IN 413
NAMESPACE 409
NAMESPACE-PREFIX 410
NOT ON EXCEPTION 411
ON EXCEPTION 412
WITH ATTRIBUTES 409

XML GENERATE ステートメント (続き)
WITH ENCODING 411
XML-DECLARATION 409
XML PARSE ステートメント
概要 388
使用 389
NOT ON EXCEPTION 403
ON EXCEPTION 403
XML イベント
処理 390
処理プロシージャ 389
説明 388
EXCEPTION 403
XML 構文解析
概要 387
コード・ページの矛盾の処理 404
終了 404
処理プロシージャでの制御フロー
393
説明 389
特殊レジスター 391
例外の処理 401
CHAR(EBCDIC) の影響 398
XML 構文解析の終了 404
XML 出力
エンコードの制御 411
拡張
基本的原理と技法 418
データ定義の変更例 419
ハイフンを下線に変換する例 421
生成
概要 407
例 413
XML 出力の拡張
基本的原理と技法 418
データ定義の変更例 419
ハイフンを下線に変換する例 421
XML 出力の生成 407
概要 407
例 413
XML 処理プロシージャ
書き込み 391
コード・ページの矛盾に関連した 404
構文解析例外の処理 401
指定 389
特殊レジスターの使用 391
パーサーでの制御フロー 393
例 395
EXIT PROGRAM または GOBACK で
のエラー 392
XML PARSE の制約事項 391
XML 生成
エラーの処理 412
エレメントの生成 408
概要 407
互換 616

XML 生成 (続き)
出力の拡張
基本的原理と技法 418
データ定義の変更例 419
ハイフンを下線に変換する例 421
生成される文字のカウント 408
説明 407
属性の生成 409
名前空間接頭部の使用 410
名前空間の使用 409
バイト・オーダー・マークなし 412
無視されるデータ項目 408
例 413
CHAR(EBCDIC) の影響 398
XML 宣言
エンコード宣言の指定 400
生成 409
XML パーサー
エラー処理 403
概要 388
適合性 700
XML 文書
アクセス 389
エンコード 397, 398
エンコードの制御 411
外部コード・ページ 399
拡張
基本的原理と技法 418
データ定義の変更例 419
ハイフンを下線に変換する例 421
各国語 398
コード・ページの指定 399
構文解析
説明 389
例 395
UTF-8 400
構文解析例外の処理 401
サポートされるコード・ページ 397
処理 387
生成
概要 407
例 413
パーサー 388
文書エンコード宣言 399
EBCDIC 特殊文字 400
UTF-8 エンコード 398
XML 文書の構文解析
概要 388
説明 389
UTF-8 400
XML 例外コード
構文解析
処理可能でない 697
処理可能な 691
生成 702

XML-CODE 特殊レジスター

- コード・ページの矛盾に関連した 404
- 構文解析での使用 387
- 構文解析の終了 404
- 構文解析の例外 403
- 構文解析の例外コード
 - エンコードの競合 402
 - 処理可能でない 697
 - 処理可能な 691
- 生成での使用 411
- 生成の例外 412
- 生成の例外コード 702
- 説明 391
- 内容 393
- パーサーと処理プロシージャ間の制御フロー 393

XML-EVENT 特殊レジスター

- 構文解析の例外 403
- 使用 387, 390
- 説明 391
- 内容 392

XML-NTEXT 特殊レジスター

- 構文解析の例外 403
- 使用 387
- 説明 391
- 内容 393

XML-TEXT 特殊レジスター

- エンコード 391
- 構文解析の例外 403
- 使用 387
- 説明 391
- 内容 393

XREF コンパイラー・オプション

- コピーブック・ファイルの検索 340
- 出力の取得 342
- 説明 316
- データおよびプロシージャ名の検出 340

XREF 出力

- データ名相互参照 349
- プログラム名相互参照 350
- COPY/BASIS 相互参照 350

Y

YEARWINDOW コンパイラー・オプション

- 説明 317

Z

zSeries ホスト・データ形式

- 考慮事項 627

ZWB コンパイラー・オプション 318

[特殊文字]

- ! 文字、16 進値 400
- # 文字、16 進値 400
- *CBL ステートメント 319
- *CONTROL ステートメント 319
- bD cob2 オプション、ヒープ・スペースの指定 249
- c cob2 オプション 245
- cmain cob2 オプション 246
- compre_ok cob2 オプション 245
- Fxxx cob2 オプション 247
- g cob2 オプション
 - デバッグ用 247
- host cob2 オプション
 - コマンド行引数への影響 532
 - コンパイラー・オプションへの影響 245
 - ホストのデータ形式 245
- host cob2_j オプション
 - オブジェクト指向アプリケーションについて 619
- I cob2 オプション
 - コピーブックの検索 245
- main cob2 オプション
 - メインプログラムの指定 246
- o cob2 オプション
 - メインプログラムの指定 246
- p cob2 オプション
 - パフォーマンスの分析 246
- pg cob2 オプション 246
- q cob2 オプション 245
- q32 cob2 オプション
 - 説明 247
 - 32 ビット Java SDK の選択 262
- q64 cob2 オプション
 - 説明 247
 - 64 ビット Java SDK の選択 262
- S cob2 オプション
 - スタック・スペースの指定 249
- v cob2 オプション 247
- # cob2 オプション 239, 247
- ? cob2 オプション 247
- .adt ファイル 266
- .lst ファイル・サフィックス 243
- .profile ファイル、環境変数の設定 225
- .wlist ファイル 292
- ? cob2 オプション 247
- [文字、16 進値 400
- >>CALLINT ステートメント
 - 説明 319
- | 文字、16 進値 400
-] 文字、16 進値 400
- _iwzGetCCSID: コード・ページ ID から CCSID への変換
 - 例 221

_iwzGetCCSID: コード・ページ ID から CCSID への変換 (続き)

- syntax 221

_iwzGetLocaleCP: ロケールおよび EBCDIC コード・ページ値の取得

- 例 221

- syntax 220



プログラム番号: 5724-V62

Printed in Japan

SC88-5744-00



日本アイ・ビー・エム株式会社
〒106-8711 東京都港区六本木3-2-12