

Enterprise COBOL for z/OS
6.5

プログラミング・ガイド



注

本書および本書で紹介する製品をご使用になる前に、[1029 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。

目次

表.....	xxi
前書き.....	xxix
本書について.....	xxix
本書の使い方.....	xxix
略語.....	xxix
一般に使用される用語の比較.....	xxx
構文図の読み方.....	xxxii
例の示し方.....	xxxii
追加の資料およびサポート.....	xxxiii
ご意見の送付方法.....	xxxiii
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.....	3
プログラムの識別.....	3
プログラムを再帰的として識別する.....	4
収容プログラムによってプログラムに呼び出し可能なマークを付ける.....	4
プログラムを初期状態に設定する.....	5
ソース・リストのヘッダーの変更.....	5
コンピューター環境の記述.....	5
例: FILE-CONTROL の記入項目.....	6
照合シーケンスの指定.....	7
シンボリック文字を定義する.....	8
ユーザー定義のクラスを定義する.....	8
オペレーティング・システムに対してファイルを定義する.....	9
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.....	18
PROCEDURE DIVISION 内でロジックが分割される方法.....	19
宣言部分.....	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.....	25
定数の使用.....	26
形象定数の使用.....	26
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).....	32
グループ・データ項目への値の割り当て (MOVE).....	33
算術結果の割り当て (MOVE または COMPUTE).....	34
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).....	36
システム論理出力装置上でのデータの表示.....	37

WITH NO ADVANCING の使用.....	37
組み込み関数の使用.....	38
テーブル (配列) とポインターの使用.....	39
ストレージとそのアドレス可能度.....	40
ユーザー定義型.....	43
TYPEDEF 文節.....	45
TYPE 節.....	45
TYPEDEF をサポートする機能.....	47
制約事項.....	48
第 3 章数値および算術演算.....	49
数値データの定義.....	49
数値データの表示.....	51
数値データの保管方法の制御.....	52
数値データの形式.....	53
例: 数値データおよび内部表現.....	56
データ形式の変換.....	58
変換および精度.....	58
ゾーンおよびパック 10 進数データのサイン表記.....	59
非互換データの検査 (数値のクラス・テスト).....	60
算術の実行.....	61
COMPUTE およびその他の算術ステートメントの使用.....	62
算術式の使用.....	62
数字組み込み関数の使用.....	63
数学用の呼び出し可能サービスの使用.....	64
データ呼び出し可能サービスの使用.....	66
例: 数字組み込み関数.....	67
固定小数点演算と浮動小数点演算の対比.....	68
例: 固定小数点計算および浮動小数点計算.....	70
通貨記号の使用.....	70
例: 複数の通貨符号.....	72
第 4 章テーブルの処理.....	73
テーブルの定義 (OCCURS).....	73
テーブルのネスト.....	74
例: 添え字付け.....	75
例: 指標付け.....	76
テーブル内の項目の参照.....	76
添え字付け.....	77
索引付け.....	77
テーブルに値を入れる方法.....	79
テーブルの動的なロード.....	79
テーブルの初期化 (INITIALIZE).....	79
テーブルの定義時の値の割り当て (VALUE).....	80
例: PERFORM と添え字付け.....	82
例: PERFORM および索引付け.....	83
可変長テーブルの作成 (DEPENDING ON).....	84
可変長テーブルのロード.....	86
可変長テーブルへの値の割り当て.....	87
複合 OCCURS DEPENDING ON.....	87
例: 複合 ODO.....	88
ODO オブジェクト値の変更の影響.....	89
テーブルの探索.....	91
逐次探索 (SEARCH).....	91
二分探索 (SEARCH ALL).....	92
テーブルのソート.....	94
組み込み関数を使用したテーブル項目の処理.....	94
例: 組み込み関数を使用したテーブルの処理.....	95

無制限テーブルおよび無制限グループの処理.....	95
例: XML 文書を構文解析するための無制限テーブルの使用.....	96
第 5 章 プログラム・アクションの選択と反復.....	99
プログラム・アクションの選択.....	99
アクションの選択項目のコーディング.....	99
条件式のコーディング.....	103
プログラム・アクションの繰り返し.....	107
インラインまたはライン外 PERFORM の選択.....	107
ループのコーディング.....	108
テーブルのループ処理.....	109
複数の段落またはセクションの実行.....	110
第 6 章 スtring の処理.....	111
データ項目の結合 (STRING).....	111
例: STRING ステートメント.....	112
データ項目の分割 (UNSTRING).....	113
例: UNSTRING ステートメント.....	114
ヌル終了Stringの取り扱い.....	116
例: ヌル終了String.....	117
データ項目のサブStringの参照.....	117
参照修飾子.....	118
例: 参照修飾子としての演算式.....	119
例: 参照修飾子としての組み込み関数.....	120
データ項目の計算および置換 (INSPECT).....	120
例: INSPECT ステートメント.....	121
データ項目の変換 (組み込み関数).....	122
大/小文字の変更 (UPPER-CASE、LOWER-CASE).....	122
逆順への変換 (REVERSE).....	123
数値への変換 (NUMVAL、NUMVAL-C、NUMVAL-F).....	123
あるコード・ページから別のコード・ページへの変換.....	124
16 進データまたはビット・データへの変換 (HEX-OF、BIT-OF).....	125
16 進データまたはビット・データからの変換 (HEX-TO-CHAR、BIT-TO-CHAR).....	125
データ項目の評価 (組み込み関数).....	126
照合シーケンスに関する単一文字の評価.....	127
最大または最小データ項目の検出.....	127
データ項目の長さの検出.....	129
コンパイルの日付の検出.....	130
第 7 章 国際環境でのデータの処理.....	131
Unicode および言語文字のエンコード.....	132
COBOL での国別データ (Unicode) の使用.....	133
国別データ項目の定義.....	134
国別リテラルの使用.....	134
COBOL ステートメントと国別データ.....	135
組み込み関数と国別データ.....	138
国別文字形象定数の使用.....	140
国別数値データ項目の定義.....	141
国別グループ.....	141
国別 (Unicode) 表現との間の変換.....	142
国別グループの使用.....	145
文字データの保管.....	148
国別 (UTF-16) データの比較.....	148
COBOL での UTF-8 データ (Unicode) の使用.....	151
UTF-8 データ項目の定義.....	151
UTF-8 リテラルの使用.....	153
COBOL ステートメントと UTF-8 データ項目.....	154
UTF-8 データ項目に対する組み込み関数サポート.....	155

Db2 に関する考慮事項.....	156
UTF-8 文字形象定数の使用.....	156
参照修飾子と UTF-8 データ項目.....	157
UTF-8 グループ.....	158
UTF-8 (Unicode) 表現と間の変換.....	159
UTF-8 データの比較.....	160
UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理.....	162
組み込み関数を使用した UTF-8 エンコード・データの処理.....	163
中国語 GB 18030 データの処理.....	167
DBCS サポート用のコーディング.....	168
DBCS データの定義.....	168
DBCS リテラルの使用.....	169
有効な DBCS 文字に関するテスト.....	169
DBCS データを含む英数字データ項目の処理.....	170
第 8 章 ファイルの処理.....	171
ファイル編成および入出力装置.....	171
ファイル編成およびアクセス・モードの選択.....	173
入出力コーディングの形式.....	173
ファイルの割り振り.....	175
入出力エラーの検査.....	176
第 9 章 QSAM ファイルの処理.....	177
COBOL での QSAM ファイルおよびレコードの定義.....	177
レコード形式の指定.....	178
ブロック・サイズの設定.....	184
QSAM ファイルの入出力ステートメントのコーディング.....	186
QSAM ファイルのオープン.....	187
QSAM ファイルの動的作成.....	188
QSAM ファイルへのレコードの追加.....	188
QSAM ファイルの更新.....	189
QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み.....	189
QSAM ファイルのクローズ.....	190
QSAM ファイルのエラーの処理.....	190
QSAM ファイルの操作.....	191
QSAM ファイルの定義および割り振り.....	191
QSAM ファイルの検索.....	193
ファイル属性をプログラムと一致させる.....	194
ストライプ拡張形式 QSAM データ・セットの使用.....	197
QSAM ファイル用のバッファの割り振り.....	197
QSAM を使用する z/OS ユニックスファイルへのアクセス.....	198
テープ上の QSAM ASCII ファイルの処理.....	198
第 10 章 VSAM ファイルの処理.....	201
VSAM ファイル.....	202
VSAM ファイル編成およびレコードの定義.....	203
VSAM ファイルの順次編成の指定方法.....	204
VSAM ファイルの索引編成の指定方法.....	204
VSAM ファイルの相対編成の指定方法.....	206
VSAM ファイルのアクセス・モードの指定.....	207
VSAM ファイルのレコード長の定義.....	207
VSAM ファイルの入出力ステートメントのコーディング.....	209
ファイル位置標識.....	211
ファイルのオープン (ESDS、KSDS、または RRDS).....	211
VSAM ファイルからのレコードの読み取り.....	214
VSAM ファイル内のレコードの更新.....	215
VSAM ファイルへのレコードの追加.....	216
VSAM ファイル内のレコードの置換.....	216

VSAM ファイルからのレコードの削除.....	217
VSAM ファイルのクローズ.....	217
VSAM ファイルでのエラー処理.....	217
パスワードによる VSAM ファイルの保護.....	218
例: VSAM 索引付きファイルのパスワード保護.....	219
z/OS および z/OS ユニックスのもとでの VSAM データ・セットの操作.....	219
VSAM ファイルの定義.....	219
代替索引の作成.....	220
VSAM ファイルの割り振り.....	222
RLS による VSAM ファイルの共用.....	223
VSAM ファイル用のレコード域の割り振り.....	225
VSAM パフォーマンスの向上.....	225
拡張アドレッシング機能サポート.....	227
 第 11 章 VSAMDB ファイルの処理.....	229
VSAMDB ファイルの定義	229
例: 動的長のキーとレコードを使った VSAMDB ファイルの定義.....	230
例: 固定長のキーとレコードを使った VSAMDB ファイルの定義.....	230
VSAMDB ファイルの使用.....	231
ファイル位置標識.....	231
VSAMDB ファイルからレコードを読み込む.....	232
例: JSON GENERATE を使って JSON 文字列を作成する.....	232
例: JSON GENERATE を使って JSON 文字列を固定長アイテムにする.....	232
RECORD KEY LENGTH フレーズの使用.....	233
動的長レコードでの READ 文の使用.....	233
VSAMDB ファイルへのレコードの追加.....	234
VSAMDB ファイルのレコードの更新.....	234
VSAMDB ファイルのレコードの削除.....	234
VSAMDB ファイルの定義 z/OS.....	234
DFSMSHVS で VSAMDB を使用する.....	235
VSAMDB システム要件.....	235
VSAMDB 全プログラムのサンプル.....	235
 第 12 章 行順次ファイルの処理.....	237
COBOL での行順次ファイルおよびレコードの定義.....	237
行順次ファイルの構造の記述.....	238
行順次ファイルの制御文字.....	238
行順次ファイルの割り振り.....	238
行順次ファイル用の入出力ステートメントのコーディング.....	239
行順次ファイルのオープン.....	240
行順次ファイルからのレコードの読み取り.....	240
行順次ファイルへのレコードの追加.....	241
行順次ファイルのクローズ.....	241
行順次ファイルのエラーの処理.....	241
 第 13 章 ファイルのソートおよびマージ.....	243
ソートおよびマージ・プロセス.....	244
ソートまたはマージ・ファイルの記述.....	245
ソートまたはマージへの入力の記述.....	245
例: SORT 用のソート・ファイルおよび入力ファイルの記述.....	246
入力プロシージャのコーディング.....	247
ソートまたはマージからの出力の記述.....	247
出力プロシージャのコーディング.....	248
例: DFSORT を使用する際の出力プロシージャのコーディング.....	249
入出力プロシージャに関する制約事項.....	249
ソート・データ・セットおよびマージ・データ・セットの定義.....	250
可変長レコードのソート.....	250
ソートまたはマージの要求.....	251

ソートまたはマージ基準の設定.....	252
例: 入出力プロシージャーを使用したソート.....	253
代替照合シーケンスの選択.....	254
同じキーを持つレコードのオリジナル・シーケンスの保持.....	254
ソートまたはマージの成否の判断.....	254
ソートまたはマージ操作の途中停止.....	255
FASTSORT によるソート・パフォーマンスの向上.....	255
JCL に関する FASTSORT の要件.....	256
ソート入出力ファイルに関する FASTSORT の要件.....	256
NOFASTSORT によるソート・エラーの検査.....	258
ソート動作の制御.....	258
制御ステートメントによる DFSORT デフォルトの変更.....	259
ソートまたはマージ操作のためのストレージの割り振り.....	260
ソート・ファイル用のスペースの割り振り.....	261
DFSORT によるチェックポイント・リスタートの使用.....	261
DFSORT での COBOL ユーザー出口ルーチンの使用.....	262
CICS のもとでのソート.....	262
CICS SORT アプリケーションの制約事項.....	262
第 14 章エラーの処理.....	265
ダンプの要求.....	265
ストリングの結合および分割におけるエラーの処理.....	266
算術演算でのエラーの処理.....	266
例: 0 による除算の検査.....	267
入出力操作でのエラーの処理.....	267
ファイルの終わり条件 (AT END) の使用.....	269
ERROR 宣言のコーディング.....	270
ファイル状況キーの使用.....	271
例: ファイル状況キー.....	272
VSAM 状況コードの使用 (VSAM ファイルのみ).....	273
例: VSAM 状況コードの検査.....	273
INVALID KEY 句のコーディング.....	275
例: FILE STATUS および INVALID KEY.....	275
プログラム呼び出し時のエラーの処理.....	275
エラー処理用のルーチンの作成.....	276
第 2 部プログラムのコンパイルおよびデバッグ.....	277
第 15 章 z/OS のもとでのコンパイル.....	279
JCL を使用したコンパイル.....	279
カタログ式プロシージャーの使用.....	280
プログラムをコンパイルするための JCL の作成.....	285
TSO のもとでのコンパイル.....	287
例: TSO のもとでコンパイルするための ALLOCATE および CALL.....	288
例: TSO のもとでコンパイルするための CLIST.....	288
アセンブラー・プログラムからコンパイラーを開始する.....	289
コンパイラー入出力の定義.....	291
z/OS のもとでコンパイラーによって使用されるデータ・セット.....	291
ソース・コード・データ・セットの定義 (SYSIN).....	294
コンパイラー・オプション・データ・セットの定義 (SYSOPTF).....	295
ソース・ライブラリーの指定 (SYSLIB).....	296
出力データ・セットの定義 (SYSPRINT).....	296
コンパイラー・メッセージの端末への送信 (SYSTEM).....	296
オブジェクト・コードの作成 (SYSLIN または SYSPUNCH).....	297
関連データ・ファイル (SYSADATA) の定義.....	297
Java ソース出力ファイル (SYSJAVA) の定義.....	297
デバッグ・データ・セットの定義 (SYSDEBUG).....	298

ライブラリー処理出力ファイル (SYSMDECK) の定義.....	298
z/OS のもとでのコンパイラー・オプションの指定.....	299
PROCESS (CBL) ステートメントでのコンパイラー・オプションの指定.....	300
例: JCL によるコンパイラー・オプションの指定.....	301
例: TSO のもとでのコンパイラー・オプションの指定.....	301
z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力.....	301
複数プログラムのコンパイル (バッチ・コンパイル).....	302
例: バッチ・コンパイル.....	303
バッチ・コンパイルでのコンパイラー・オプションの指定.....	304
例: バッチ・コンパイルでのオプションの優先順位.....	305
例: バッチ・コンパイルでの LANGUAGE オプション.....	305
ソース・プログラムのエラーの訂正.....	306
コンパイラー・メッセージのリストの生成.....	307
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.....	307
コンパイラー診断メッセージの形式.....	308
コンパイラー診断メッセージの重大度コード.....	308
 第 16 章 z/OS ユニックス のもとでのコンパイル.....	311
z/OS ユニックス のもとでの環境変数の設定.....	311
z/OS ユニックス のもとでのコンパイラー・オプションの指定.....	312
cob2 コマンドを使用したコンパイルおよびリンク.....	313
z/OS ユニックス のもとでの DLL の作成.....	314
例: z/OS ユニックス のもとでの cob2 によるコンパイルおよびリンク.....	315
cob2 の構文およびオプション.....	315
cob2 入出力ファイル.....	318
スクリプトを使用したコンパイル.....	319
 第 17 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.....	321
z/OS ユニックス のもとでの OO アプリケーションのコンパイル、リンク、実行.....	321
z/OS ユニックス のもとでのオブジェクト指向アプリケーションのコンパイル.....	321
z/OS ユニックス のもとでのオブジェクト指向アプリケーションの準備.....	322
例: z/OS ユニックス の下での COBOL クラス定義のコンパイルおよびリンク.....	323
z/OS ユニックス のもとでのオブジェクト指向アプリケーションの実行.....	324
JCL または TSO/E での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.....	325
JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル.....	326
JCL または TSO/E でのオブジェクト指向アプリケーションの準備と実行.....	327
例: JCL による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、実行.....	328
Java SDKs for z/OS の使用.....	329
オブジェクト指向構文、および Java 6 以降.....	330
 第 18 章 Java と相互運用する非 OO COBOL アプリケーションのコンパイル、リンク、および実行.....	331
相互運用可能な非 OO アプリケーションの構築ジャワ.....	331
COBPROD アプリケーションの紹介.....	332
Java と相互運用する非 OO COBOL アプリケーションをコンパイル、バインド、および実行する ための準備.....	332
ステップ 1: ユーザー COBOL プログラムのコンパイル.....	333
ステップ 2: Java スタブ・プログラムの DLL を構築するための cjbuild ユーティリティーの実行.....	334
ステップ 3: COBOL プログラムのリンク.....	338
ステップ 4: Java プログラムのビルド.....	338
例: COBPROD アプリケーション - ビルドと実行.....	338
COBPROD アプリケーションをビルドして実行するためのサンプル JCL.....	345
例: COBOL 呼び出しジャワアプリケーション - 構築と実行.....	351
COBOL-calls-java アプリケーションをビルドして実行するためのサンプル JCL.....	355
トラブルシューティング.....	360
非 OO COBOL/Java 相互運用可能アプリケーションの実行に関する AMODE の考慮事項.....	361
cjbuild コマンドリファレンス.....	361
 第 19 章 COBOL AMODE 64 アプリケーションのコンパイル、バインド、および実行.....	367

z/OS の下で AMODE 64 プログラムをコンパイルおよびバインド	367
DLL に関する考慮事項.....	368
z/OS ユニックス の下で AMODE 64 プログラムをコンパイルおよびバインド.....	368
COBOL AMODE 64 アプリケーションの実行.....	369
COBOL 専用ランタイム・オプションの指定.....	369
第 20 章 COBOL コンパイラとランタイムオプション.....	371
COBOL コンパイラ・オプション.....	371
85 COBOL 標準に準拠するオプション設定.....	375
矛盾するコンパイラ・オプション.....	376
ADATA.....	378
ADV.....	378
AFP.....	378
APOST/QUOTE.....	379
ARCH.....	379
ARITH.....	381
AWO.....	382
BLOCK0.....	383
BUFSIZE.....	384
CICS.....	384
CODEPAGE.....	385
COMPILE.....	388
コンディション.....	388
COPYLOC.....	389
COPYRIGHT.....	390
CURRENCY.....	390
データ.....	391
DBCS.....	392
DECK.....	393
DEFINE.....	393
DIAGTRUNC.....	394
DISPSIGN.....	395
DLL.....	396
DUMP.....	398
DYNAM.....	398
出口.....	399
EXPORTALL.....	401
FASTSRT.....	402
FLAG.....	402
FLAGSTD.....	403
HGPR.....	405
INITCHECK.....	405
INITIAL.....	407
INLINE.....	407
INTDATE.....	408
INVDATA.....	408
JAVAIOP.....	412
LANGUAGE.....	413
LINECOUNT.....	414
LIST.....	414
LP.....	415
LSACHECK.....	416
MAP.....	418
MAXPCF.....	419
MDECK.....	419
名前.....	421
NSYMBOL.....	421
番号.....	422

NUMCHECK.....	422
NUMPROC.....	426
OBJECT.....	427
OFFSET.....	427
OPTFILE.....	428
OPTIMIZE.....	429
OUTDD.....	430
PARMCHECK.....	431
PGMNAME.....	432
QUALIFY.....	434
RENT.....	435
RMODE.....	436
RULES.....	437
SEQUENCE.....	439
サービス.....	440
SMARTBIN.....	440
SOURCE.....	442
SPACE.....	443
SQL.....	443
SQLCCSID.....	444
SQLIMS.....	445
SSRANGE.....	446
STGOPT.....	447
SUPPRESS.....	448
TERMINAL.....	448
TEST.....	448
THREAD.....	453
TRUNC.....	454
TUNE.....	456
VBREF.....	458
VLR.....	458
VSAMOPENFS.....	461
WORD.....	461
XMLPARSE.....	462
外部参照.....	463
ZONECHECK.....	464
ZONEDATA.....	465
ZWB.....	468
COBOL ランタイム オプション.....	468
COBOL ランタイム オプションの指定.....	470
一意の COBOL ランタイム・メッセージ (IGZUQMSG)	471
レポート COBOL ランタイム・メッセージ (IGZRPTMSG)	472
VSAM 動的アクセス読み取りオプション VSAMDYNAMICDIR.....	475
COBOL ランタイム オプション レポート オプション DISABLEUOPTREPORT を無効にする....	475
IGZUOPT および IGZQUOPT ランタイム オプション制御ブロックの構築.....	476
COBOL IGZUOPT ランタイム オプションの JCL サンプル.....	476
COBOL ランタイム オプション レポート.....	476
第 21 章コンパイラ指示ステートメント.....	481
第 22 章デバッグ.....	485
ソース言語によるデバッグ.....	486
プログラム・ロジックのトレース.....	486
入出力エラーの検出および処理.....	487
データの妥当性検査.....	487
初期化されていないデータの移動、初期化、または設定.....	488
プロシージャに関する情報の生成.....	488
コンパイラ・オプションを使用したデバッグ.....	490

コーディング・エラーの検出.....	490
行シーケンス問題の検出.....	491
無効な COBOL データや無効な COBOL プログラムの検査.....	491
有効範囲の検査.....	492
診断するエラーのレベルの選択.....	492
プログラム・エンティティ定義および参照の検出.....	494
データ項目のリスト.....	495
AMODE 64 プログラムのデバッグ.....	495
デバッガーの使用.....	499
リストの入手.....	500
例: 短縮リスト.....	502
例: SOURCE および NUMBER 出力.....	504
例: MAP 出力.....	505
LIST 出力の読み取り.....	512
例: XREF 出力: データ名相互参照.....	531
例: OFFSET コンパイラ出力.....	536
例: VBREF コンパイラ出力.....	536
例: 条件付きコンパイル出力.....	537
CEEDUMP 処理での情報の抑止 (IGZ1OPT).....	538
COBOL プログラムのレポートの生成.....	538

第 3 部特定の環境に合わせた COBOL プログラムの目標..... 543

第 23 章 COBOL プログラムの開発 (CICS の場合).....	545
CICS のもとで実行する COBOL プログラムのコーディング.....	545
CICS のもとでのシステム日付の取得.....	547
COBOL プログラムとの間の呼び出し.....	547
ECI 呼び出しの成否の判断.....	549
CICS オプションを使用したコンパイル.....	549
CICS サブオプションの分離.....	550
組み込みの CICS 変換プログラム.....	551
分離型の CICS 変換プログラムの使用.....	551
CICS 予約語テーブル.....	553
CICS HANDLE を使用したエラー処理.....	554
例: CICS HANDLE を使用したエラー処理.....	554
第 24 章 Db2 環境用のプログラミング.....	557
Db2 コプロセッサ.....	557
分離型 Db2 プリコンパイラの使用.....	558
SQL ステートメントのコーディング.....	558
Db2 コプロセッサを用いた SQL INCLUDE の使用.....	559
SQL ステートメントでの文字データの使用.....	559
SQL ステートメントでの国別 10 進数データの使用.....	560
SQL ステートメントでの国別グループ項目の使用.....	561
SQL ステートメントでのバイナリ項目の使用.....	561
SQL ステートメントの成否の判断.....	561
SQL オプションを使用したコンパイル.....	561
Db2 サブオプションの分離.....	562
COBOL および Db2 CCSID の決定.....	563
SQL ステートメントのストリング・ホスト変数のコード・ページ決定.....	563
SQLCCSID または NOSQLCCSID オプションを使用したプログラミング.....	564
Db2 のプリコンパイラとコプロセッサの動作方法の相違.....	565
DYNAM または NODYNAM コンパイラ・オプションの選択.....	567
第 25 章 COBOL プログラムの開発 (IMS の場合).....	569
IMS SQL コプロセッサ.....	569
SQLIMS ステートメントのコーディング.....	570

IMS SQL コプロセッサで SQLIMS INCLUDE を使用.....	570
SQLIMS ステートメントでの文字データの使用.....	570
SQLIMS ステートメントでのバイナリー項目の使用.....	571
SQLIMS ステートメントの成否の判断.....	571
SQLIMS オプションを使用したコンパイル.....	571
IMS サブオプションの分離.....	572
IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク.....	573
IMS のもとでのオブジェクト指向 COBOL と Java の使用.....	573
IMS のもとにおける Java アプリケーションからの COBOL メソッドの呼び出し.....	574
COBOL で開始する COBOL と Java の混合アプリケーションの作成.....	575
混合言語 IMS アプリケーションの作成.....	575
 第 26 章 z/OS ユニックス のもとでの COBOL プログラムの実行.....	579
z/OS ユニックス 環境での実行.....	579
環境変数の設定およびアクセス.....	580
実行に影響を与える環境変数の設定.....	581
ランタイム環境変数.....	581
例: 環境変数の設定とアクセス.....	582
UNIX/POSIX API の呼び出し.....	582
z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス.....	584
例: z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス.....	584
 第 27 章 AMODE 64 プログラムの開発.....	587
AMODE 64 プログラムでのポインター・データ項目.....	587
AMODE 64 サポート付きコンパイラ制限.....	588
AMODE 64 サポート付き CALL ステートメント.....	588
コンパイラ・オプションを使用した、AMODE 64 プログラムのコンパイル.....	588
AMODE 64 に関する他の考慮事項.....	590
 第 28 章混合 AMODE 言語環境 での COBOL プログラムの開発.....	593
AMODE 31 の COBOL プログラムと AMODE 64 の Java プログラムとの対話.....	593
AMODE 31 の COBOL プログラムと、AMODE 64 の Java プログラムとの間のアプリケーション呼び出し.....	593
混合 AMODE COBOL/Java インターオペラビリティ・アプリケーションの構築.....	594
混合 AMODE COBOL/Java インターオペラビリティ・プログラムの実行.....	594
混合 AMODE COBOL/Java 相互相互運用性アプリケーションの例.....	595
その他の考慮事項と制約事項.....	596
AMODE 31 および AMODE 64 の混合プログラムを使用する COBOL アプリケーション.....	596
AMODE 31 プログラムと AMODE 64 プログラムの間の動的呼び出し.....	597
COBOL ランタイム・オプション AMODE3164.....	598
その他の考慮事項と制約事項.....	598
 第 4 部複雑なアプリケーションの構造化.....	601
 第 29 章サブプログラムの使用.....	603
メインプログラム、サブプログラム、および呼び出し.....	603
メインプログラムまたはサブプログラムの終了と再入.....	604
別のプログラムへの制御権移動.....	605
静的呼び出しの作成.....	606
動的呼び出しの作成.....	607
AMODE 切り替え.....	609
静的呼び出しと動的呼び出しのパフォーマンスについての 考慮事項.....	611
静的呼び出しと動的呼び出しの両方の作成.....	611
例: 静的および動的 CALL ステートメント.....	612
ネストされた COBOL プログラムの呼び出し.....	613
再帰呼び出しの実行.....	617
オブジェクト指向プログラムとの間での呼び出し.....	617

プロシージャー・ポインターと関数ポインターの使用.....	617
使用するポインター・タイプの決定.....	619
代替入り口点の呼び出し.....	620
プログラムを再入可能にする.....	620
第 30 章データの共用.....	623
データの受け渡し.....	623
呼び出し側プログラムの中での引数の記述.....	625
呼び出し先プログラムの中でのパラメーターの記述.....	626
OMITTED 引数.....	627
LINKAGE SECTION のコーディング.....	627
引数を受け渡すための PROCEDURE DIVISION のコーディング.....	628
受け渡されるデータのグループ化.....	628
ヌル終了ストリングの処理.....	629
ポインターによるチェーン・リストの処理.....	629
戻りコード情報の引き渡し.....	632
RETURN-CODE 特殊レジスターの使用.....	632
PROCEDURE DIVISION RETURNING ... の使用.....	633
CALL ... の指定 RETURNING.....	633
EXTERNAL 節によるデータの共用.....	633
プログラム間でのファイルの共用 (外部ファイル).....	634
例: 外部ファイルの使用.....	634
z/OS 下でのメインプログラム・パラメーターへのアクセス.....	637
例: z/OS 下でのメインプログラム・パラメーターへのアクセス.....	637
第 31 章 DLL または DLL アプリケーションの作成.....	639
ダイナミック・リンク・ライブラリー (DLL).....	639
DLL を作成するためのプログラムのコンパイル.....	640
DLL のリンク.....	641
例: プロシージャー型 DLL アプリケーションのサンプル JCL.....	642
DLL での CALL ID の使用.....	643
z/OS UNIX ファイルシステムでの DLL の探索順序.....	644
DLL リンケージと動的呼び出しの併用.....	645
DLL でのプロシージャー・ポインターまたは関数ポインターの使用.....	645
非 DLL からの DLL の呼び出し.....	646
例: 非 DLL からの DLL の呼び出し.....	646
C/C++ プログラムでの COBOL DLL の使用.....	648
OO COBOL アプリケーションでの DLL の使用.....	648
第 32 章マルチスレッド化のための COBOL プログラムの準備.....	651
マルチスレッド化.....	651
マルチスレッド化サポートのための THREAD の選択.....	653
マルチスレッド化されたプログラムへの制御権移動.....	653
マルチスレッド化されたプログラムの終了.....	653
マルチスレッド化によるファイルの処理.....	654
ファイル定義 (FD) ストレージ.....	655
マルチスレッド化によるファイル・アクセスのシリアライズ.....	655
例: マルチスレッド化によるファイル入出力の使用パターン.....	656
マルチスレッド化による COBOL 制限の処理.....	656
第 33 章ユーザー定義関数の使用.....	659
ユーザー定義関数の作成.....	659
ユーザー定義関数定義のコーディング.....	659
ユーザー定義関数定義のコンパイル.....	660
ユーザー定義関数の呼び出し.....	660
ユーザー定義関数の外部名の指定.....	661
ユーザー定義関数のリンク・エディット.....	661
引数 BY CONTENT をユーザー定義関数に渡します.....	662

第 34 章関数プロトタイプの使用.....	665
関数プロトタイプ of 例.....	665
外部名を持つ関数プロトタイプ.....	665
関数プロトタイプ of ベストプラクティス.....	666
言語標準 of 考慮.....	667
第 5 部 Web サービスに COBOL を使用.....	669
第 35 章 Web サービス・インターフェース.....	671
第 36 章 JSON 入力 of 処理.....	673
JSON 文書 of 構文解析.....	673
有効な COBOL データ名ではない JSON 名をデータ項目に一致させる方法.....	674
データ項目が JSON PARSE ステートメントによって取り込まれないようにする.....	674
JSON 配列 of 処理.....	675
JSON 匿名配列 of 解析.....	676
JSON null 値 of 処理.....	678
JSON PARSE of 例.....	679
第 37 章 JSON 出力 of 生成.....	683
JSON 匿名配列 of 生成.....	684
JSON null 値 of 生成.....	685
第 38 章 XML 入力 of 処理.....	687
COBOL での XML パーサー.....	688
XML 文書へのアクセス.....	689
XML 文書 of 構文解析.....	690
XML を処理するためのプロシージャール of 作成.....	691
XML イベント.....	693
XML テキスト of COBOL データ項目への変換.....	698
検証を伴う XML 文書 of 構文解析.....	698
XML 文書を 1 セグメントずつ構文解析.....	701
XML-INFORMATION 特殊レジスターを使用した分割 of 処理.....	703
XML 文書 of エンコード方式.....	704
XML 入力文書エンコード.....	705
UTF-8 でエンコードされた XML 文書 of 構文解析.....	708
XML PARSE of 例外処理.....	709
XML パーサーによるエラー of 処理方法.....	711
エンコード競合 of 処理.....	712
XML 構文解析 of 終了.....	713
XML PARSE of 例.....	714
例: 単純な文書 of 構文解析.....	714
例: XML の処理用プログラム.....	715
例: 名前空間を使用する XML 文書 of 構文解析.....	719
例: XML 文書を 1 セグメントずつ構文解析.....	722
例: 検証を伴う XML 文書 of 構文解析.....	724
第 39 章 XML 出力 of 生成.....	727
XML 出力 of 生成.....	727
生成される XML 出力 of エンコード of 制御.....	732
XML GENERATE 例外 of 処理.....	732
例: XML の生成.....	733
XML 出力 of 拡張.....	737
例: XML 出力 of 拡張.....	738
第 6 部オブジェクト指向プログラムの開発.....	741

第 40 章オブジェクト指向プログラムの作成.....	743
例: 口座.....	744
クラスの定義.....	746
クラス定義用の CLASS-ID 段落.....	748
クラス定義用の REPOSITORY 段落.....	748
クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION.....	750
例: クラスの定義.....	751
クラス・インスタンス・メソッドの定義.....	751
クラス・インスタンス・メソッド定義用の METHOD-ID 段落.....	752
クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION.....	752
クラス・インスタンス・メソッド定義用の DATA DIVISION.....	753
クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION.....	754
インスタンス・メソッドのオーバーライド.....	755
インスタンス・メソッドの多重定義.....	755
属性 (get および set) メソッドのコーディング.....	756
例: メソッドの定義.....	757
クライアントの定義.....	759
クライアント定義用の REPOSITORY 段落.....	760
クライアント定義用の DATA DIVISION.....	761
オブジェクト参照の比較および設定.....	762
メソッドの呼び出し (INVOKE).....	763
クラスのインスタンスの作成および初期化.....	767
クラスのインスタンスの解放.....	769
例: クライアントの定義.....	769
サブクラスの定義.....	770
サブクラス定義用の CLASS-ID 段落.....	771
サブクラス定義用の REPOSITORY 段落.....	771
サブクラス・インスタンス・データ定義用の WORKING-STORAGE SECTION.....	772
サブクラス・インスタンス・メソッドの定義.....	772
例: サブクラスの定義 (メソッドに関して).....	773
ファクトリー・セクションの定義.....	774
ファクトリー・データ定義用の WORKING-STORAGE SECTION.....	774
ファクトリー・メソッドの定義.....	775
例: ファクトリーの定義 (メソッドに関して).....	777
プロシージャ指向 COBOL プログラムのラッピング.....	782
OO アプリケーションの構造化.....	782
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.....	783
第 41 章 Java メソッドとの通信.....	785
JNI サービスへのアクセス.....	785
Java 例外の処理.....	786
ローカル参照とグローバル参照の管理.....	787
Java アクセス制御.....	789
Java とのデータ共用.....	789
OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ.....	790
Java 用の配列およびストリングの宣言.....	791
Java 配列の取り扱い.....	792
Java ストリングの取り扱い.....	795
例: COBOL で書かれた J2EE クライアント.....	797
例: バッチ COBOL プログラムから Java を呼び出す.....	800

第 7 部オブジェクト指向 (OO) COBOL フレームワークの外部での COBOL/Java 相互運用性.....805

第 42 章既存のプログラムを Java から呼び出し可能にする.....	807

第 43 章 COBOL からの静的 Java メソッドの呼び出し.....	811
第 44 章 COBOL での Java 互換配列型の使用.....	813
バイト配列および汎用グループ・パラメーター.....	813
Java の相互運用可能アプリケーションにおける配列更新のセマンティクスと制限.....	814
第 45 章 Java から COBOL WORKING-STORAGE データ項目へのアクセス.....	815
第 46 章 Java 相互運用可能 COBOL アプリケーションでのエラー処理.....	817
第 47 章 AMODE の考慮事項.....	819
第 8 部特殊処理.....	821
第 48 章 割り込みおよびチェックポイント・リスタート.....	823
チェックポイントの設定.....	823
チェックポイントの設計.....	824
チェックポイントが成功したかどうかのテスト.....	824
チェックポイント・データ・セットの定義用の DD ステートメント.....	825
チェックポイント時に生成されるメッセージ.....	826
プログラムの再始動.....	826
自動再始動の要求.....	827
据え置き再始動の要求.....	827
据え置き再始動の要求用の形式.....	828
再始動用のジョブの再実行依頼.....	829
例: 特定チェックポイント・ステップでのジョブの再始動.....	829
例: ステップ再始動の要求.....	829
例: ステップ再始動のためのジョブの再実行依頼.....	829
例: チェックポイント・リスタートのためのジョブの再実行依頼.....	830
第 49 章 COBOL プログラムからの zlib 圧縮の使用.....	831
第 9 部パフォーマンスおよび生産性の向上.....	835
第 50 章 プログラムのチューニング.....	837
最適なプログラミング・スタイルの使用.....	837
構造化プログラミングの使用.....	838
一括表示表現.....	838
シンボリック定数の使用.....	838
効率的なデータ型の選択.....	839
効率的な計算データ項目の選択.....	839
一貫性のあるデータ型の使用.....	840
算術式の効率化.....	840
指数計算の効率化.....	840
効率的な VOLATILE 節の使用.....	840
テーブルの効率的処理.....	841
テーブル参照の最適化.....	842
コードの最適化.....	843
最適化.....	843
パフォーマンスを向上させるコンパイラ機能の選択.....	844
パフォーマンスに関連するコンパイラ・オプション.....	845
パフォーマンスの評価.....	849
CICS、IMS、または VSAM での効率的な実行.....	850
静的呼び出しまたは動的呼び出しの選択.....	850
第 51 章 コーディングの単純化.....	851

反復コーディングの除去.....	852
例: COPY ステートメントの使用.....	853
言語環境呼び出し可能サービスの使用.....	853
言語環境の呼び出し可能サービスのサンプル・リスト.....	855
言語環境・サービスの呼び出し.....	855
例: 言語環境の呼び出し可能サービス.....	856
形式 2 SORT ステートメントを使用したテーブルのソート.....	857

付録 A 付録..... 859

中間結果および算術精度.....	859
中間結果用の用語.....	860
例: 中間結果の計算.....	860
固定小数点データと中間結果.....	861
固定小数点算術で評価される組み込み関数.....	864
浮動小数点データと中間結果.....	866
非算術ステートメントの算術式.....	867
2 バイト文字セット (DBCS) データの変換.....	867
DBCS の表記.....	868
英数字から DBCS データへの変換 (IGZCA2D).....	868
DBCS から英数字データへの変換 (IGZCD2A).....	870
XML 参照資料.....	872
XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外.....	872
XMLPARSE(COMPAT) が有効な場合の XML PARSE 例外.....	874
XML GENERATE 例外.....	880
JSON 参照資料.....	881
JSON GENERATE 例外.....	881
JSON PARSE 状態、関連コード、およびランタイム・メッセージ.....	882
EXIT コンパイラー・オプション.....	885
ユーザー出口作業域の使用.....	886
出口モジュールからの呼び出し.....	886
INEXIT の処理.....	887
LIBEXIT の処理.....	888
PRTEXIT の処理.....	891
ADEXIT の処理.....	892
MSGEXIT の処理.....	893
出口モジュールでのエラー処理.....	903
CICS、SQL および SQLIMS の各ステートメントで EXIT コンパイラー・オプションを使用する..	904
JNI.cpy コピーブック.....	906
COBOL SYSADATA ファイルの内容.....	909
SYSADATA ファイルに影響するコンパイラー・オプション.....	910
SYSADATA レコード・タイプ.....	911
例: SYSADATA.....	912
SYSADATA レコード記述.....	913
共通ヘッダー・セクション.....	914
ジョブ識別レコード: X'0000'.....	916
ADATA 識別レコード: X'0001'.....	917
コンパイル単位の開始 終了レコード: X'0002'.....	917
オプション・レコード: X'0010'.....	917
外部シンボル・レコード: X'0020'.....	930
構文解析ツリー・レコード: X'0024'.....	932
トークン・レコード: X'0030'.....	959
ソース・エラー・レコード: X'0032'.....	985
ソース・レコード: X'0038'.....	985
COPY REPLACING レコード: X'0039'.....	986
記号レコード: X'0042'.....	986
記号相互参照レコード: X'0044'.....	999
ネストされたプログラム・レコード: X'0046'.....	1001

ライブラリー・レコード: X'0060'.....	1001
統計レコード: X'0090'.....	1002
EVENTS レコード: X'0120'.....	1002
サンプル・プログラムの使用.....	1006
IGYTCARA: バッチ・アプリケーション.....	1006
IGYTCARB: 対話式プログラム.....	1010
IGYTSALE: ネストされたプログラム・アプリケーション.....	1013
示されている言語エレメントおよび概念.....	1020
JCL パラメーターへのアクセス.....	1025
Enterprise COBOL for z/OS のアクセシビリティ機能.....	1026
特記事項.....	1029
商標.....	1031
用語集.....	1033
リソース・リスト.....	1077
Enterprise COBOL for z/OS の資料.....	1077
関連資料.....	1077
索引.....	1081

表

1. FILE-CONTROL 項目.....	6
2. FILE SECTION 記入項目.....	13
3. プログラム内のデータ項目への割り当て.....	27
4. RMODE 属性に対する RMODE および RENT コンパイラー・オプションの影響.....	42
5. COMP-5 データ項目の値の範囲.....	55
6. 数値項目の内部表現.....	56
7. NUMCLS(PRIM) および有効な符号.....	61
8. NUMCLS(ALT) および有効な符号.....	61
9. 算術演算子の評価の順序.....	63
10. 数値組み込み関数.....	64
11. 演算組み込み関数と呼び出し可能サービスの互換性.....	65
12. 日付組み込み関数と呼び出し可能サービスの INTDATE(LILIAN) と互換性.....	66
13. 日付組み込み関数と呼び出し可能サービスの INTDATE(ANSI) と互換性.....	66
14. ユーロ記号の 16 進値.....	71
15. COBOL ステートメントおよび国別データ.....	135
16. 組み込み関数および国別文字データ.....	138
17. 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.....	147
18. 英数字データ、DBCS データ、および国別データのエンコード方式およびサイズ.....	148
19. COBOL ファイルのファイル編成、アクセス・モード、レコード・フォーマットの要約.....	173
20. QSAM ファイル割り振り.....	192
21. QSAM ファイルの最大レコード長.....	195
22. VSAM、COBOL、および非 VSAM 用語の比較.....	201
23. VSAM データ・セット・タイプの比較.....	202

24. VSAM ファイル編成、アクセス・モード、およびレコード・フォーマット.....	204
25. VSAM 可変長レコードの定義.....	208
26. VSAM 可変長レコードの定義.....	208
27. VSAM 順次ファイル用入出力ステートメント.....	210
28. VSAM 相対ファイルおよび索引付きファイル用入出力ステートメント.....	210
29. VSAM ファイルにレコードをロードするためのステートメント.....	213
30. VSAM ファイル内のレコードを更新するためのステートメント.....	215
31. VSAM パフォーマンスを改善する方法.....	225
32. z/OS VSAM と COBOL 用語の比較.....	229
33. VSAMDB プログラムサンプル.....	235
34. NOFASTSORT によるソート・エラーの検査のメソッド.....	258
35. ソート動作を制御するためのメソッド.....	259
36. コンパイラー・データ・セット.....	292
37. 固定長コンパイラー・データ・セットのブロック・サイズ.....	294
38. 可変長コンパイラー・データ・セットのブロック・サイズ.....	294
39. コンパイラ出力の種類 z/OS.....	301
40. コンパイラー診断メッセージの重大度コード.....	308
41. cob2 コマンドへの入力ファイル.....	318
42. cob2 コマンドからの出力ファイル.....	318
43. クラス定義のコンパイルおよびリンクのコマンド.....	322
44. JVM をカスタマイズするための java コマンド・オプション.....	324
45.	335
46. オプションと説明.....	362
47. コンパイラー・オプション.....	371
48. 相互に排他的なコンパイラー・オプション.....	376

49. EBCDIC マルチバイト・コード化文字セット ID.....	387
50. DISPSIGN(COMPAT) オプションまたは DISPSIGN(SEP) を指定した DISPLAY 出力.....	395
51. 以前の COBOL バージョンからマイグレーションする場合の INVDATA オプションと NUMPROC オプションの設定.....	409
52. LANGUAGE コンパイラー・オプションの値.....	414
53. 削除されたオプションから新しいオプションへのマッピング.....	430
54. 読み取られるレコードの長さでファイル状況.....	459
55. COBOL ランタイム オプション.....	468
56. AMODE 31 COBOL ランタイム オプション.....	470
57. AMODE 64 COBOL ランタイム オプション.....	470
58. COBOL ランタイム オプションと対応する JCL サンプル.....	476
59. AMODE 31 ランタイムシナリオの例.....	477
60. AMODE 64 ランタイムシナリオの例.....	478
61. コンパイラー・メッセージの重大度レベル.....	493
62. リストを取得するためのコンパイラー・オプションの使用.....	500
63. MAP 出力で使用する用語.....	509
64. LIST および MAP 出力で使用する記号.....	510
65. INFO BYTE セクションでのコンパイラー・オプション.....	513
66. シグニチャー情報バイト.....	514
67. CICS のもとでの COBOL およびアセンブラー間の呼び出し.....	548
68. 組み込みの CICS 変換プログラムに必要なコンパイラー・オプション.....	549
69. 分離型の CICS 変換プログラムに必要なコンパイラー・オプション.....	552
70. 分離型の CICS 変換プログラムに推奨される TRUNC コンパイラー・オプション.....	552
71. POSIX 関数呼び出しのサンプル.....	583
72. 終了ステートメントの影響.....	604
73. CALL ステートメントでデータを渡す方法.....	624

74. DLL アプリケーションのコンパイラー・オプション.....	640
75. DLL アプリケーションのバインダー・オプション.....	641
76. XML パーサーが使用する特殊レジスター.....	692
77. XMLPARSE(XMLSS) が有効な場合の XML-CODE に対する処理プロシーチャーの変更の結果.....	694
78. 処理手順の変更の結果 XML-CODE と XMLPARSE(COMPAT)事実上.....	695
79. XML 文書のコード化文字セット.....	704
80. 空白文字 (white-space characters) の 16 進値.....	706
81. XML エンコード宣言の別名.....	707
82. さまざまな EBCDIC CCSID 用特殊文字の 16 進値.....	708
83. XML イベントおよび特殊レジスター.....	714
84. XML イベントおよび特殊レジスター.....	719
85. 宣言されていない名前空間接頭部を持つ XML 文書を解析することで得られる XML イベントと特殊レジスター.....	721
86. ENCODING 句を省略した場合に生成される XML のエンコード.....	732
87. クラス定義の構造.....	746
88. インスタンス・メソッド定義の構造.....	751
89. COBOL クライアントの構造.....	759
90. COBOL クライアントでの引数の合致.....	764
91. COBOL クライアントで戻されるデータ項目の合致.....	766
92. ファクトリー定義の構造.....	774
93. ファクトリー・メソッド定義の構成.....	775
94. ローカルおよびグローバル参照の JNI サービス.....	789
95. COBOL および Java で相互運用可能なデータ型.....	790
96. COBOL および Java で相互運用可能な配列およびストリング.....	791
97. COBOL および Java で相互運用可能でない配列型.....	792
98. JNI 配列サービス.....	792

99. jstring 参照と国別データ間の変換サービス.....	795
100. jstring 参照と英数字データ間の変換サービス.....	796
101. パフォーマンスに関連するコンパイラ・オプション.....	845
102. パフォーマンス・チューニング・ワークシート.....	849
103. 言語環境 呼び出し可能サービス.....	855
104. 形式 1 と形式 2 の SORT ステートメントの比較.....	857
105. IGZCA2D 戻りコード.....	869
106. IGZCD2A の戻りコード.....	871
107. Enterprise COBOL に固有な XML PARSE 例外の理由コード.....	873
108. 継続を許可する XML PARSE 例外.....	874
109. 継続を許可しない XML PARSE 例外（のために XMLPARSE(COMPAT)）.....	878
110. XML GENERATE 例外.....	880
111. JSON GENERATE 例外.....	881
112. JSON 非例外状態の理由コード.....	882
113. JSON 例外状態の理由コード.....	883
114. ユーザー出口作業域のレイアウト.....	886
115. INEXIT 処理.....	887
116. INEXIT パラメーター.....	887
117. LIBEXIT 処理.....	888
118. ネストなしの COPY ステートメントによる LIBEXIT の処理.....	889
119. ネストされた COPY ステートメントによる LIBEXIT の処理.....	889
120. LIBEXIT パラメーター.....	890
121. PRTEXTIT 処理.....	891
122. PRTEXTIT パラメーター.....	891
123. ADEXIT 処理.....	892

124. ADEXIT パラメーター	893
125. MSGEXIT の処理	894
126. MSGEXIT パラメーター	894
127. FIPS (FLAGSTD) メッセージのカテゴリ	896
128. CICS、SQL および SQLIMS ステートメントに対して出口モジュールで可能なアクション	904
129. SYSADATA レコード・タイプ	911
130. SYSADATA 共通ヘッダー・セクション	914
131. SYSADATA ジョブ識別レコード	916
132. ADATA 識別レコード	917
133. SYSADATA コンパイル単位の開始 終了レコード	917
134. SYSADATA オプション・レコード	918
135. SYSADATA 外部シンボル・レコード	931
136. SYSADATA 構文解析ツリー・レコード	932
137. SYSADATA トークン・レコード	960
138. SYSADATA ソース・エラー・レコード	985
139. SYSADATA ソース・レコード	985
140. SYSADATA COPY REPLACING レコード	986
141. SYSADATA 記号レコード	986
142. SYSADATA 記号相互参照レコード	1000
143. SYSADATA ネストされたプログラム・レコード	1001
144. SYSADATA ライブラリー・レコード	1001
145. SYSADATA 統計レコード	1002
146. SYSADATA EVENTS TIMESTAMP レコード・レイアウト	1003
147. SYSADATA EVENTS PROCESSOR レコード・レイアウト	1003
148. SYSADATA EVENTS FILE END レコード・レイアウト	1003

149. SYSADATA EVENTS PROGRAM レコード・レイアウト.....	1004
150. SYSADATA EVENTS FILE ID レコード・レイアウト.....	1004
151. SYSADATA EVENTS ERROR レコード・レイアウト.....	1005

前書き

本書について

本書は COBOL プログラマーおよびシステム・プログラマー向けです。Enterprise COBOL for z/OS® を使用して COBOL プログラムをコンパイルする方法を理解するために役立ちます。また本書では、プログラム・パフォーマンスを最適化し、エラーを処理するために必要なオペレーティング・システム機能についても説明します。

COBOL 言語について、および IBM® COBOL コンパイラー用のプログラムを書くために必要な解説書については、「Enterprise COBOL for z/OS 言語解説書」を参照してください。

本情報では、「COBOL」または「Enterprise COBOL」とは、「IBM Enterprise COBOL for z/OS」または「IBM エンタープライズ COBOL バリユーユニット版 z/OS」を指します。

本書の使い方

本書は、Enterprise COBOL プログラムの作成およびコンパイルに役立ちます。

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 Enterprise 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、「IBM Enterprise COBOL for z/OS 言語解説書」を参照してください。

Enterprise COBOL へのプログラムの移行については、「IBM Enterprise COBOL for z/OS 移行ガイド」を参照してください。

IBM z/OS Language Environment® により、Enterprise COBOL プログラムの実行に必要なランタイム環境とランタイム・サービスが提供されます。プログラムのリンク・エディットと実行については、「IBM z/OS 言語環境プログラミングガイド」および「IBM z/OS 言語環境プログラミング・リファレンス」を参照してください。

一般的に使用される Enterprise COBOL および 言語環境 用語の比較については、xxx ページの『一般に使用される用語の比較』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。最もよく使用される製品名の省略形を次の表に示します。

使用される用語	長形式
CICS®	CICS Transaction Server
z/OS デバッガー	IBM z/OS デバッガ (以前の名前は デバッグ・ツール) ¹
Enterprise COBOL	IBM エンタープライズ COBOL z/OS または IBM エンタープライズ COBOL バリユーユニット版 z/OS
言語環境または LE	IBM z/OS 言語環境
MVS	MVS/ESA
z/OS UNIX	z/OS UNIX システムサービス

使用される用語	長形式
注： 1. IBM z/OS デバッガは、以下の製品で提供される共通コンポーネントです。 • IBM Debug for z/OS (旧称: IBM Debug Tool for z/OS) • IBM Developer for z/OS(旧称 IBM Developer for z Systems® および Rational® Developer for z Systems) • IBM Developer for z/OS (Enterprise Edition)。これは、IBM Application Delivery Foundation for z/OS バンドルにも提供されています。 • IBM Z and Cloud Modernization Stack 異なる製品で提供されるデバッグ機能は異なります。詳細情報は、IBM z/OS Debugger の概要 を参照してください。 COBOL アプリケーションを作成するために使用している Enterprise COBOL のレベルに応じて、デバッガを特定のバージョンに更新してください。 • z/OS Debugger xxx 以降は、Enterprise COBOL 6.5 およびそれ以前のバージョンをサポートしています • z/OS Debugger 15.0.5 以降は、Enterprise COBOL 6.4 以前のバージョンをサポートします • z/OS Debugger 14.2 以降は、Enterprise COBOL 6.3 以前のバージョンをサポートします • z/OS Debugger 14.1 は、Enterprise COBOL 6.2 以前のバージョンをサポートします • z/OS Debugger 14.0 は、Enterprise COBOL 6.1 以前のバージョンをサポートします • Debug Tool 13.1 は、Enterprise COBOL 5.1 以前のバージョンをサポートします。	

これらの省略語に加えて、以下の標準の組み合わせをまとめて「85 COBOL 標準」と呼んでいます。

- 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 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

以下の標準を「2002 COBOL 標準」と呼んでいます。

- *INCITS/ISO/IEC 1989-2002, Information technology - Programming languages - COBOL*

以下の標準を「2014 COBOL 標準」と呼んでいます。

- *INCITS/ISO/IEC 1989:2014, Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

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、用語集にリストされています。

一般に使用される用語の比較

IBM z/OS 言語環境および IBM エンタープライズ COBOL z/OS の情報で使用される用語、および同等の用語については、下の表を参照してください。

言語環境 の用語	Enterprise COBOL と同等
集合体	グループ項目

言語環境の用語	Enterprise COBOL と同等
配列	OCCURS 節を使用して作成されたテーブル
配列エレメント	テーブル・エレメント
エンクレーブ	実行単位
外部データ	EXTERNAL 節を使用して定義される WORKING-STORAGE データ
ローカル・データ	非 EXTERNAL データ項目
値によるパラメーターの直接受け渡し	BY VALUE
参照によるパラメーターの間接受け渡し	BY REFERENCE
値によるパラメーターの間接受け渡し	BY CONTENT
ルーチン	プログラム
スカラー	基本項目

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。

▶▶— は、構文図の開始点を示す記号です。

—▶ は、構文図が次の線に続くことを示す記号です。

▶— は、構文図が前の線から続いていることを示す記号です。

—▶◀ は、構文図の終わりを示す記号です。

完全なステートメント以外の構文単位の図は、▶— 記号で始まり、—▶ 記号で終わっています。

- 必須項目は、水平線 (メインパス) と同じ高さに示されます。

▶▶ required_item ▶◀

- オプション項目は、メインパスの下側に示されます。

▶▶ required_item — optional_item —▶◀

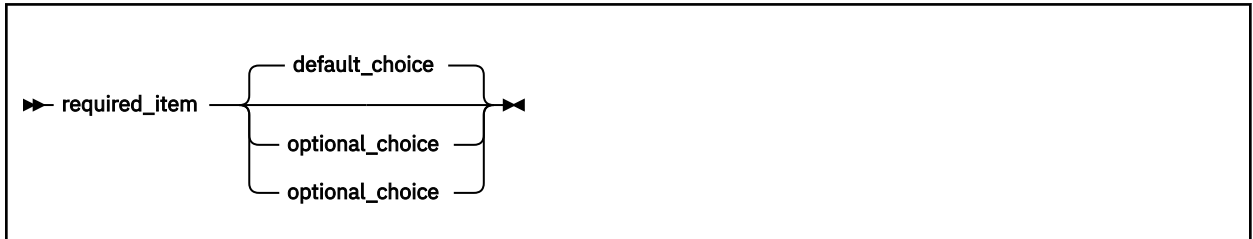
- 複数の項目から選択できる場合には、それらの項目は縦方向に重ねて示されます。それらの項目のうち 1 つを選択しなければならない場合には、スタックのうちの 1 つの項目がメインパス上に示されます。

▶▶ required_item — required_choice1 —▶◀
— required_choice2 —▶◀

項目の選択が任意である場合には、スタック全体がメインパス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、それがメインパスより上に示され、残りの選択項目はメインパスより下に示されます。



- メインラインの上を左に戻る矢印は、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表示されます (例えば、`FROM`)。スペルは、示されているとおりに正確に入力する必要があります。変数は小文字のイタリックで示されます (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。

いくつかの例を説明文からより明確に区別するために、**a monospace font**。

テキスト内の COBOL キーワードとコンパイラ・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。

PDF 形式のドキュメントから例をコピーして貼り付ける場合は、例にスペース (ある場合) が適切に配置されていることを確認してください。COBOL ソーステキストが、COBOL 参照形式の中に *Enterprise COBOL for z/OS* 言語解説書。あるいは、HTML 形式のドキュメントから例をコピーして貼り付けることもできます。その場合、スペースはすでに配置されているはずです。

追加の資料およびサポート

IBM エンタープライズ COBOL z/OS では、このバージョンと以前のバージョンの全ライブラリーの PDF 版をライブラリー・ページ (<https://www.ibm.com/support/pages/enterprise-cobol-zos-documentation-library>) で提供しています。これらの資料は日本語でも入手できます。

また、サポート情報は <https://www.ibm.com/support/pages/node/6560933> に用意されています。

ご意見の送付方法

本書または Enterprise COBOL の他のマニュアルについてご意見がありましたら、IBM 発行のマニュアルに関する情報の Web ページ (<http://www.ibm.com/jp/manuals/>) よりお送りください。今後の参考にさせていただきます。(URL は、変更になる場合があります) この情報またはその他の Enterprise COBOL の資料についてのコメントは、コメントを compinfo@cn.ibm.com に送信していただく。

マニュアルの名前、資料番号、Enterprise COBOL のバージョン、および必要な場合は ご意見のあるテキストの具体的な場所 (ページ番号やセクション見出しなど) を必ずご連絡ください。

情報を送信する場合 IBM、あなたは許可します IBM いかなる方法でも情報を使用または配布する非独占的権利 IBM あなたに対していかなる義務も負うことなく、適切であると判断します。

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

このタスクについて

COBOL プログラムは4つの部門で構成されています。IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、DATA DIVISION、そしてPROCEDURE DIVISION。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

[3 ページの『プログラムの識別』](#)

[5 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』](#)

[18 ページの『データの処理』](#)

[746 ページの『クラスの定義』](#)

[751 ページの『クラス・インスタンス・メソッドの定義』](#)

[782 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。

このタスクについて

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.    05/31/2024.  
Date-Compiled.   05/31/2024.
```

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られます。
- NAME コンパイラー・オプションを使用した場合は、コンパイルによって作成されるオブジェクト・モジュールを識別するために、その名前がNAME バインダー (リンケージ・エディター) の制御ステートメントに入れられます。

ヒント: IBM 製品が使用している接頭部で始まるプログラム名は使用しないでください。名前が以下の接頭部のいずれかで始まるプログラムを使用すると、CALL ステートメントは意図されたプログラムではなく、IBM ライブラリーまたはコンパイラー・ルーチンを呼び出してしまう可能性があります。

- AFB
- AFH

- CBC
- CEE
- CEH
- CEL
- CEQ
- CEU
- DFH
- DSN
- EDC
- FOR
- IBM
- IFY
- IGY
- IGZ
- ILB

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

[5 ページの『ソース・リストのヘッダーの変更』](#)

[4 ページの『プログラムを再帰的として識別する』](#)

[4 ページの『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』](#)

[5 ページの『プログラムを初期状態に設定する』](#)

関連参照

コンパイラー限界値 (*Enterprise COBOL for z/OS 言語解説書*)

プログラム名の規則 (*Enterprise COBOL for z/OS 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。

このタスクについて

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。THREAD オプションを使用してコンパイルするプログラムには、RECURSIVE をコーディングする必要があります。

[15 ページの『例: ストレージ・セクション』](#)を参照してください。これは、再帰的プログラムが WORKING-STORAGE および LOCAL-STORAGE の両方を使用することを示しています。

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用』](#)

[617 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。

このタスクについて

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

[614 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

使用 INITIAL 条項の PROGRAM-ID プログラムが呼び出されるたびに、そのプログラムとネストされたそこに含まれるプログラムは初期状態に置かれます。

このタスクについて

プログラムが初期状態に設定されると、以下のようになります。

- 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

[604 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』](#)

[606 ページの『静的呼び出しの作成』](#)

[607 ページの『動的呼び出しの作成』](#)

関連参照

[407 ページの『INITIAL』](#)

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、コンパイラー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。

このタスクについて

以下の例はこれら 5 つのエLEMENTを示しています。

```
PP 5655-EC6 IBM Enterprise COBOL for z/OS 6.5.0 PXXXXXXX    Date 05/31/2024   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。コンパイラー指示 TITLE ステートメントを使用して、リストの後続ページのヘッダーをカスタマイズすることができます。

注: IBM エンタープライズ COBOL バリユーユニット版 z/OS 製品を使用している場合、ソース・リストの先頭ページのヘッダーは IBM Enterprise COBOL for z/OS のものと同じです。プロダクト番号は 5697-V61 ではなく 5655-EC6 です。Value Unit Edition 製品の 5697-V61 プロダクト番号は製品発注のとき、およびインストール時の製品登録のときにのみ重要となります。

関連参照

TITLE ステートメント (*Enterprise COBOL for z/OS* 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。

このタスクについて

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ためのコンピュータ (SOURCE-COMPUTER 段落)
- プログラムを実行するためのコンピュータ (OBJECT-COMPUTER 段落)
- 通貨記号や記号文字などの特別な項目 (SPECIAL-NAMES 段落)
- ユーザー定義クラス (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、物理的に存在している外部 QSAM、VSAM、または z/OS UNIX ファイルシステム データ・セットに関連付ける。

COBOL 用語のファイルという用語と、オペレーティング・システム (OS) 用語のデータ・セットという用語は、本質的に同じ意味を持ち、ここでの情報では区別なく使用されます。

顧客情報管理システム (CICS) およびオンライン情報管理システム (IMS) のメッセージ処理プログラム (MPP) については、ENVIRONMENT DIVISION ヘッダーと、オプションとして、CONFIGURATION SECTION だけをコーディングしてください。CICS で実行する COBOL プログラム内にファイル定義をコーディングしないでください。IMS では、バッチ・プログラムについてのみ、COBOL によるファイルの定義が許可されます。

- プログラムと外部メディア間でのデータ・レコードの効率的な伝送を制御するために情報を提供する。

6 ページの『例: FILE-CONTROL の記入項目』

関連タスク

7 ページの『照合シーケンスの指定』

8 ページの『シンボリック文字を定義する』

8 ページの『ユーザー定義のクラスを定義する』

9 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定義する』

関連参照

セクションおよび段落 (*Enterprise COBOL for z/OS* 言語解説書)

例: FILE-CONTROL の記入項目

次の表に示す FILE-CONTROL 記入項目の例は、QSAM 順次ファイル、VSAM 索引付きファイル、および行順次ファイル用です。

表 1. FILE-CONTROL 項目		
QSAM ファイル	VSAM ファイル	行順次ファイル
SELECT PRINTFILE ¹ ASSIGN TO UPDPRINT ² ORGANIZATION IS SEQUENTIAL ³ ACCESS IS SEQUENTIAL. ⁴	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-VSAM- STATUS.	SELECT PRINTFILE ¹ ASSIGN TO UPDPRINT ² ORGANIZATION IS LINE SEQUENTIAL ³ ACCESS IS SEQUENTIAL. ⁴

表 1. FILE-CONTROL 項目 (続き)		
QSAM ファイル	VSAM ファイル	行順次ファイル
1. SELECT 節は、外部データ・セットと関連付けられる、COBOL プログラム内のファイルを選択します。 2. ASSIGN 節は、プログラムのファイル名を実際のデータ・ファイルの外部名と関連付けます。外部名は、DD 声明または環境変数。 3. ORGANIZATION 節は、ファイルの編成を記述します。QSAM ファイルの場合、ORGANIZATION 節は任意指定です。 4. ACCESS MODE 節は、レコードを処理する方式 (順次、ランダム、または動的) を定義します。QSAM および行順次ファイルの場合、ACCESS MODE 句はオプションです。これらのファイルの編成は常に順次です。 5. VSAM ファイルの場合、使用する VSAM ファイルのタイプによって、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

[177 ページの『第 9 章 QSAM ファイルの処理』](#)
[201 ページの『第 10 章 VSAM ファイルの処理』](#)
[237 ページの『第 12 章 行順次ファイルの処理』](#)
[5 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。

このタスクについて

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)

[8 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ァベットを基に、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: EBCDIC と同じ照合シーケンスを参照します。
- 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 EBCDIC シーケンスの代替。

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

[254 ページの『代替照合シーケンスの選択』](#)
[148 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。

変更すると EBCDIC シーケンスの SPECIAL-NAMES 段落に含まれる文字の照合順序だけでなく、全体の照合順序が影響を受けます。SPECIAL-NAMES 段落。

```
IDENTIFICATION DIVISION.  
ENVIRONMENT DIVISION.  
CONFIGURATION SECTION.  
Source-Computer. IBM-390.      Object-Computer. IBM-390  
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

[7 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。序数位置を用いて文字を識別してください。位置 1 は文字 X'00' に対応します。

このタスクについて

例えば、バックスペース文字 (EBCDIC アルファベットでは X'16') 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 BACKSPACE IS 23
```

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。

このタスクについて

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。)

オペレーティング・システムに対してファイルを定義する

COBOL プログラムで処理するすべてのファイルについて、適切なシステム・データ定義を使用し、ファイルをオペレーティング・システム (OS) に対して定義する必要があります。

このタスクについて

オペレーティング・システムに応じて、このシステム・データ定義の形式は以下のいずれかになります。

- DD 声明文 MVs JCL。
- TSO のもとでは、ALLOCATE コマンド。
- z/OS または z/OS ユニックス の環境変数。この内容は、MVS データ・セットまたは z/OS UNIX ファイルシステム内のファイルのいずれかを定義することができます。

以下の例は、FILE-CONTROL システムデータ定義へのエントリと FD エントリー FILE SECTION:

- JCL DD ステートメント:

```
(1)
//OUTFILE DD DSN=MY.OUT171,UNIT=SYSDA,SPACE=(TRK,(50,5))
/*
```

- Environment variable (export command):

```
(1)
export OUTFILE=DSN(MY.OUT171),UNIT(SYSDA),SPACE(TRK,(50,5))
```

- COBOL コード:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CARPOOL
    ASSIGN TO OUTFILE (1)
    ORGANIZATION IS SEQUENTIAL.
. . .
DATA DIVISION.
FILE SECTION.
FD CARPOOL (2)
    LABEL RECORD STANDARD
    BLOCK CONTAINS 0 CHARACTERS
    RECORD CONTAINS 80 CHARACTERS
```

(1)

ASSIGN 節の *assignment-name* は、DD ステートメントの *ddname* OUTFILE、または export コマンドの環境変数 OUTFILE を指します。

- //OUTFILE DD DSN=OUT171 . . .、または
- export OUTFILE= . . .

(2)

FILE-CONTROL 記入項目にファイル *file-name* を指定する場合は、次のように、そのファイルを FD 記入項目で記述しなければなりません。

```
SELECT CARPOOL
FD CARPOOL
```

関連タスク

[11 ページの『バッファおよび装置スペースの最適化』](#)

関連参照

[13 ページの『FILE SECTION 記入項目』](#)

FILE SECTION (*Enterprise COBOL for z/OS* 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、そのファイルの名前を実行時に異なるシステム・ファイルに関連付けることができます。

このタスクについて

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。あるいは、DSNAME 値の DD 声明または DSN または PATH 値の export 実行時に別のファイルを使用するコマンド。

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。

ASSIGN 節の *assignment-name* で使用する名前は、DD ステートメントの DD 名または export コマンドの環境変数と同じでなければなりません。

SELECT 節で使用する *file-name* (SELECT MAINFILE など) は、FD *file-name* 記入項目と同じでなければなりません。

2 つのファイルが SELECT 節でそれぞれ同じ ddname または環境変数名を使用することはできません。もし使用した場合、結果は予測できません。例えば、DISPLAY の出力先が SYSOUT になっている場合、ファイルに対する SELECT 節で DD 名または環境変数として SYSOUT を使用しないでください。

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 DD ステートメントまたは export コマンドをコーディングすることで、同じ COBOL プログラムからさまざまなファイルにアクセスすることを示します。

次の SELECT 節を含む COBOL プログラムを考えてみます。

```
SELECT MAINFILE ASSIGN TO DA-3330-S-MAINA
```

3 つのファイル、MAIN1、MAIN2、および MAIN3 が入力ファイルとなる可能性があります。プログラムを実行する前に、プログラム実行を要求するジョブ・ステップに、次の DD ステートメントの 1 つをコーディングするか、または、プログラムを実行する同じシェルから、次の export コマンドの 1 つを発行します。

```
//MAIN1 DD DSNAME=MY.MAIN1,. . .
export MAIN1=DSN(MY.MAIN1),. . .

//MAIN2 DD DSNAME=MY.MAIN2,. . .
export MAIN2=DSN(MY.MAIN2),. . .

//MAIN3 DD DSNAME=MY.MAIN3,. . .
export MAIN3=DSN(MY.MAIN3),. . .
```

したがって、プログラムで MAIN を参照すると、それは現在 DD 名または環境変数名 MAINA に割り当てられているファイルへの参照になります。

この例では、PATH(*path*) 形式の export コマンドを使用して z/OS UNIX ファイルシステム の行順次ファイルを参照できないことに注意してください。行順次ファイルでは編成フィールド (S- または AS-) を指定することができないためです。

バッファおよび装置スペースの最適化

APPLY WRITE-ONLY 節を使用すると、ブロック化可変長レコードの順次ファイルを作成する際に、バッファおよび装置スペースを最適に使用することができます。

このタスクについて

と APPLY WRITE-ONLY 指定すると、次のレコードがバッファの未使用部分に収まらない場合にのみバッファが切り捨てられます。APPLY WRITE-ONLY を指定しない場合は、最大サイズのレコードを収容できる十分なスペースが残っていないと、バッファは切り捨てられます。

APPLY WRITE-ONLY 節が意味を持つのは、ブロック化可変長レコードの順次ファイルの場合だけです。

AWO コンパイラ・オプションは、すべての適格ファイルに暗黙の APPLY WRITE-ONLY 節を適用します。NOAWO コンパイラ・オプションが指定されると、ファイルに APPLY WRITE-ONLY 節が指定されていても、効力はありません。APPLY WRITE-ONLY 節の方が、NOAWO コンパイラ・オプションに優先します。

ファイル処理用に割り当てられたレコード領域があります。ただし、バッファとデバイス スペースの最適化により、レコード領域にはプログラムで処理される実際のレコード データが含まれる場合と含まれない場合があります。

- ファイル処理はレコード領域で行われ、次のいずれかの条件が満たされる場合、プログラムはレコード領域のデータにアクセスできます。
 - ファイルは外部です
 - ファイルは「SAME RECORD AREA IO-CONTROL セクションの「」ステートメント
 - このプログラムは、AWO コンパイラオプション
- ファイル処理は通常、バッファ内で行われ、次の条件がすべて当てはまる場合、レコード領域内のデータはプログラムによってアクセス可能であるとは保証されません (ストレージが未定義)。
 - ファイルは外部ではありません
 - ファイルは「SAME RECORD AREA IO-CONTROL セクションの「」ステートメント
 - このプログラムは、NOAWO コンパイラオプション

したがって、プログラムはファイルのレコード領域内のデータを直接参照しないでください。代わりに、ファイルレコードが読み込まれるときは、WORKING-STORAGE で定義されたデータ項目に読み込まれるか、レコード領域に読み込まれた場合は、プログラムで使用するためにすぐに WORKING-STORAGE のデータ項目に移動される必要があります。例:

```
FD MY-INPUT-FILE
  LABEL RECORDS ARE STANDARD
  RECORDING MODE IS V
  BLOCK CONTAINS 0 RECORDS
  DATA RECORD IS IN-RECORD-AREA.
01 IN-RECORD-AREA PIC X(100).

WORKING-STORAGE.
01 WS-IN-RECORD-AREA PIC X(100).

PROCEDURE DIVISION.
...
  READ MY-INPUT-FILE INTO WS-IN-RECORD-AREA.
```

関連参照

[382 ページの『AWO』](#)

データの記述

データの特性を定義し、データ定義を DATA DIVISION の 1 つ以上のセクションにグループ化します。

このタスクについて

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

の Enterprise COBOL コンパイラは最大サイズを制限します DATA DIVISION 要素。詳細については、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。

関連概念

[14 ページの『WORKING-STORAGE と LOCAL-STORAGE の比較』](#)

関連タスク

[12 ページの『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

コンパイラ限界値 (*Enterprise COBOL for z/OS 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。

このタスクについて

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のフィールドを記述します。
 - レコード全体をレベル 01 記述でコーディングし、次に WORKING-STORAGE SECTION で、レコードのフィールドをより詳しく記述する作業用コピーをコーディングすることができます。使用 READ INTO 記録を WORKING-STORAGE。処理は、WORKING-STORAGE のデータのコピーに対して行われます。処理されたデータは、WRITE FROM ステートメントによって、FILE SECTION で定義されたレコード域に書き込まれます。
 - 設定されるレコード名は、WRITE および REWRITE ステートメントの対象となります。
 - QSAM ファイルの場合のみ、レコード形式を RECORDING MODE 句。RECORDING MODE 節を省くと、コンパイラは RECORD 節およびレベル 01 レコード記述に基づいてレコード形式を判別します。
 - QSAM ファイルの場合は、BLOCK CONTAINS 節でファイルについてのブロック化因数を設定することができます。省略した場合 BLOCK CONTAINS 句が指定されている場合、ファイルはデフォルトでブロック解除されます。しかし、z/OS のデータ管理機能 (DD ファイル・ジョブ制御ステートメントを含む) で、これをオーバーライドできます。

- 行順次ファイルの場合は、BLOCK CONTAINS 節でファイルについてのブロック化因数を設定することができます。コーディングするとき BLOCK CONTAINS 1 RECORDS、または BLOCK CONTAINS *n* CHARACTERS、どこん 1 つの論理レコードの長さですバイト単位、WRITE ステートメントを実行すると、レコードはバッファリングされるのではなく、すぐにファイルに転送されます。この手法は、各レコードを直ちに (エラー・ログなどに) 書き込みたい場合に有用です。

同じ実行単位内のプログラムは、共通ファイルを共用したり、共通ファイルにアクセスすることができます。これを行う方法は、プログラムがネストされた構造 (含まれている構造) の一部であるか、個別にコンパイルされる (バッチ・シーケンスの一部としてコンパイルされるプログラムを含む) か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。

関連概念

[614 ページの『ネストされたプログラム』](#)

関連タスク

[634 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

[13 ページの『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 2. FILE SECTION 記入項目		
節	定義対象	備考
FD	のファイル名参照される PROCEDURE DIVISION 入出力 ステートメント (OPEN、 CLOSE、READ、また START そして DELETEVSAM の場合)	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> を介して <i>ddname</i> と関連付けられます。
BLOCK CONTAINS	物理レコードのサイズ	CHARACTERS 句が指定された場合、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。 QSAM: 指定する場合は、JCL またはデータ・セット・ラベルの情報と一致していなければなりません。もし指定 BLOCK CONTAINS 0、または指定しない場合は、システムが最適なブロックサイズを決定します。 行順次: WRITE ステートメントのバッファリングを制御するために指定できます。 VSAM: 構文が検査されますが、実行に影響はありません。

表 2. FILE SECTION 記入項目 (続き)		
節	定義対象	備考
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。この節を指定する場合、これは JCL またはデータ・セット・ラベルの情報と一致する必要があります。 <i>n</i> が 0 である場合には、JCL またはデータ・セット・ラベルに LRECL をコーディングしなければなりません。
RECORD IS VARYING	論理レコード (可変長) のサイズ	整数サイズ (指定した場合) 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。この節を指定する場合、これは JCL またはデータ・セット・ラベルの情報と一致する必要があります。コンパイラはレコード記述が一致しているか検査します。
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。この節を指定する場合、これは JCL またはデータ・セット・ラベルの情報と一致する必要があります。コンパイラはレコード記述が一致しているか検査します。
LABEL RECORDS	QSAM ファイルのラベル	VSAM: コメントとして処理されます
STANDARD	ラベルが存在する	QSAM: コメントとして処理されます
OMITTED	ラベルが存在しない	QSAM: コメントとして処理されます
データ名	ユーザーによって定義されたラベル	QSAM: (オプションの) テープまたはディスクに許可されます
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
LINAGE	論理ページの深さ	QSAM のみ
CODE-SET	ASCII または EBCDIC ファイル	QSAM のみ。 ASCII ファイルが CODE-SET 条項、対応する DD 声明には DCB=(OPTCD=Q. . .) または DCB=(RECFM=D. . .) ファイルが使用して作成されていない場合はコード化されます VS COBOL II、COBOL 用 OS/390® & VM、または IBM エンタープライズ COBOL z/OS。
RECORDING MODE	物理レコード記述	QSAM のみ

関連参照

FILE SECTION (*Enterprise COBOL for z/OS* 言語解説書)

WORKING-STORAGE と LOCAL-STORAGE の比較

データ項目の割り当てと初期化方法は、項目が WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION。

プログラムの WORKING-STORAGE は、実行単位の開始時に割り振られます。

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 クラス定義の WORKING-STORAGE については、関連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。

スレッド化: 複数のスレッドで同時に実行されるプログラムは、それぞれの起動先が WORKING-STORAGE データの単一コピーへのアクセスを共有します。LOCAL-STORAGE データは起動先ごとに個別のコピーを使用します。

15 ページの『[例: ストレージ・セクション](#)』

関連タスク

604 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](#)』

651 ページの『[第 32 章 マルチスレッド化のための COBOL プログラムの準備](#)』

750 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

WORKING-STORAGE SECTION (*Enterprise COBOL for z/OS* 言語解説書)

LOCAL-STORAGE SECTION (*Enterprise COBOL for z/OS* 言語解説書)

例: ストレージ・セクション

次の例は、WORKING-STORAGE と LOCAL-STORAGE の両方を使用する再帰的プログラムです。

```
CBL pgmn(1u)
*****
* 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
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。別個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。

このタスクについて

関連タスク

[17 ページの『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用』](#)

[623 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用

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。

このタスクについて

呼び出しプログラムでは、CALL . . . USING または INVOKE . . . USING データを渡すためのステートメント。

関連タスク

[623 ページの『データの受け渡し』](#)

[627 ページの『LINKAGE SECTION のコーディング』](#)

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 節を指定できます。の GLOBAL 句は、含まれるプログラム自体にその名前の定義がない限り、データ名が、それを定義するプログラム内に含まれるすべてのプログラムで使用可能であることを指定します。

このタスクについて

ネストされたプログラムは、兄弟プログラム (両方のプログラムを含む同じプログラムにおいて同じネスト・レベルにあるプログラム) におけるデータ項目にはアクセスできませんが、どちらのプログラムも、EXTERNAL 節で定義されたデータ項目や、両方のプログラムを含むプログラムにおけるデータ項目 (このデータ項目が GLOBAL 節で定義されている場合) を参照できます。

関連概念

[614 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。

このタスクについて

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

[651 ページの『マルチスレッド化』](#)

関連タスク

[617 ページの『再帰呼び出しの実行』](#)

[654 ページの『マルチスレッド化によるファイルの処理』](#)

関連参照

[453 ページの『THREAD』](#)

SET ステートメント (*Enterprise COBOL for z/OS 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。

このタスクについて

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
```

必ず定義してくださいデータ名の中に LINKAGE SECTION の DATA DIVISION。

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
```

組み合わせることもできます USING そして RETURNING で PROCEDURE DIVISION ヘッダ：

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

必ず定義してくださいデータ名そして *dataname2* の中に LINKAGE SECTION。

関連概念

[19 ページの『PROCEDURE DIVISION 内でロジックが分割される方法』](#)

関連タスク

[627 ページの『LINKAGE SECTION のコーディング』](#)

[628 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』](#)

[633 ページの『PROCEDURE DIVISION RETURNING ... の使用』](#)

[852 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 (*Enterprise COBOL for z/OS 言語解説書*)

USING 句 (*Enterprise COBOL for z/OS 言語解説書*)

CALL ステートメント (*Enterprise COBOL for z/OS 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、プロシージャー、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、その後に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 ステートメントで始まります。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

[20 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』](#)

[19 ページの『命令ステートメント』](#)

[19 ページの『条件ステートメント』](#)

[22 ページの『宣言部分』](#)

関連参照

PROCEDURE DIVISION 構造 (*Enterprise COBOL for z/OS* 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

[19 ページの『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。
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。
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。たとえば、特定の例外が発生しなかった場合に実行する命令を指定できます。NOT ON SIZE ERROR。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
- I/O 声明文 INVALID KEY、AT END、または AT END-OF-PAGE
- 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

[99 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 (*Enterprise COBOL for z/OS 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、または制御フロー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

481 ページの『第 21 章 コンパイラ指示ステートメント』

C コンパイラ指示ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

範囲終了符号

範囲終了符号はステートメントの終了を示します。明示的または暗黙的な範囲終了符号にすることができます。

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

次のプログラム・フラグメントにおける 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
```

暗黙終了符号を使用すると、ステートメントの終わり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・フラグメントで、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。

プログラムの明確さを向上させ、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してください。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』](#)

宣言部分

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。プロシージャの中に、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

[487 ページの『入出力エラーの検出および処理』](#)

関連参照

宣言 (*Enterprise COBOL for z/OS* 言語解説書)

第 2 章 データの使用

このタスクについて

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。

関連概念

40 ページの『ストレージとそのアドレス可能性』

関連タスク

23 ページの『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』

36 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

38 ページの『組み込み関数の使用』

39 ページの『テーブル (配列) とポインターの使用』

131 ページの『第 7 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ミング言語では、変数 (データ項目)、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。

このタスクについて

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。定義することもできますインデックス名およびデータ項目は次のように記述される USAGE POINTER、USAGE FUNCTION-POINTER、USAGE PROCEDURE-POINTER、または USAGE OBJECT REFERENCE。すべてのデータ定義は DATA DIVISION あなたのプログラムの。

関連タスク

23 ページの『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』

25 ページの『リテラルの使用』

26 ページの『定数の使用』

26 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ー (*Enterprise COBOL for z/OS* 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、データ項目に名前と長さを与えるときに定義されたデータ型に制限されます。

このタスクについて

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
. . .
```

代わりに定義する上記の顧客名は国内または UTF-8 データ項目を指定して PICTURE そして USAGE 条項として Pic N(20)USAGE NATIONAL または Pic U(20) USAGE UTF-8、それぞれ。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。UTF-8 データ項目は Unicode UTF-8 で表されます。Unicode UTF-8 は可変長の Unicode エンコードであり、1 バイトから 4 バイトのストレージを使用して各文字を表します。

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

[151 ページの『COBOL での UTF-8 データ \(Unicode\) の使用』](#)

関連参照

[421 ページの『NSYMBOL』](#)

[148 ページの『文字データの保管』](#)

PICTURE 節 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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、国別グループ項目、UTF-8 グループ項目のいずれでも構いません。

例えば、Customer-Record 以下であり英数字のグループ項目で、2 つから成る従属英数字グループアイテム (Customer-Name そして Part-Order) があり、それぞれに基本データ項目が含まれています。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次のように記述されなければならない。USAGE NATIONAL。

```
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
. . .
```

上記の例では、グループ項目は代わりに USAGE NATIONAL グループ レベルの句。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。指定するグループ USAGE 句は英数字のグループであり、移動や比較などの多くの操作で、基本データ項目のように動作します。USAGE DISPLAY (ただし、データの編集や変換は行われません)。

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

[141 ページの『国別グループ』](#)

関連タスク

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

[145 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』](#)

[148 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 (*Enterprise COBOL for z/OS* 言語解説書)

PICTURE 節 (*Enterprise COBOL for z/OS* 言語解説書)

MOVE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

USAGE 節 (*Enterprise COBOL for z/OS* 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。

このタスクについて

値のデータ項目を定義したり、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 は数字リテラルです。

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

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

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[134 ページの『国別リテラルの使用』](#)

[169 ページの『DBCS リテラルの使用』](#)

関連参照

[421 ページの『NSYMBOL』](#)

リテラル (*Enterprise COBOL for z/OS* 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。

このタスクについて

```
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

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

[168 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。

このタスクについて

例:

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

関連タスク

[140 ページの『国別文字形象定数の使用』](#)

[168 ページの『DBCS サポート用のコーディング』](#)

[156 ページの『UTF-8 文字形象定数の使用』](#)

関連参照

形象定数 (Enterprise COBOL for z/OS 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。

このタスクについて

表 3. プログラム内のデータ項目への割り当て	
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。 • 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。
組み込み関数の結果を割り当てます。	使用 COMPUTE、ADD、SUBTRACT、MULTIPLY、DIVIDE、または MOVE 声明。
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します。
ファイルから値を受け取る。	使用 READ (または READ INTO) 声明。
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します。
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。
以下のいずれかのアクションを行う。 • 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・インスタンスを関連付ける	SET 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)』
- 32 ページの『基本データ項目への値の割り当て (MOVE)』
- 33 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への割り当て (ACCEPT)』
- 111 ページの『データ項目の結合 (STRING)』
- 113 ページの『データ項目の分割 (UNSTRING)』
- 34 ページの『算術結果の割り当て (MOVE または COMPUTE)』
- 120 ページの『データ項目の計算および置換 (INSPECT)』
- 131 ページの『第 7 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
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。この例では、*identifier-1* もっている Usage National。
3. 国番号の 16 進数表記 (UTF-16) 文字 ' ' (空白スペース 3 つ)。 *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 (「4040」) 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	bbbbbbbbb ¹	bbbbABCD ¹
1. 記号 <i>b</i> 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 <i>b</i> 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
      INITIALIZE NATIONAL-1 NATIONAL TO VALUE
```

NATIONAL-3	最初の INITIALIZE の前 の NATIONAL-1	最初の INITIALIZE の後 の NATIONAL-1	2 番目の INITIALIZE の 後の NATIONAL-1
00430044 ¹	00410042 ²	00430044 ¹	00410042 ²
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		

数値編集の初期化 (USAGE DISPLAY) データ項目:

```

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

数値編集の初期化 (USAGE NATIONAL) データ項目:

```

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-EDITED 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

31 ページの『構造の初期化 (INITIALIZE)』

79 ページの『テーブルの初期化 (INITIALIZE)』

49 ページの『数値データの定義』

関連参照

421 ページの『NSYMBOL』

構造の初期化 (INITIALIZE)

INITIALIZE ステートメントを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リセット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。

このタスクについて

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	C00201000000000000000100000000	b000000000000000000000000000000 ¹
1. 記号 <i>b</i> 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をリセットするには、INITIALIZE そのグループ項目へのステートメント。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 の国別文字の各位置には、**NX"0030"** (国別 10 進数のゼロ) が入ります。

使用する場合は **INITIALIZE** 英数字または国別グループ データ項目を初期化するステートメントの場合、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を使用して処理されます。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。コーディングしない場合は **REPLACING** のフレーズ **INITIALIZE** 声明：

- **SPACE** は、英字、英数字、英数字編集、DBCS、カテゴリー国別、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

[141 ページの『国別グループ』](#)

関連タスク

[79 ページの『テーブルの初期化 \(INITIALIZE\)』](#)

[145 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 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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 代数値が移動されます。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、カテゴリー国別または国別編集データ項目に移動でき、送信項目が変換されます。国別データ項目をカテゴリー国別または国別編集のデータ項目に移動できます。カテゴリー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ります。

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。

```
CBL CODEPAGE(00875)
..
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

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[33 ページの『グループ・データ項目への値の割り当て \(MOVE\)』](#)

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

関連参照

[385 ページの『CODEPAGE』](#)

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

MOVE ステートメント (*Enterprise COBOL for z/OS 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。

このタスクについて

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。コンパイラは、各国民グループ項目が国民カテゴリの基本項目であるかのように、つまり各項目が次のように記述されているかのように処理します。PIC N(メートル)、どこメートル国別文字位置でのその項目の長さです。

移動でき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変換します。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。なれ従属データの説明を確認する送信グループと受信グループ項目互換性があります。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。

コーディングすることができます CORRESPONDING フレーズ MOVE あるグループ項目の従属基本項目を、別のグループ項目内の同じ名前の対応する従属基本項目に移動するステートメント。

```
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。その他のデータ項目 Group-N 変更はありません。Filler 受信グループアイテム内のアイテムは、MOVE CORRESPONDING 声明。)

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 句に関する関連した解説書を参照してください。

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

[141 ページの『国別グループ』](#)

関連タスク

[32 ページの『基本データ項目への値の割り当て \(MOVE\)』](#)

[145 ページの『国別グループの使用』](#)

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

関連参照

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

MOVE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

CORRESPONDING 句 (*Enterprise COBOL for z/OS 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。

このタスクについて

```
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・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。を使用する場合 ON SIZE ERROR のフレーズ COMPUTE ステートメントでは、コンパイラはサイズ オーバーフロー状態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。ありません ON SIZE ERROR サポート MOVE 声明。

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ることもできます。例:

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

言語環境の呼び出し可能サービスを使用すれば、日付、時刻、数値その他の計算の結果をデータ項目に割り当てることができます。言語環境サービスは標準の COBOL を通じて利用可能 CALL ステートメントで

返される値は、CALL 声明。例えば、次のステートメントをコーディングして、データ項目の絶対値を見つけるために言語環境のサービス CEESIABS を呼び出すことができます。

```
Call 'CEESIABS' Using Arg, Feedback-code, Result.
```

この呼び出しの結果、データ項目 **Result** には、データ項目 **Arg** の値の絶対値が割り当てられます。データ項目 **Feedback-code** には、サービスが正常に完了したかどうかを示す戻りコードが入ります。特定の呼び出し可能サービスの要件に従って、正しい記述を使用してすべてのデータ項目を **DATA DIVISION** で定義しなければなりません。上記の例の場合は、データ項目を次のように定義することができます。

```
77 Arg          Pic s9(9)  Binary.  
77 Feedback-code Pic x(12) Display.  
77 Result       Pic s9(9)  Binary.
```

関連参照

394 ページの『[DIAGTRUNC](#)』

組み込み関数 (*Enterprise COBOL for z/OS* 言語解説書)

言語環境プログラミング・リファレンス (呼び出し可能サービス)

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。

このタスクについて

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。例:

```
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、次の変更を行います。

- 変化 **Console** に **Device**、どこ **Device** 有効なシステム **Device** (例: **SYSIN**)。例えば:

```
SYSIN is Names-Input
```

device は **z/OS UNIX** ファイルシステムパスを参照する **DD** 名にすることもできます。この **DD** 名が定義されていない場合、プログラムが **z/OS ユニックス** 環境で実行されていれば、**stdin** が入力ソースになります。この **DD** 名が定義されておらず、プログラムが **z/OS ユニックス** 環境で実行されていない場合には、**ACCEPT** ステートメントは失敗します。

使用すると **ACCEPT** ステートメントでは、英数字または国別グループ項目、または基本データ項目に値を割り当てることができます。**USAGE DISPLAY**、**USAGE DISPLAY-1**、または **USAGE NATIONAL**。

値を **USAGE NATIONAL** データ項目に割り当てると、コンソールからの入力データは **CODEPAGE** コンパイラー・オプションで指定された **EBCDIC** コード・ページから国別 (**Unicode UTF-16**) 表現に変換されます。**ACCEPT** ステートメントを使用したときに、国別データの変換が行われるのは、この場合のみです。この場合に変換が行われるのは、画面から入力データがくることが認識されているからです。

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

132 ページの『[Unicode および言語文字のエンコード](#)』

関連タスク

[143 ページの『英数字または DBCS から国別への変換 \(NATIONAL-OF\)』](#)

関連参照

[385 ページの『CODEPAGE』](#)

ACCEPT ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

SPECIAL-NAMES 段落 (*Enterprise COBOL for z/OS 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。

このタスクについて

```
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、次のメッセージをシステム論理出力装置します。

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

データを他の宛先に書き込むにはシステム論理出力デバイス、使用 UPON フレーズ目的地以外 SYSOUT。たとえば、次の文はファイルに書き込みますに指定されている SYSPUNCH DD 声明：

```
Display "Hello" upon syspunch.
```

ファイルを z/OS UNIX ファイルシステムを使用することにより SYSPUNCH DD 声明。例えば、次のように定義すると、DISPLAY 出力はファイル */u/userid/cobol/demo.lst* に書き込まれます。

```
//SYSPUNCH DD PATH='/u/userid/cobol/demo.lst',  
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU,  
// FILEDATA=TEXT
```

次のステートメントは、ジョブ・ログまたはコンソール、および TSO 画面 (TSO のもとで実行している場合) に書き込みを行います。

```
Display "Hello" upon console.
```

USAGE NATIONAL データ項目の値をコンソールに表示するとき、そのデータ項目は、CODEPAGE オプションの値に基づいて Unicode (UTF-16) 表現から EBCDIC に変換されます。DISPLAY ステートメントを使用したときに、国別データの変換が行われるのは、この場合のみです。この場合に変換が行われるのは、出力が画面へ送信されることが認識されているからです。

出力の宛先が別の装置であるときに国別データ項目を変換するには、DISPLAY-OF 組み込み関数 (以下の例を参照) を使用します。

```
01 Data-in-Unicode pic N(10) usage national.  
...  
Display function Display-of(Data-in-Unicode, 00037)
```

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[37 ページの『システム論理出力装置上でのデータの表示』](#)

[37 ページの『WITH NO ADVANCING の使用』](#)

[143 ページの『国別の英数字への変換 \(DISPLAY-OF\)』](#)

[545 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

[385 ページの『CODEPAGE』](#)

DISPLAY ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

システム論理出力装置上でのデータの表示

データをシステム論理出力装置に書き込むには、UPON 節を省略するか、または宛先 SYSOUT を指定した UPON 節を使用してください。

このタスクについて

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

出力は、OUTDD コンパイラー・オプションで指定された DD 名に送られます。z/OS UNIX ファイルシステム内のファイルはこの DD 名で指定できます。

もし、OUTDD ddname が割り当てられておらず、実行されていません z/OS ユニックス環境、デフォルト DD の SYSOUT=* が割り当てられます。OUTDD という DD 名が割り振られていないときに、プログラムが z/OS ユニックス 環境で実行されていれば、_IGZ_SYSOUT 環境変数が次のように使用されます。

未定義または stdout に設定されている

出力は stdout に送られます (ファイル記述子 1)。

stderr に設定されている

出力は stderr に送られます (ファイル記述子 2)。

それ以外 (stdout または stderr 以外に設定されている)

の DISPLAY ステートメントが失敗する;severity-3 言語環境条件が発生します。

DISPLAY 出力が stdout または stderr に経路指定された場合、出力はレコード単位に分割されません。出力は改行のない単一の文字ストリームとして書き込まれます。

もし OUTDD そしてその言語環境ランタイムオプション MSGFILE 同じ DD 名を指定すると、DISPLAY 出力と言語環境ランタイム診断は、言語環境メッセージ ファイル。

関連タスク

[580 ページの『環境変数の設定およびアクセス』](#)

関連参照

[430 ページの『OUTDD』](#)

DISPLAY ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

WITH NO ADVANCING の使用

WITH NO ADVANCING 句を指定し、出力が DD 名に送られる場合は、印刷制御文字 + (正符号) が次の DISPLAY ステートメントの最初の出力位置に入れます。+ は ANSI 定義の印刷制御文字であり、これはレコード印刷前の行送りを抑止します。

このタスクについて

指定する場合は WITH NO ADVANCING フレーズと出力が stdout または stderr に送信される場合は、ストリームの末尾に改行文字は追加されません。その後の DISPLAY ステートメントがストリームの終わりに文字を追加する場合があります。

指定しない場合 WITH NO ADVANCING、出力が DD 名に送られる場合、プリンタ制御文字「」（スペース）が次の出力位置から最初の出力位置に配置されます。DISPLAY 単一スペースの出力を示すステートメント。

```
DISPLAY "ABC"  
DISPLAY "CDEF" WITH NO ADVANCING  
DISPLAY "GHIJK" WITH NO ADVANCING  
DISPLAY "LMNOPQ"  
DISPLAY "RSTUVWX"
```

上記のステートメントをコーディングした場合は、以下の結果が出力装置に送られます。

```
ABC  
CDEF  
+GHIJK  
+LMNOPQ  
RSTUVWX
```

印刷される出力は、出力装置が印刷制御文字をどのように解釈するかによって異なります。

WITH NO ADVANCING 句を指定しなかった場合、出力が stdout または stderr に送られるときは、ストリームの終わりに改行文字が追加されます。

関連参照

DISPLAY ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

組み込み関数の使用

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。

このタスクについて

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。(必要に応じて、関数名が REPOSITORY 段落で参照されている場合は予約語 FUNCTION を省略できます。) 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。また、関数 ID を REPOSITORY 段落内の ENVIRONMENT DIVISION で使用することもできます。

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、サブストリング指定 (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 と MIN 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

[94 ページの『組み込み関数を使用したテーブル項目の処理』](#)

[122 ページの『データ項目の変換 \(組み込み関数\)』](#)

[126 ページの『データ項目の評価 \(組み込み関数\)』](#)

関連参照

REPOSITORY 段落 (*Enterprise COBOL for z/OS* 言語解説書)

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。テーブルとは、論理的に連続したデータ項目のセットであり、DATA DIVISION を使用することにより OCCURS 句。

このタスクについて

ポインターは、仮想記憶アドレスを含むデータ項目です。明示的に定義するか、USAGE IS POINTER 条項の DATA DIVISION または暗黙的に ADDRESS OF 特殊レジスタ。

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。

- プログラム間で受け渡しするには、CALL . . . BY REFERENCE 声明。
- ALLOCATE ステートメントおよび FREE ステートメントを使用して、割り振られたストレージやフリー・ストレージを向くようにポインターを設定する。
- 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。
- 無効なアドレスを含むように初期化するには、VALUE IS NULL。

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。

- 限定されたベースアドレス指定を実現する。特に、次のように定義されたレコード領域のアドレスを渡したり受け取ったりしたい場合に有効である。OCCURS DEPENDING ON したがって、位置はさまざまです。
- チェーン・リストの処理。

関連タスク

[73 ページの『テーブルの定義 \(OCCURS\)』](#)

[617 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』](#)

ストレージとそのアドレス可能性

COBOL プログラムを実行するときは、プログラムおよびプログラムで使用されるデータは仮想記憶域にあります。COBOL で使用するストレージは、16 MB 境界より下にすることも、16 MB 境界より上で 2 GB 境界よりも下にすることも、2 GB 境界より上にすることもできます。このストレージをアドレッシングするために 3 つのアドレッシング・モード (24 ビット、31 ビット、および 64 ビット) を使用できます。

24 ビット・アドレッシングを使用して、16 MB 境界より下のストレージをアドレッシングすることができます (ただし、16 MB 境界より上のアドレッシングはできません)。31 ビット・アドレッシングを使用すると、16 MB 境界より上と 16 MB 境界より下のどちらのストレージでもアドレッシングできます。31 ビット・アドレッシングで無制限のストレージをアドレッシングできるため、2 GB 境界より下でプログラムから使用できるストレージをすべてアドレッシングできます。

Enterprise COBOL 現在は、z/OS LP (64) コンパイラー・オプションで 64 ビット仮想アドレス機能を直接利用できるようになったが、31 ビットまたは 24 ビット・アドレス・モードで動作する COBOL アプリケーションは、64 ビット z/OS システム上で完全にサポートされています。

アドレッシング・モード (AMODE) は、ご使用のプログラムでサポートされているハードウェア・アドレッシング・モードを示す属性 (24 ビット・アドレッシング、31 ビット・アドレッシング、24 ビット/31 ビット・アドレッシング、および 64 ビット・アドレッシング) です。これらの属性は、それぞれ AMODE 24、AMODE 31、AMODE ANY、および AMODE 64 です。プログラム・オブジェクト、および実行中のプログラムには、それぞれ AMODE 属性があります。Enterprise COBOL 6 オブジェクト・プログラムは、AMODE MIN AMODE 24 が可能な場合は、AMODE 31、または AMODE 64 のいずれかです。[41 ページの『AMODE の制約事項』](#)を参照してください。

常駐モード (RMODE) は、仮想記憶域のどこ (16 MB 境界の下、または 16 MB 境界の下か上) にプログラムが常駐するのかを識別するプログラム・オブジェクトの属性です。この属性は、RMODE 24 または RMODE ANY です。

注: Enterprise COBOL 6 は、RMODE 64 をサポートしません。

Enterprise COBOL は、言語環境のサービスを使用して、実行時に使用されるストレージを制御します。したがって、COBOL コンパイラー・オプションおよび言語環境 ランタイム・オプションは、単独でまたは組み合わせによって、プログラムとデータの AMODE 属性および RMODE 属性に影響を与えます。

DATA

RENT でコンパイルされたプログラムの WORKING-STORAGE データ、I-O バッファ、およびパラメーター・リストのストレージの場所に影響を与えるコンパイラー・オプション。

RMODE

常駐モードに影響を与えるコンパイラー・オプション。

RENT

再入可能プログラムを生成するためのコンパイラー・オプション。

HEAP

ランタイム・ヒープのストレージを制御するランタイム・オプション。例えば、COBOL WORKING-STORAGE は、COBOL プログラムが RENT オプションを使用してコンパイルされていて以下のいずれかのケースに相当する場合にヒープ・ストレージから割り振られます。

- 企業 COBOL 4.2 またはそれ以前のリリースでコンパイル済み
- DATA (24) コンパイラー・オプションを指定してコンパイル
- CICS での実行
- COBOL プログラム (COBOL 5.1.0 を除く) およびアセンブラを含むプログラム・オブジェクト内の COBOL 5.1.1 またはそれ以降のプログラム。プログラム・オブジェクト内に言語環境プログラム 相互言語間の呼び出しはなく、COBOL 5.1.0 プログラムもありません。
- プログラム・オブジェクト内の COBOL 5 プログラムで、メイン・エン트리・ポイントが COBOL 5 である場合。この場合、プログラム・オブジェクトには、COBOL が C、C++、または PL/I と静的にリンクする言語環境プログラム 言語間呼び出しを含めることができます。そのようなプログラム・オブジェクト内のすべての COBOL 5 プログラム (それらがメイン・エン트리・ポイントではない場合でも) には、ヒープ・ストレージから割り振られた WORKING-STORAGE があります。

- COBOL 6.1 またはそれ以降のプログラム

STACK

ランタイム・スタックのストレージを制御するランタイム・オプション。例えば、COBOL LOCAL-STORAGE は、スタック・ストレージから割り振られます。

ALL31

アプリケーションを完全に AMODE 31 で実行できるかどうかを指定するランタイム・オプション。

AMODE の制約事項

AMODE 24 の実行は、以下のケースではサポートされていません。アプリケーションは AMODE 31 で実行する必要があります。これは、COBOL 3 および 4 と同じ一連の AMODE 24 制限です。

- XML PARSE ステートメントを含むプログラム
- XML GENERATE ステートメントを含むプログラム
- C プログラム、C++ プログラム、または PL/I プログラムにバインドされた COBOL を含み、静的 CALL を通じて通信を行うプログラム・オブジェクト
- オブジェクト指向言語構文 (INVOKE ステートメントなど) またはオブジェクト指向クラス定義を含むプログラム
- 以下のいずれかのコンパイラー・オプションを使用してコンパイルされたプログラム
 - DLL
 - PGMNAME (LONGUPPER)
 - PGMNAME (LONGMIXED)
- マルチスレッド・アプリケーション

注: THREAD オプションを指定してコンパイルされたプログラムは、AMODE 24 で実行できますが、複数のスレッドまたは PL/I タスクを持たないアプリケーションでのみ実行できます。

- z/OS UNIX ファイル・システムから実行されるプログラム

注: z/OS UNIX ファイル・システムに常駐する AMODE 31 ドライバー・プログラムには、MVS PDSE に常駐する AMODE 24 プログラム・モジュールに対する動的呼び出しを含むことができます。

- EXIT コンパイラー・オプションで指定された COBOL コンパイラー出口モジュールとして使用されるプログラム
- XPLINK を使用する 言語環境プログラム エンクレーブ。XPLINK コンパイラー・オプションを使用してコンパイルされた非 COBOL プログラムを含めるエンクレーブ、または XPLINK ランタイム・オプションを指定して実行するエンクレーブのいずれかを含みます。

注:

- COBOL プログラムをアドレッシング・モード 24 で実行するには、すべての COBOL プログラムを、Enterprise COBOL 5.1.1 以降のバージョン、または Enterprise COBOL 4.2 以前のバージョンを使用してコンパイルする必要があります。プログラム・オブジェクトに含まれるいずれかのコンポーネントが Enterprise COBOL 5.1.0 を使用してコンパイルされたものである場合、そのプログラム・オブジェクトはアドレッシング・モード 31 で実行されなければなりません。アドレッシング・モード 24 を使用して実行する COBOL プログラムは、バインダー・オプション RMODE (24) を使用してリンクする必要があります。
- COBOL プログラムをアドレッシング・モード 64 で実行するには、すべての COBOL プログラムを、Enterprise COBOL 6.3 以降のバージョンを使用してコンパイルし、LP (64) コンパイラー・オプションを使用する必要があります。

RMODE の設定

RMODE オプションと RENT オプションによって、プログラムの RMODE 属性が決まります。

表 4. RMODE 属性に対する RMODE および RENT コンパイラー・オプションの影響		
RMODE コンパイラー・オプション	RENT コンパイラー・オプション	RMODE 属性
RMODE (AUTO)	RENT	RMODE ANY
RMODE (AUTO)	NORENT	RMODE 24
RMODE (24)	RENT または NORENT	RMODE 24
RMODE (ANY)	RENT	RMODE ANY
RMODE (ANY)	NORENT	コンパイラー・オプションが矛盾しています。 NORENT オプションが以下の場合、指定、RMODE (24) または RMODE (AUTO) コンパイラー オプションを指定する必要があります。

リンク・エディットに関する考慮事項: COBOL が生成するオブジェクト・コードに属性 RMODE 24 がある場合には、RMODE 24 でオブジェクト・コードをリンク・エディットする必要があります。COBOL が生成するオブジェクト・コードの属性が RMODE ANY の場合は、それを RMODE ANY または RMODE 24 でリンク・エディットすることができます。

データの受け渡しに関するストレージ制限

16 MB 境界より上のストレージに割り振られたパラメーターを、AMODE 24 サブプログラムに渡さないでください。31 ビット・アドレッシング・モードで実行されるプログラムの場合は、WORKING-STORAGE データおよびパラメーター・リストを強制的にラインより下に置き、AMODE 24 で実行されるプログラムにデータを渡します。

- RENT および DATA (24) コンパイラー・オプションを指定してコンパイルするか、WORKING-STORAGE が HEAP 上にある場合 (HEAP オプションの前の説明を参照) は、HEAP (, BELOW) ランタイム・オプションを指定して実行します。
- NORENT コンパイラー・オプションを指定してコンパイルします。

データ域のロケーション

再入可能プログラムの場合、DATA コンパイラー・オプション、および HEAP ランタイム・オプションは、WORKING-STORAGE SECTION および FD レコード域などのデータ域のストレージを 16 MB ラインより下から取得するか、または無制限のストレージから取得するかを制御します。プログラムが、16 MB ラインより上の仮想ストレージ・アドレスで、31 ビット・アドレッシングを使用して実行される場合は、RENT および RMODE (ANY) または RMODE (AUTO) を使用してプログラムをコンパイルします。DATA オプションは、NORENT でコンパイルされたプログラムには影響しません。

LOCAL-STORAGE データのストレージ

LOCAL-STORAGE データ項目の場所は、STACK ランタイム・オプションとプログラムの AMODE によって制御されます。STACK (, ANYWHERE) ランタイム・オプションが有効で、プログラムが AMODE 31 で実行されている場合、LOCAL-STORAGE データ項目は無制限のストレージで獲得されます。それ以外の場合、LOCAL-STORAGE は、16 MB 境界より下で獲得されます。DATA コンパイラー・オプションは、LOCAL-STORAGE データの場所には影響しません。

外部データに対するストレージ

動的データ域 (WORKING-STORAGE、FD レコード域、およびパラメーター・リスト) 用にストレージを取得する方法に影響を与えるだけでなく、DATA コンパイラー・オプションは、EXTERNAL データ用のストレージを取得する場所にも影響を与えることができます。EXTERNAL データに必要なストレージは、以下の条件が満たされている場合に、無制限のストレージから取得されます。

- プログラムは、DATA(31) および RENT コンパイラー・オプション を指定してコンパイルされます。
- HEAP(, ANYWHERE) ランタイム・オプションが有効である。
- ALL31(ON) ランタイム・オプションが有効である。

それ以外の場合はすべて、EXTERNAL データ用のストレージは 16 MB 境界より下から取得されます。

ALL31(ON) ランタイム・オプションを指定する場合は、実行単位内のすべてのプログラムが、31 ビット・アドレッシング・モードで実行できなければなりません。

QSAM 入出力バッファ用のストレージ

DATA コンパイラー・オプションは、QSAM ファイルの入出力バッファが取得される場所にも影響を与えることができます。QSAM ファイルのバッファの割り振りおよび DATA コンパイラー・オプションについては、以下の関連文献を参照してください。

ALLOCATE ステートメントのストレージ

DATA コンパイラー・オプション設定は、ALLOCATE でストレージを取得する方法に影響します。

- もし DATA(24)が有効であり、LOC 31 のフレーズ ALLOCATE 文が指定されていない、ALLOCATE 16 MB ラインより下からストレージを取得します。
- もし DATA(31)が有効であり、LOC 24 のフレーズ ALLOCATE 文が指定されていない、ALLOCATE 16 MB ラインより上のストレージを取得しようとします。

関連概念

[609 ページの『AMODE 切り替え』](#)

ヒープ・ストレージに関する AMODE の考慮事項
(言語環境プログラミングガイド)

関連タスク

[603 ページの『第 29 章 サブプログラムの使用』](#)

[623 ページの『第 30 章 データの共用』](#)

関連参照

[197 ページの『QSAM ファイル用のバッファの割り振り』](#)

[225 ページの『VSAM ファイル用のレコード域の割り振り』](#)

[391 ページの『データ』](#)

[435 ページの『RENT』](#)

[436 ページの『RMODE』](#)

[845 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』](#)

ALLOCATE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

HEAP, STACK, ALL31 (言語環境プログラミング・リファレンス)

MVS プログラム管理: ユーザーズ・ガイドおよび解説書

ユーザー定義型

COBOL におけるユーザー定義型は、TYPEDEF 句で定義されるグループ項目またはエレメンタリー項目である。ユーザー定義型は、データ記述項目と構造が対応する型と同一である 1 つ以上のデータ項目を作成するためのテンプレートと見なすことができる。この型定義から作成されるデータ項目は「型インスタンス」と呼ばれる。COBOL では、型インスタンスは TYPE 句を使って定義される。

TYPEDEF 句がグループ項目の定義に現れると、その結果の型は「グループ型」と呼ばれる。

TYPDEF 句が初等項目の定義に現れると、その結果の型は「初等型」と呼ばれる。

注: エンタープライズ COBOL で使用可能な型付け項目は、ISO COBOL 標準では「弱型付け項目」と呼ばれる。なぜなら、これらの型の名前は、プログラム中のどこに出てきても、関連する型定義の省略形または単なる「ショートハンド」として扱われるからであり、また、この種の型付きオペランド間の型適合性は、さまざまな操作において強制されないからである。

例えば、これらの型のインスタンスを使用する場合、MOVE 文の送信者と受信者の間、CALL 文の引数とパラメータの間、比較演算の 2 つのオペランドの間では、型適合性は強制されない。

ISO COBOL 標準は、型のインスタンスに対して型適合性検査を実行することを示すために、TYPDEF 句のオプションの STRONG 句を提供している。これらのインスタンスは「強く型付けされたアイテム」と呼ばれる。TYPDEF STRONG は、現在 Enterprise COBOL ではサポートされていない。

ユーザー定義型には次のような利点がある：

- 特定のユースケース用にカスタマイズされたデータ構造をきれいに定義できるようにする。
 - COPY で使用するデータ定義に置換タグ（例：TAG:）を含める必要はない。REPLACING 操作は、型インスタンスの生成をシミュレートするためのもので、ユーザー定義型が存在する以前の一般的なパターンである。
- 複数のモジュールにまたがるグループの一貫した定義とフィールド名の一貫した使用を保証します。
- データの整合性を高める。

例初等型を定義し、それをインスタンス化する

この例では、初等型の定義と、その型の複数のインスタンス化を示す。

```
* elementary type definition
01 MY-STRING-T TYPDEF PIC X(20) JUSTIFIED.

* type instance definitions
01 STR1 TYPE MY-STRING-T.
01 STR2 TYPE MY-STRING-T.

* define elementary type instance within a group
01 G1.
   03 S1 TYPE MY-STRING-T.
;
PROCEDURE DIVISION.
   MOVE STR1 TO STR2
   MOVE 'Hello, world!' TO S1
```

この例では、MY_STRING-T が型名である。

例グループ・タイプの定義とインスタンス化

この例は、それ自体が以前に定義された初等型のインスタンスを含むグループ型の定義を示す。

```
* elementary type definition
01 MY-STRING-T TYPDEF PIC X(20) JUSTIFIED.

* group type definition
01 MY-GROUP-T TYPDEF.
   03 S1 TYPE MY-STRING-T. *> elementary type instance
   03 N1 PIC S9(9).

* type instance definitions
01 G1 TYPE MY-GROUP-T. *> G1 is a group item
01 G2 TYPE MY-GROUP-T. *> G2 is a group item
;
PROCEDURE DIVISION.
   MOVE S1 OF G1 TO S1 OF G2
   COMPUTE N1 OF G1 = (N1 OF G1 + N1 OF G2) * 2
```

TYPEDEF 文節

TYPEDEF 句はデータ記述項目の中に現れ、型を定義するために使われる。TYPEDEF 句を持つ項目名は型名とみなされる。

フォーマット



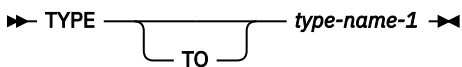
基本ルール

- TYPEDEF 句は 01 レベルと 77 レベルの項目にのみ指定できる。TYPEDEF 句を含む 01 レベル項目は、エレメンタリー項目またはグループ項目 ("エレメンタリー型"または"グループ型") とすることができる。
- TYPEDEF 句がデータ記述項目に現れる場合は、型名の後の最初の句でなければならない。
- TYPEDEF 句を持つデータ記述項目は、名前の変更や再定義はできない。
- EXTERNAL 句も VOLATILE 句も、TYPEDEF 句とは互換性がない。
- GLOBAL が TYPEDEF とともに使われる場合、それは型そのものに適用され、型のインスタンスには適用されない (型はスコープされる)。
- TYPEDEF を使った型の定義は、それが参照される前にデータ部に記述されていなければならない (前方参照は許されない)。
- TYPEDEF 句は、データ部のどのセクションのデータ記述項目にも使用できる。それ以降は、異なるセクションを含め、データ部のどこからでも参照できる。
 - FILE セクションで定義されたタイプは、FD または SD の下で定義されなければならない。特定の FD や SD の下で定義されることは重要ではない。重要なのは、TYPE 句で参照される前のセクションに型の定義があることだ。
- 同じグループ・タイプに含まれるデータ項目を参照する OCCURS DEPENDING ON オブジェクトがテーブルにある場合、グループ・タイプはそのデータ記述項目に OCCURS DEPENDING ON 節を持つテーブルを含むことはできません。

TYPE 節

TYPE 句は、データ記述の項目に現れ、定義される項目が指定された型のインスタンスであることを示す。
type-name-1.

フォーマット



基本ルール

- TYPE 句は、データ記述項目のデータ名の後であればどこにでも書くことができる。
- TYPE 句を含むデータ記述項目には、以下の句を含めることはできない (これらの属性は、型自体に由来することが期待される) :
 - PICTURE
 - 利用方法
 - SIGN
 - BLANK WHEN ZERO
 - DYNAMIC
 - JUSTIFIED
 - SYNCHRONIZED

- TYPE 句で許される句：
 - OCCURS
 - 値
 - EXTERNAL
 - VOLATILE
 - GLOBAL
- 型は、TYPE 句で使う前に定義しておく必要がある。
- TYPE 句を含むデータ記述項目は、level-88 項目を含む下位項目を許可しない。下位の定義は、型自体の中で定義されることが期待される。
- 77 レベル項目に現れる TYPE 句は、グループタイプを参照することはできない。

TYPE 定義における VALUE 句とその他の句の使用

TYPE 句を含むデータ記述項目で VALUE 句が指定されている場合、対応する型定義に同じレベルの VALUE 句が含まれていれば、それが上書きされる。

例 VALUE 句の上書きと TYPE による追加指定

```
* elementary type definition
01 MY-STRING-T TYPEDEF PIC X(20) JUSTIFIED
  VALUE 'Init value 0'.

* type instance definitions override VALUE
* clause in type definitions
01 STR1 TYPE MY-STRING-T VALUE 'Init value 1'.
01 STR2 TYPE MY-STRING-T.
01 G1.
03 STR3 TYPE MY-STRING-T OCCURS 10 TIMES.
:
PROCEDURE DIVISION.
  DISPLAY 'STR1 = ' STR1 *> ' Init value 1'
  DISPLAY 'STR2 = ' STR2 *> ' Init value 0'
  MOVE 'Hello' TO STR3(5)
```

注：STR1 なぜなら、MY-STRING-T 型のインスタンスとして、その定義で指定された VALUE 句が、型自体の中で定義された VALUE 句よりも優先されるからである。

STR1 一方、MY-STRING-T の VALUE 句はオーバーライドされないで、「初期値 0」となる。

STR3 は、MY-STRING-T 型のインスタンスとして、その定義に OCCURS 節を含んでいる。その結果、英数字要素のテーブルが作成され、各要素は「初期値 0」に初期化される。

例同じデータ記述項目で TYPEDEF 句と TYPE 句を使用する場合

TYPEDEF 句と TYPE 句は、データ記述項目で一緒に使用することができる。これは、以前に定義された型をベースとする新しい型を定義する際に行われる。

```
* define an elementary type
01 MY-DECIMAL-T TYPEDEF PIC S7(9)V9(2) COMP-3.

* define a type that is a table based on MY-DECIMAL-T
01 MY-DECIMAL-LIST-T TYPEDEF TYPE MY-DECIMAL-T OCCURS 10 TIMES.
01 DECLIST TYPE MY-DECIMAL-LIST-T.
:
PROCEDURE DIVISION.
  MOVE 12.23 TO DECLIST(5)
```

例 level-88 アイテムを初等型で扱う

初等型の型インスタンスは、level-88 項目を含め、従属項目を持つことはできない。level-88 の項目を初等型と共に使用するには、初等型の定義自体に level-88 の項目を記述する。

```
* WRONG WAY

* Attempt level-88 items on elementary type instance
01 MY-ENUM-T TYPEDEF PIC 9(9) BINARY.

* type instance definition
01 NUM1 TYPE MY-ENUM-T.
```



```

88 VAL1 VALUE 1. *> not allowed
88 VAL2 VALUE 2. *> not allowed
:
PROCEDURE DIVISION.
    SET VAL1 OF NUM1 TO TRUE

```

```

* CORRECT WAY

* Put level-88 items in elementary typedef
01 MY-ENUM-T TYPEDEF PIC 9(9) BINARY.
    88 VAL1 VALUE 1. *> (it_is_allowed!)
    88 VAL2 VALUE 2. *> (it_is_allowed!)

* type instance definition
01 NUM1 TYPE MY-ENUM-T.
:
PROCEDURE DIVISION.
    SET VAL1 OF NUM1 TO TRUE

```

TYPDEF をサポートする機能

ハイロー資格オペレーター

プログラム内にグループタイプのインスタンスが複数存在する場合、そのようなアイテムを一意に識別するために、グループタイプのインスタンス内の下位アイテムを修飾することが頻繁に必要なになる。

ハイ・ロー修飾演算子「::」（「OF」の代替）は、他の言語では一般的な、最上位の名前部から最下位の名前部までの下位項目を修飾する合理的な方法を可能にする。

例新しい high-low 演算子を使用して、下位グループタイプのインスタンス項目を一意に識別する

この例では、ハイ・ロー修飾演算子の使い方を説明する。

```

* type definition
01 MYTYPE-T TYPEDEF.
    03 G1.
        05 G2.
            07 DATA1 PIC X(20).
            07 DATA2 PIC S9(9) COMP-3.

* type instance definitions
01 GRP1 TYPE MYTYPE-T.
01 GRP2 TYPE MYTYPE-T.
:
PROCEDURE DIVISION.
    MOVE 23 TO GRP1::G1::G2::DATA2      *> using ::
    MOVE 48 TO DATA2 OF G2 OF G1 OF GRP2 *> using OF

```

インデックス名資格

インデックス名は、関連するテーブル名を用いて修飾することができる。特に、1つ以上のインデックス名を含むテーブルを定義するグループ・タイプのインスタンスを扱う場合、このアプローチによってインデックス名の一意な識別が保証されます

例：グループ型インスタンスでインデックス名を修飾する

この例では、インデックス名を持つテーブルを含むグループ・タイプの複数のインスタンスを持つプログラムにおいて、インデックス名を一意に修飾する方法を示します。

```

* type definition
01 MYTYPE-T TYPEDEF.
    03 MYTAB OCCURS 10 TIMES INDEXED BY INX.
        05 DATA1 PIC X(20).
        05 DATA2 PIC S9(9) COMP-3.

* type instance definitions
01 GRP1 TYPE MYTYPE-T.
01 GRP2 TYPE MYTYPE-T.
:
PROCEDURE DIVISION.
    SET GRP1::INX TO 8  *> INX in GRP1

```

```
MOVE 'Hello, world 1!' TO  
  GRP1::DATA1(GRP1::INX)  
SET GRP2::INX TO 4 *> INX in GRP2  
MOVE 'Hello, world 2!' TO  
  GRP2::DATA1(GRP2::INX)
```

制約事項

型の使用には以下の制約がある：

- データ記述項目が TYPEDEF 句を含むデータ項目も、そのような項目の下位項目も、RENAMES 句の対象にしてはならない。
- TYPEDEF 句で定義されたデータ項目は、REDEFINES 句の目的語としては使用できない。ただし、グループタイプ定義内の下位項目は再定義できる。このような場合、再定義項目は型定義の一部とみなされる。
- OCCURS DEPENDING ON 句で定義されたテーブルを含むグループ・タイプは、依存オブジェクトと同じグループ・タイプ内のデータ項目を参照できない。
- EXTERNAL 句、VOLATILE 句、RENAMES 句は、TYPEDEF 句を含むデータ記述項目では使用できない。
- 以下の句は、TYPE 句を含むデータ記述項目では使用できない：BLANK WHEN ZERO、DYNAMIC LENGTH、JUSTIFIED、PICTURE、SIGN、SYNCHRONIZED、USAGE。

第 3 章 数値および算術演算

このタスクについて

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

[49 ページの『数値データの定義』](#)

[51 ページの『数値データの表示』](#)

[52 ページの『数値データの保管方法の制御』](#)

[60 ページの『非互換データの検査 \(数値のクラス・テスト\)』](#)

[61 ページの『算術の実行』](#)

[70 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。

このタスクについて

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

```
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で、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約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。

[56 ページの『例: 数値データおよび内部表現』](#)

関連概念

[53 ページの『数値データの形式』](#)

[859 ページの『中間結果および算術精度』](#)

関連タスク

[51 ページの『数値データの表示』](#)

[52 ページの『数値データの保管方法の制御』](#)

[61 ページの『算術の実行』](#)

[141 ページの『国別数値データ項目の定義』](#)

関連参照

[59 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』](#)

[148 ページの『文字データの保管』](#)

[381 ページの『ARITH』](#)

[426 ページの『NUMPROC』](#)

SIGN 節 (*Enterprise COBOL for z/OS 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。

このタスクについて

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
```

の内容が Price0150099 (値 1 を表す) 500.99)、\$ 1,500.99 コードを実行すると表示されます。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。そうするために、定義する数値編集項目は USAGE NATIONAL。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、Edited-price と宣言される USAGE NATIONAL 上記のコードでは、項目は国別文字を使用して編集および表示されます。

USAGE NATIONAL を持つ数値データ項目または数字編集データ項目を EBCDIC で表示するには、それらの項目を CONSOLE に向けてください。例えば、Edited-price 上記のコードには USAGE NATIONAL、\$ 1,500.99 上記の最後のステートメントが次の場合、プログラムを実行すると、次のメッセージが表示されます。

```
Display Edited-price Upon Console
```

基本数値項目または数値編集項目にゼロの値が格納されるときに、その項目をスペースで埋めるには、次のようにコーディングします。BLANK WHEN ZERO アイテムの条項。例えば、DISPLAY 以下のステートメントにより、ゼロの代わりに空白が表示されます。

```
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 upon console
```

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。次の例では、数値編集項目の値は（それが USAGE DISPLAY または USAGE NATIONAL）数値項目に移動されます:

```
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。Price0150099 と表示される場合は、Edited-price 持っていた USAGE NATIONAL。

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。

56 ページの『例: 数値データおよび内部表現』

関連タスク

36 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

52 ページの『数値データの保管方法の制御』

49 ページの『数値データの定義』

61 ページの『算術の実行』

141 ページの『国別数値データ項目の定義』

142 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 (*Enterprise COBOL for z/OS 言語解説書*)

BLANK WHEN ZERO 節 (*Enterprise COBOL for z/OS 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。

このタスクについて

次のような理由から、形式を制御場合があります。

- 計算データ型で実行される演算は、USAGE DISPLAY または USAGE NATIONAL データ型。
- 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 進数形式は、DISPLAY または NATIONAL バイナリ形式よりも効率的にフォーマットします。
- 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。

- 外部小数点 (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。例:

```
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` 節の規約および `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)。

[56 ページの『例: 数値データおよび内部表現』](#)

関連概念

[53 ページの『数値データの形式』](#)

[58 ページの『データ形式の変換』](#)

[859 ページの『中間結果および算術精度』](#)

関連タスク

[49 ページの『数値データの定義』](#)

[51 ページの『数値データの表示』](#)

[61 ページの『算術の実行』](#)

関連参照

[58 ページの『変換および精度』](#)

[59 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幾つかあります。

外部 10 進数 (`DISPLAY` および `NATIONAL`) 項目

`USAGE DISPLAY` がカテゴリー数値データ項目に対して有効な場合 (ユーザーがコーディングしたか、デフォルトにより)、ストレージの各位置 (バイト) には 1 桁の 10 進数が入ります。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。

`USAGE NATIONAL` がカテゴリー数値データ項目に対して有効な場合、各 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` のある外部 10 進数項目には、有効な UTF-16 数字のみが含まれていなければなりません。 そうでない場合、データは正しくないものになり、生成されるコードの動作は不確定になります。 `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、他のカテゴリー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約的に実行し、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

`USAGE DISPLAY` が浮動小数点データ項目に対して有効な場合 (ユーザーがコーディングしたか、またはデフォルトにより)、各 `PICTURE` 文字位置 (使用されている場合は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使用します。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ために必要に応じて、本書では表示浮動小数点データ項目と呼びます。

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。

USAGE NATIONAL が浮動小数点データ項目に対して有効な場合、各 PICTURE 文字位置 (使用されている場合は v を除く) は 2 バイトのストレージを使用します。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。代わりに Compute-Result-N を EBCDIC 文字で表示するには、それをコンソールに送ってください。

```
Display Compute-Result-N Upon Console
```

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、左端のビットが演算記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、コンパイラによる余分な処理が必要となります。SIZE ERROR 条件および丸めのテストは、他のタイプの場合よりも煩雑です。

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・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 として定義したデータ項目は、ストレージ内では 2 進データとして表されます。ただし、USAGE COMP 項目とは異なり、これらの項目には、PICTURE 節内の 9 の数によって暗黙に指定される値に制限されるのではなく、ネイティブ・バイナリー表現の容量 (2、4、または 8 バイト) までの大きさの値を含めることができます。

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 5. COMP-5 データ項目の値の範囲		
PICTURE	ストレージ表記	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 として記述するデータ項目は、ストレージ内では 2 進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、*Enterprise COBOL for z/OS* 言語解説書を参照してください。

TRUNC コンパイラ・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 は単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。左端のビットには符号が入り、次の 7 つのビットには指数が入り、残りの 3 または 7 バイトには仮数が入ります。

COMP-1 および COMP-2 データ項目は、System z® の 16 進形式で保管されます。

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

[859 ページの『中間結果および算術精度』](#)

関連タスク

49 ページの『数値データの定義』

141 ページの『国別数値データ項目の定義』

関連参照

148 ページの『文字データの保管』

454 ページの『TRUNC』

データのクラスおよびカテゴリー (Enterprise COBOL for z/OS 言語解説書)

SIGN 節 (Enterprise COBOL for z/OS 言語解説書)

VALUE 節 (Enterprise COBOL for z/OS 言語解説書)

例: 数値データおよび内部表現

次の表が示すのは、数値項目の内部表現です。

表 6. 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

表 6. 数値項目の内部表現 (続き)			
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 PIC S9999 COMP PIC S9999 COMP-4	+ 1234	04 D2
		- 1234	FB 2E
	PIC S9999 COMP-5	+ 12345 ¹	30 39
		- 12345 ¹	CF C7
	PIC 9999 BINARY PIC 9999 COMP PIC 9999 COMP-4	1234	04 D2
		60000 ¹	EA 60
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 4F
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

表 6. 数値項目の内部表現 (続き)			
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
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。			

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 または COMPUTE ステートメントの受信側への割り当ての場合) 変換します。

変換とは、実際には、値のあるデータ項目から別のデータ項目に移動することです。コンパイラーは、MOVE および COMPUTE ステートメントについて使用されるのと同じ規則を使用して、比較および算術の実行時に必要とされる変換を実行します。

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

58 ページの『変換および精度』

59 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 사용되는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 9 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 9 個だけ受け取り、残りの桁は 0 になります。

USAGE COMP-2 データ項目が 18 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 18 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 9 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。

もし USAGE COMP-2 データ項目は USAGE COMP-1 データ項目の場合、対象データ項目の下位位置で丸めが行われます。

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

[859 ページの『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' が指定された場合、s は符号表記、d は数字を表します。SIGN IS SEPARATE 節のないゾーン 10 進数 (USAGE DISPLAY) データの有効な符号表記は、次のとおりです。

正:

C、A、E、および F

負:

D と B

COBOL NUMPROC コンパイラー・オプションは、ゾーン 10 進数データおよび内部 10 進数データの符号処理に影響します。NUMPROC は、2 進数データ、国別 10 進数データ、および浮動小数点データには影響しません。

NUMPROC(PFD)

X'sd' (s はサイン表記であり、d は数字を表す) が与えられた場合に、NUMPROC (PFD) を使用すると、コンパイラーはデータ内の符号が 3 つの優先符号の 1 つであると想定します。

符号付き正数または 0:

X'C'

符号付き負数:

X'D'

符号なしまたは英数字:

X'F'

この想定に基づいて、コンパイラーは与えられるすべての符号を使用してデータを処理します。優先符号は、必要な場合にのみ生成されます (例えば、符号なしデータが符号付きデータに移動される場合)。NUMPROC (PFD) オプションを使用すると、処理時間を節約することができますが、正しい処理を行うためには、データに優先符号を使用しなければなりません。

NUMPROC(NOPFD)

NUMPROC(NOPFD) コンパイラー・オプションが有効であると、コンパイラーはすべての有効な符号構成を受け入れます。受け取り側では、常に、優先符号が生成されます。NUMPROC(NOPFD) を使用すると、NUMPROC(PFD) より効率が低下しますが、優先符号を使用しないデータが存在する可能性があるときには NUMPROC(NOPFD) を使用しなければなりません。

符号なしのゾーン 10 進数送出側が英数字受信に移動されても、符号は未変更のままです (NUMPROC(NOPFD) が有効な場合でも)。

関連参照

426 ページの『NUMPROC』

468 ページの『ZWB』

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が PICTURE および USAGE 節に対して有効であると想定し、その妥当性を検査しません。追加処理で項目を使用する前に、データ項目の内容が PICTURE 節および USAGE 節に準拠していることを確認してください。

このタスクについて

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。データ記述と互換性のない値を項目に指定すると、PROCEDURE DIVISION 内のその項目への参照は未定義になり、結果は予測不能となります。

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。例:

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

数値クラス・テストは、データ項目の PICTURE および USAGE に有効な一連の値に対して、データ項目の内容を検査します。例えば、パック 10 進数項目は、数字位置に 16 進値 X'0' から X'9' があり、符号位置 (分離または非分離) に有効な符号値があるかどうか検査されます。USAGE DISPLAY を持つ外部 10 進データ項目は、桁位置 (各バイトの下位 4 ビット) に 16 進値 X'0' から X'9' があるかどうか、各バイトの上位 4 ビットに有効なゾーン・コードがあるかどうか、および符号位置 (分離または非分離) に有効な符号値があるかどうか検査されます。符号コードは、符号バイトの上位 4 ビットにあるか、SIGN IS SEPARATE が指定されている場合は別のバイトにあります。SIGN IS SEPARATE 節を使用する場合、すべてのバイトの上位 4 ビットは x'F' でなければなりません。

注: INVDATA コンパイラー・オプションを使用すると、数値比較で USAGE DISPLAY 数値 (ゾーン 10 進数) データ項目内の無効なゾーン・コードを許容できますが、ゾーン 10 進数データ項目内の無効なゾーン・コードは、数値クラス・テストでは非数値として扱われます。

ゾーン 10 進数 および パック 10 進数項目の場合、数値のクラス・テストは、NUMPROC コンパイラー・オプションおよび NUMCLS オプション (インストール時に設定される) の影響を受けます。インストールの際に使用された NUMCLS 設定を確認するには、システム・プログラマーに問い合わせてください。

NUMCLS (PRIM) がインストール先で有効な場合は、次の表を使用して、コンパイラーが符号に有効と見なす値を見つけてください。

表 7. <i>NUMCLS(PRIM)</i> および有効な符号		
	NUMPROC (NOPFD)	NUMPROC (PFD)
signed	C、D、F	C、D、+0 (正のゼロ)
符号なし	F	F
分離符号	+, -	+, -, +0 (正のゼロ)

NUMCLS (ALT) がインストール先で有効な場合は、次の表を使用して、コンパイラーが符号に有効と見なす値を見つけてください。

表 8. <i>NUMCLS(ALT)</i> および有効な符号		
	NUMPROC (NOPFD)	NUMPROC (PFD)
signed	A から F	C、D、+0 (正のゼロ)
符号なし	F	F
分離符号	+, -	+, -, +0 (正のゼロ)

数値クラス条件の詳細については、*Enterprise COBOL for z/OS* 言語解説書の *Class condition* を参照してください。

NUMCHECK (ZON, PAC) オプションを使用して、送信データ項目として使用されるゾーン 10 進数 (数値 USAGE DISPLAY) およびパック 10 進数 (COMP-3) データ項目の暗黙的な数値クラス・テストをコンパイラーに生成させることもできます。この数値クラス・テストは、データの妥当性検査を行い、NUMPROC コンパイラー・オプションに対して符号フィールドの妥当性検査も行います。これは、NUMPROC (PFD) を使用できるかどうかを判断するのに役立ちます。詳しくは、[422 ページの『NUMCHECK』](#)を参照してください。

その他の無効データの問題 (特に COBOL のマイグレーション目的) の詳細については、「*Enterprise COBOL* 移行ガイド」の「無効データに関する FAQ」を参照してください。

関連参照

[422 ページの『NUMCHECK』](#)

[426 ページの『NUMPROC』](#)

[408 ページの『INVDATA』](#)

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。

このタスクについて

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。数学関数および日時操作の言語環境呼び出し可能サービスは、算術結果をデータ項目に割り当てる手段も提供します。

関連タスク

[62 ページの『COMPUTE およびその他の算術ステートメントの使用』](#)

[62 ページの『算術式の使用』](#)

[63 ページの『数値組み込み関数の使用』](#)

[64 ページの『数学用の呼び出し可能サービスの使用』](#)

[66 ページの『データ呼び出し可能サービスの使用』](#)

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。

このタスクについて

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。
例:

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場合、ゾーン 10 進数データ項目を使用する場合と同様に、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

[68 ページの『固定小数点演算と浮動小数点演算の対比』](#)
[859 ページの『中間結果および算術精度』](#)

関連タスク

[49 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)。

このタスクについて

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 9. 算術演算子の評価の順序		
オペレーター	意味	評価の順序
単項 + または -	代数符号	最初
**	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

優先順位が同レベルの演算子は、左から右へ評価されます。括弧を使用すると、評価の順序を変更できます。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

68 ページの『固定小数点演算と浮動小数点演算の対比』

859 ページの『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。

このタスクについて

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。

FLOATING POINT

長精度 (64 ビット) または拡張精度を返すもの(128 ビット)浮動小数点値 (デフォルトオプションを使用してコンパイルするかどうかによって異なります) ARITH (COMPAT) または使用 ARITH (EXTEND))

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 10. 数値組み込み関数				
数値処理	日時	金融	数学	統計
LENGTH MAX MIN NUMVAL NUMVAL-C NUMVAL-F SIGN TEST-NUMVAL TEST-NUMVAL-C TEST-NUMVAL-F ORD-MAX ORD-MIN	CURRENT-DATE DATE-OF-INTEGER DATE-TO-YYYYMMDD DAY-OF-INTEGER DAY-TO-YYYYDDD INTEGER-OF-DATE INTEGER-OF-DAY WHEN-COMPILED YEAR-TO-YYYY	ANNUITY PRESENT-VALUE	ABS ACOS ASIN ATAN COS E EXP EXP10 FACTORIAL INTEGER INTEGER-PART LOG LOG10 MOD PI REM SIN SQRT SUM TAN	MEAN MEDIAN MIDRANGE RANDOM RANGE STANDARD-DEVIATION VARIANCE

67 ページの『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、コンパイラー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、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、以下のステートメントには、3 つの関数引数 (a、b、および演算式 (c / d)) があります。

```
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ーは、整数の引数で使用できるところであればどこでも引数として使用することができます。

数字組み込み関数の機能の多くは、言語環境呼び出し可能サービスによっても提供されます。

関連概念

68 ページの『固定小数点演算と浮動小数点演算の対比』

859 ページの『中間結果および算術精度』

関連参照

381 ページの『ARITH』

数学用の呼び出し可能サービスの使用

ほとんどの COBOL 組み込み関数には、同じ結果を得るのに使用できる、対応する数学用の呼び出し可能サービスがあります。

このタスクについて

デフォルト・オプション ARITH (COMPAT) を使用してコンパイルすると、COBOL 浮動小数点組み込み関数は長精度 (64 ビット) の結果を戻します。オプション ARITH (EXTEND) を使用してコンパイルすると、COBOL 浮動小数点組み込み関数 (RANDOM は例外) は拡張精度 (128 ビット) の結果を戻します。

例えば (以下の表の最初の行を考慮します)、ARITH (COMPAT) を使用してコンパイルした場合、CEESDACS は ACOS と同じ結果を戻します。ARITH (EXTEND) を使用してコンパイルした場合は、CEESQACS が ACOS と同じ結果を戻します。

表 11. 演算組み込み関数と呼び出し可能サービスの互換性			
COBOL 組み込み関数	対応する長精度言語環境呼び出し可能サービス	対応する拡張精度言語環境呼び出し可能サービス	組み込み関数と呼び出し可能サービスで結果が同じか
ACOS	CEESDACS	CEESQACS	はい
ASIN	CEESDASN	CEESQASN	はい
ATAN	CEESDATN	CEESQATN	はい
COS	CEESDCOS	CEESQCOS	はい
E	CEESDEXP (<i>parm1</i> が 1.0 に設定)	CEESQEXP (<i>parm1</i> が 1.0 に設定)	はい
EXP	CEESDEXP	CEESQEXP	はい
EXP10	CEESDXPD (<i>parm1</i> が 10.0 に設定)	CEESQXPQ (<i>parm1</i> が 10.0 に設定)	はい
LOG	CEESDLOG	CEESQLOG	はい
LOG10	CEESDLG1	CEESQLG1	はい
RANDOM ¹	CEERANO	なし	いいえ
REM	CEESDMOD	CEESQMOD	はい
SIN	CEESDSIN	CEESQSIN	はい
SQRT	CEESDSQT	CEESQSQT	はい
TAN	CEESDTAN	CEESQTAN	はい
1. RANDOM は、31 桁の引数を渡し、ARITH (EXTEND) を指定してコンパイルした場合でも、長精度 (64 ビット) 浮動小数点の結果を戻します。			

RANDOM 組み込み関数と CEERANO サービスはともに、0 から 1 の範囲の乱数を生成します。ただし、それぞれが独自のアルゴリズムを使用するため、RANDOM と CEERANO は同じシードから異なる乱数を生成します。

同じ結果をもたらす関数であっても、組み込み関数と言語環境呼び出し可能サービスの使用法は異なります。組み込み関数の引数に要求されるデータ型に関する規則の方が、制限が緩くなっています。数字組み込み関数の場合には、任意の数値データ型の引数を使用することができます。しかし、CALL ステートメントを使用して言語環境呼び出し可能サービスを呼び出す場合は、パラメーターをそのサービスに必要な数値データ型 (通常は COMP-1 または COMP-2) と一致させなければなりません。

組み込み関数と言語環境呼び出し可能サービスのエラー処理は異なる場合があります。言語環境の数学サービスを呼び出すときに明示的なフィードバック・トークンを渡す場合は、それぞれの呼び出しの後でそのフィードバック・コードを検査し、エラーを処理するための明示的なアクションを取らなければなりません。しかし、フィードバック・トークンを使用して明示的に OMITTED を呼び出した場合は、トークンを検査する必要はありません。エラーがあれば、言語環境が自動的にエラーを通知します。

関連概念

68 ページの『固定小数点演算と浮動小数点演算の対比』

859 ページの『中間結果および算術精度』

関連タスク

853 ページの『言語環境呼び出し可能サービスの使用』

関連参照

381 ページの『ARITH』

データ呼び出し可能サービスの使用

COBOL の日付組み込み関数と言語環境の日付呼び出し可能サービスは両方とも、グレゴリオ暦を基本とします。ただし、開始日付は、INTDATE コンパイラ・オプションの設定値によって異なることがあります。

このタスクについて

いつ INTDATE(LILIAN) が有効な場合、COBOL は 1582 年 10 月 15 日を 1 日目として使用します。言語環境常に 1582 年 10 月 15 日を 1 日目として使用します。INTDATE(LILIAN) を使用すると、COBOL 組み込み関数と言語環境の日付呼び出し可能サービスから同じ結果が得られます。次の表は、INTDATE(LILIAN) が有効な場合の結果を比較したものです。

表 12. 日付組み込み関数と呼び出し可能サービスの INTDATE(LILIAN) と互換性		
COBOL 組み込み関数	言語環境の呼び出し可能サービス	結果
DATE-OF-INTEG	CEEDATE (ピクチャー・ストリング YYYYYMDD 付き)	互換性あり
DAY-OF-INTEG	CEEDATE (ピクチャー・ストリング YYYYYDDD 付き)	互換性あり
INTEGER-OF-DATE	CEEDAYS	互換性あり
INTEGER-OF-DATE	CEECBLDY	非互換

デフォルト設定である INTDATE(ANSI) が有効な場合、COBOL は第 1 日として 1601 年 1 月 1 日を使用します。次の表は、INTDATE(ANSI) が有効な場合の結果を比較したものです。

表 13. 日付組み込み関数と呼び出し可能サービスの INTDATE(ANSI) と互換性		
COBOL 組み込み関数	言語環境の呼び出し可能サービス	結果
INTEGER-OF-DATE	CEECBLDY	互換性あり
DATE-OF-INTEG	CEEDATE (ピクチャー・ストリング YYYYYMDD 付き)	非互換
DAY-OF-INTEG	CEEDATE (ピクチャー・ストリング YYYYYDDD 付き)	非互換
INTEGER-OF-DATE	CEEDAYS	非互換

関連タスク

853 ページの『言語環境呼び出し可能サービスの使用』

関連参照

408 ページの『INTDATE』

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
```

日時

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。返される最初の 8 文字は CURRENT-DATE 関数は、日付を 4 桁の年、2 桁の月、2 桁の日の形式で表します (YYYYMMDD 日付は整数値に変換され、この値に 90 が加算され、整数が元の日付に戻されます。YYYYMMDD フォーマット。

```
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

企業投資判断では、計画された投資の採算性を評価するために、期待される将来の現金流入の現在価値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 =
```

```
Function
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

[123 ページの『数値への変換 \(NUMVAL、NUMVAL-C、NUMVAL-F\)』](#)

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。

注: 固定小数点評価は、16 進浮動小数点命令とはまったく異なる、10 進浮動小数点命令で行われる場合があります。

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、次のように定義されるデータ項目としてコーディングすると、浮動小数点になります。USAGE COMP-1、USAGE COMP-2、または外部浮動小数点 (USAGE DISPLAY または USAGE NATIONAL 浮動小数点付き PICTURE)。

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。

- 指数に小数位が含まれている。

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

一般に、算術演算に上記の浮動小数点の特性が含まれていなければ、コンパイラーはその算術演算を固定小数点演算として評価します。つまり、算術計算は、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して定義され、かつどの指数も小数位を持つ値を表さない場合にのみ、固定小数点として処理されます。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。つまり、それぞれの属性が他の計算に影響を与える可能性があります。両方の式が固定小数点で評価されるか、両方の式が浮動小数点で評価されることになります。これは、(比較で一方の被比較数が明示的に示されない場合でも) 簡略比較にも当てはまります。例:

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

このステートメントには 2 つの比較があります。(a + d) = (b + e)、そして (a + d) = c。それでも (a + d) 2 番目の比較では明示的に表示されませんが、その比較では比較対象になります。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、コンパイラによって固定小数点演算として処理されます。

暗黙的な比較 (関係演算子を使用されない) は、単位として処理されません。ただし、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での評価に関しては、個別に扱われます。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
```

[70 ページの『例: 固定小数点計算および浮動小数点計算』](#)

関連参照

[867 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)ve+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、プログラムで幾つかの通貨記号を使用できます。

このタスクについて

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

財務情報を表示するためのシンボルを指定するには、CURRENCY SIGN 条項 (SPECIAL-NAMES 段落の CONFIGURATION SECTION) とともに PICTURE それらのシンボルに関連する文字。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 進数値 X'9F' を示しています。

```
Currency Sign X'9F' with Picture Symbol 'U'.
01 Deposit-Amount      Pic UUUUU9.99.
```

キーボードにユーロ記号に対応する文字がない場合は、16 進数値として指定する必要があります。CURRENCY SIGN 句。ユーロ記号の 16 進値は、次の表に示すように、使用されているコード・ページに応じて、X'9F' または X'5A' のいずれかです。

表 14. ユーロ記号の 16 進値			
コード・ページ CCSID	適用される国	変更元	ユーロ記号
1140	米国、カナダ、オランダ、ポルトガル、オーストラリア、ニュージーランド	037	X'9F'
1141	オーストリア、ドイツ	273	X'9F'
1142	デンマーク、ノルウェー	277	X'5A'
1143	フィンランド、スウェーデン	278	X'5A'
1144	イタリア	280	X'9F'
1145	スペイン、ラテンアメリカ - スペイン語圏	284	X'9F'
1146	英国	285	X'9F'
1147	フランス	297	X'9F'
1148	ベルギー、カナダ、スイス	500	X'9F'
1149	アイスランド	871	X'9F'

関連参照

390 ページの『CURRENCY』

CURRENCY SIGN 節 (*Enterprise COBOL for z/OS* 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 "Y".
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 -FFFFF9.99.
    02 Report-in-Euro   Pic -YYYYY9.99.
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。

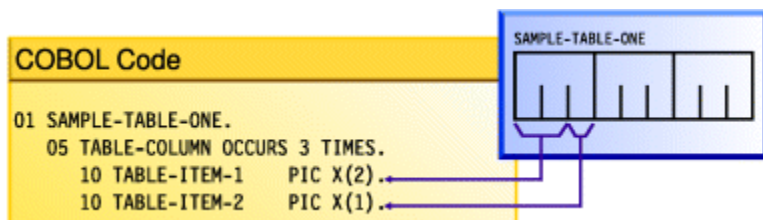
```
Deposit in euro = EUR 8000.00
Deposit in franc = CHF 12311.44
```

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

このタスクについて

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。

繰り返し項目を別個の連続したエントリとして定義するのではなく、DATA DIVISION、あなたは OCCURS 条項の DATA DIVISION テーブルを定義するエントリ。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プログラムの速度を高めるうえで大切なものです。

関連概念

[87 ページの『複合 OCCURS
DEPENDENT ON』](#)

関連タスク

- [73 ページの『テーブルの定義 \(OCCURS\)』](#)
- [74 ページの『テーブルのネスト』](#)
- [76 ページの『テーブル内の項目の参照』](#)
- [79 ページの『テーブルに値を入れる方法』](#)
- [84 ページの『可変長テーブルの作成 \(DEPENDENT ON\)』](#)
- [91 ページの『テーブルの探索』](#)
- [94 ページの『テーブルのソート』](#)
- [94 ページの『組み込み関数を使用したテーブル項目の処理』](#)
- [95 ページの『無制限テーブルおよび無制限グループの処理』](#)

[841 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。

このタスクについて

```
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 記述で使用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次のように記述されなければならない。USAGE NATIONAL、そして符号付きの従属数値データ項目は、暗黙的または明示的に記述されなければならない。SIGN IS SEPARATE 句。

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、DEPENDING ON のフレーズ OCCURS 句。

テーブル要素がテーブルの1つ以上のキーフィールドの値に基づいて昇順または降順で並べられることを指定するには、ASCENDING または DESCENDING KEY のフレーズ OCCURS 条項、またはその両方。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。(もし USAGE NATIONAL キーは、国別カテゴリーにすることも、国別編集項目、数値編集項目、国別10進項目、または国別浮動小数点項目にすることもできます。

コーディングする必要があります ASCENDING または DESCENDING KEY のフレーズ OCCURS 二分探索を行う句 (SEARCH ALL) です。フォーマット 2 を使用できます SORT 定義されたキーに従ってテーブルを順序付けするステートメントにより、テーブルは SEARCH ALL 声明。ご了承ください SEARCH ALL テーブルがキーに従って順序付けられていない場合、予期しない結果が返されます。

93 ページの『例: 二分探索』

関連概念

141 ページの『国別グループ』

関連タスク

74 ページの『テーブルのネスト』

76 ページの『テーブル内の項目の参照』

79 ページの『テーブルに値を入れる方法』

84 ページの『可変長テーブルの作成 (DEPENDING ON)』

145 ページの『国別グループの使用』

92 ページの『二分探索 (SEARCH ALL)』

49 ページの『数値データの定義』

関連参照

OCCURS 節 (*Enterprise COBOL for z/OS 言語解説書*)

SIGN 節 (*Enterprise COBOL for z/OS 言語解説書*)

ASCENDING KEY および DESCENDING KEY 句

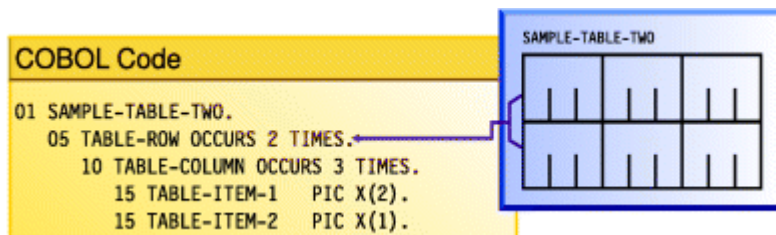
(*Enterprise COBOL for z/OS 言語解説書*)

SORT ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

テーブルのネスト

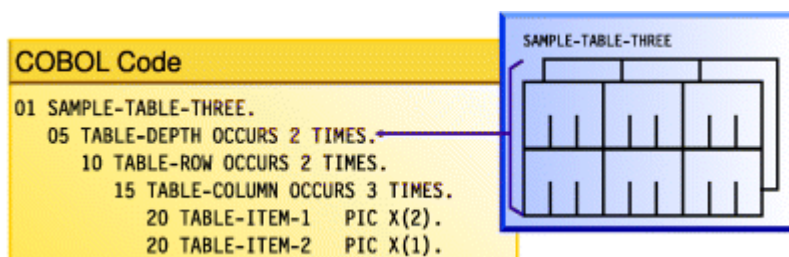
2次元テーブルを作成するには、ある1次元テーブルのそれぞれのオカレンス (出現) の中で別の1次元テーブルを定義します。

このタスクについて



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 (それ自身が別の 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。例:



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。

[75 ページの『例: 添え字付け』](#)

[76 ページの『例: 指標付け』](#)

関連タスク

[73 ページの『テーブルの定義 \(OCCURS\)』](#)

[76 ページの『テーブル内の項目の参照』](#)

[79 ページの『テーブルに値を入れる方法』](#)

[84 ページの『可変長テーブルの作成 \(DEPENDENT ON\)』](#)

[91 ページの『テーブルの探索』](#)

[94 ページの『組み込み関数を使用したテーブル項目の処理』](#)

[841 ページの『テーブルの効率的処理』](#)

関連参照

OCCURS 節 (*Enterprise COBOL for z/OS 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

[77 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

[77 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。

このタスクについて

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

[77 ページの『添え字付け』](#)

[77 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。

このタスクについて

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。例:

```
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。例:

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

上記の例の 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 文字の長さだけ、テーブル・エレメント 1 に移動します。

ANY-TABLE before the change: ABCDEFGHIJ ABCDEFGHIJ ABCDEFGHIJ	ANY-TABLE after the change: AB??EFGHIJ ABCDEFGHIJ ABCDEFGHIJ

[75 ページの『例: 添え字付け』](#)

関連タスク

[77 ページの『索引付け』](#)

[79 ページの『テーブルに値を入れる方法』](#)

[91 ページの『テーブルの探索』](#)

[841 ページの『テーブルの効率的処理』](#)

索引付け

インデックスを作成するには、INDEXED BY のフレーズ OCCURS インデックス名を識別する句。

このタスクについて

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。

あなたは USAGE IS INDEX インデックスを作成する句データ項目、任意のテーブルでインデックス 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.
```

インデックス名 INX-A テーブルをトラバースするために使用される TABLE-ITEM その上。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。

あなたは SET 割り当てるステートメントインデックスデータ項目保存した値はインデックス名、ステートメント内の SET INX-B TO INX-A その上。例えば、レコードを可変長テーブルにロードする際に、最終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。その後で、現行指標値を最終レコードの指標値と比較することにより、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す。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。

[76 ページの『例: 指標付け』](#)

関連タスク

[77 ページの『添え字付け』](#)

[79 ページの『テーブルに値を入れる方法』](#)

[91 ページの『テーブルの探索』](#)

[94 ページの『組み込み関数を使用したテーブル項目の処理』](#)

[841 ページの『テーブルの効率的処理』](#)

関連参照

INDEXED BY 句 (*Enterprise COBOL for z/OS* 言語解説書)

INDEX 句 (*Enterprise COBOL for z/OS* 言語解説書)

SET ステートメント (*Enterprise COBOL for z/OS* 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。

このタスクについて

関連タスク

[79 ページの『テーブルの動的なロード』](#)

[86 ページの『可変長テーブルのロード』](#)

[79 ページの『テーブルの初期化 \(INITIALIZE\)』](#)

[80 ページの『テーブルの定義時の値の割り当て \(VALUE\)』](#)

[87 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、変更済みの値を動的にテーブルに読み込むことができます。

このタスクについて

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。

[82 ページの『例: PERFORM と添え字付け』](#)

[83 ページの『例: PERFORM および索引付け』](#)

関連参照

PERFORM ステートメント (*Enterprise COBOL for z/OS* 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。

このタスクについて

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".
```

使用すると INITIALIZE テーブルを初期化するステートメントでは、テーブルはグループ項目として処理されます (つまり、グループセマンティクスを持つ)。グループ内の基本データ項目が認識され、処理されます。例えば、TABLE-ONE 次のように定義される英数字のグループです。

```
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 国グループとして定義されたテーブルをロードするステートメント。例えば、TABLE-ONE 上記に示した GROUP-USAGE NATIONAL 条項、および Trans-code そして Part-number 持っていた N の代わりに X 彼らの PICTURE 句の場合、次の文は INITIALIZE 上記の記述とは異なり、TABLE-ONE 代わりに UTF-16:

```
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。

あなたは REPLACING のフレーズ INITIALIZE ステートメントは同様にすべての基本要素を初期化します ALPHABETIC、DBCS、ALPHANUMERIC-EDITED、NATIONAL-EDITED、そして NUMERIC-EDITED テーブル内のデータ項目。

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 \(INITIALIZE\)』](#)

[80 ページの『テーブルの定義時の値の割り当て \(VALUE\)』](#)

[87 ページの『可変長テーブルへの値の割り当て』](#)

[109 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』](#)

[145 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 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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

81 ページの『それぞれのテーブル項目の個別の初期化』

82 ページの『グループ・レベルでのテーブルの初期化』

82 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。

このタスクについて

以下のコード例に示されている手法を使用します。

手順

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) を定義します。
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。

タスクの結果

```
*****
***      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)』

87 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 (*Enterprise COBOL for z/OS 言語解説書*)

OCCURS 節 (*Enterprise COBOL for z/OS 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。

このタスクについて

以下の例の英数字グループ・データ項目 TABLE-ONE は、TABLE-TWO の 4 個の要素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.
```

次の例では、国別グループデータ項目 Table-OneN 使用 VALUE 従属データ項目の 3 つの要素をそれぞれ初期化する句 Table-TwoN（それぞれは暗黙的に USAGE NATIONAL）。国別グループデータ項目は、VALUE 以下に示すように、英数字リテラル、または国別リテラルを使用する句。

```
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 (*Enterprise COBOL for z/OS 言語解説書*)

GROUP-USAGE 節 (*Enterprise COBOL for z/OS 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。

このタスクについて

```
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

[87 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 (*Enterprise COBOL for z/OS 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。

```

*****
***      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.
  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。

```

*****
***      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 (DEPENDENT 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 DEPENDENT ON (ODO) 条項。

このタスクについて

```
X OCCURS 1 TO 10 TIMES DEPENDENT 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。

無制限テーブルおよび無制限グループを指定することもできます。詳しくは、「Enterprise COBOL for z/OS 言語解説書」の『可変長テーブル』を参照してください。

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、DEPENDENT ON フレーズと主題の長さ OCCURS 句。
- オブジェクト内のデータの適合性 OCCURS DEPENDENT ON 条項に PICTURE 句

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。それ以外の場合は、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、最大のお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 回のおカレンスと 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は、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 (再び受信アイテム) ここで REC-1 可変位置のグループ (複雑な ODO)、実際の長さ REC-1 移動の直前に ODO オブジェクトの現在の値を使用して計算されます (FIELD-1)。次の例では、REC-1 として REC-2 同じ記録にあるが、REC-2 従属的ではない REC-1 したがって、位置はさまざまです。

```
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
   RECORDING MODE F
   BLOCK 0 RECORDS
   RECORD 80 CHARACTERS
   LABEL RECORD STANDARD.
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

[87 ページの『複合 OCCURS
DEPENDING ON』](#)

関連タスク

[87 ページの『可変長テーブルへの値の割り当て』](#)

[86 ページの『可変長テーブルのロード』](#)

[90 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』](#)

[129 ページの『データ項目の長さの検出』](#)

関連参照

OCCURS DEPENDING ON 節

(*Enterprise COBOL for z/OS 言語解説書*)

可変長テーブル (*Enterprise COBOL for z/OS 言語解説書*)

可変長テーブルのロード

あなたは行うまで構造 (TEST AFTER 可変長テーブルの読み込みを制御するためにループを使用します。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。

このタスクについて

```

DATA DIVISION.
FILE SECTION.
FD  LOCATION-FILE
   RECORDING MODE F
   BLOCK 0 RECORDS
   RECORD 80 CHARACTERS
   LABEL RECORD STANDARD.
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

コーディングすることができます VALUE 句従属データ項目を持つ英数字または国別グループ項目の場合含まれている OCCURS 条項と DEPENDING ON フレーズ。各下位構造には、DEPENDING ON フレーズは最大出現回数を使用して初期化されます。

このタスクについて

テーブル全体を DEPENDING ON フレーズでは、すべての要素は ODO の最大値を使用して初期化されます (OCCURS DEPENDING ON) 物体。

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 つの要素 (Y(1) から Y(3)) を参照します。

関連タスク

[80 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

(Enterprise COBOL for z/OS 言語解説書)

複合 OCCURS DEPENDING ON

いくつかの種類の複合体 OCCURS DEPENDING ON (複雑な ODO) が可能です。複合 ODO は、85 COBOL 標準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。
- 可変長要素を持つテーブル: OCCURS 句には、OCCURS 条項と DEPENDING ON フレーズ。
- 可変長要素を持つテーブルの指標名。
- 可変長要素を持つテーブルの要素。

[88 ページの『例: 複合 ODO』](#)

関連タスク

[89 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』](#)

[90 ページの『要素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

[89 ページの『ODO オブジェクト値の変更の影響』](#)

OCCURS DEPENDING ON 節

(Enterprise COBOL for z/OS 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.
```

意味：この例では、COUNTER-1 は ODO オブジェクトつまり、それは DEPENDING ON 条項の RECORD-1。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 オブジェクトは可変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 句が指定された 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) が続いている場合に、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。

例:

- 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

[89 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』](#)

[90 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。

このタスクについて

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・オフセットは有効ではなくなります。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、予期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)
- あ SET フォームの記述 SET インデックス名 UP|DOWN BY 整数(フォーマット 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。

手順

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。)

タスクの結果

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 (88 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 (*Enterprise COBOL for z/OS* 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、エレメントをテーブルに追加する際に、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おそれがあります。

このタスクについて

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。

手順

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。

タスクの結果

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 S99.
   05 CONTROL-2              PIC S99.
   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).

```

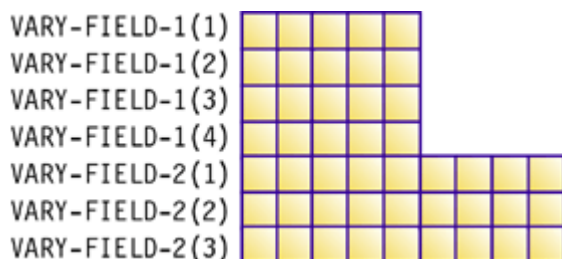
VARY-FIELD-1 の各エレメントは 5 バイトで、VARY-FIELD-2 の各エレメントは 9 バイトです。CONTROL-1 と CONTROL-2 の両方に値 3 が含まれている場合は、VARY-FIELD-1 および VARY-FIELD-2 のストレージは次のように示すことができます。

VARY-FIELD-1(1)										
VARY-FIELD-1(2)										
VARY-FIELD-1(3)										
VARY-FIELD-2(1)										
VARY-FIELD-2(2)										
VARY-FIELD-2(3)										

4 番目の要素を追加するには VARY-FIELD-1、最初の 5 バイトがオーバーレイされないようにするには、次のようにコーディングします。VARY-FIELD-2、(GROUP-ITEM-2 さまざまな場所にある一時的なストレージとして機能します GROUP-ITEM-1。

```
MOVE GROUP-ITEM-1 TO GROUP-ITEM-2.  
ADD 1 TO CONTROL-1.  
MOVE five-byte-field TO  
  VARY-FIELD-1 (CONTROL-1).  
MOVE GROUP-ITEM-2 TO GROUP-ITEM-1.
```

VARY-FIELD-1 と VARY-FIELD-2 の更新後のストレージは次のように示すことができます。



VARY-FIELD-1 の 4 番目のエレメントが VARY-FIELD-2 の最初のエレメントをオーバーレイしていないことに注意してください。

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。

このタスクについて

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) のみの回数になります。ただし、二分探索では、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

[91 ページの『逐次探索 \(SEARCH\)』](#)

[92 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。

このタスクについて

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。

92 ページの『例: 逐次探索』

関連参照

SEARCH ステートメント (*Enterprise COBOL for z/OS 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。

このタスクについて

使用するには SEARCH ALL 声明テーブルを検索するには、テーブルで ASCENDING または DESCENDING KEY のフレーズ OCCURS 条項、またはその両方、およびすでに指定されたキーまたはキーに注文されている必要があります ASCENDING そして DESCENDING KEY フレーズ。フォーマット 2 を使用できます SORT 定義されたキーに従ってテーブルを順序付けするステートメントにより、テーブルは SEARCH ALL 声明。ご了承ください SEARCH ALL テーブルがキーに従って順序付けられていない場合、予期しない結果が返されます。

の中に WHEN のフレーズ SEARCH ALL ステートメントでは、ASCENDING または DESCENDING KEY テーブル用のフレーズ、ただし、先行するキーがある場合は、すべてテストする必要があります。テストは等しい条件でなければならず、WHEN 句は、キー (テーブルに関連付けられた最初のインデックス名で添え字付けされる) またはキーに関連付けられた条件名のいずれか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。

93 ページの『例: 二分探索』

関連タスク

73 ページの『テーブルの定義 (OCCURS)』

関連参照

SEARCH ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

一般比較条件 (*Enterprise COBOL for z/OS 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と等しいエントリが見つかった場合 (VALUE-1、VALUE-2、そして VALUE-3、それぞれ)、PART-1 そのエントリは OUTPUT-AREA。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。

テーブルのソート

形式 2 SORT ステートメントを使用してテーブルをソートできます。これは 2002 COBOL 標準の一部です。

このタスクについて

形式 2 SORT ステートメントは、指定されたテーブル・キーに従ってテーブル・エレメントをソートします。これは特に、SEARCH ALL で使用されるテーブルに役立ちます。ソートのキーはテーブル定義の一部として指定でき、これは SEARCH ALL ステートメントでも使用できます。また、ソートのキーは SORT ステートメントの一部として指定することもできます。これは、テーブル定義で指定されたものとは異なるキーを使用してテーブルをソートしたい場合や、テーブルでキーが指定されていない場合でも可能です。

形式 2 SORT ステートメントでは、形式 1 SORT ステートメントのように入出力プロシージャーを使用する必要はありません。

指定されたキーに基づいてテーブルをソートする例を以下に示します。

```
WORKING-STORAGE SECTION.  
01 GROUP-ITEM.  
    05 TABL OCCURS 10 TIMES  
        10 ELEM-ITEM1 PIC X.  
        10 ELEM-ITEM2 PIC X.  
        10 ELEM-ITEM3 PIC X.  
..  
PROCEDURE DIVISION.  
    ..  
    SORT TABL DESCENDING ELEM-ITEM2 ELEM-ITEM3.  
    IF TABL (1)...
```

関連参照

SORT ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

857 ページの『形式 2 SORT ステートメントを使用したテーブルのソート』

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、関数の引数要件と互換性があるようにする必要があります。

このタスクについて

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エレメント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))
```

95 ページの『例: 組み込み関数を使用したテーブルの処理』

関連タスク

38 ページの『組み込み関数の使用』

[122 ページの『データ項目の変換 \(組み込み関数\)』](#)

[126 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 (Enterprise COBOL for z/OS 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))
```

無制限テーブルおよび無制限グループの処理

無制限グループは、呼び出し先プログラムに対する入力パラメーターとして処理することができます。無制限グループ用のメモリーは、呼び出し側プログラムによって提供されます。また、単一プログラム内で無制限グループの定義、初期化、および処理を行うこともできます。

このタスクについて

単一プログラム内で無制限テーブルおよび無制限グループを処理するには、以下の手順に従ってください。

1. LINKAGE SECTION で、無制限グループの一部になる無制限テーブルを定義します (構文は OCCURS *n* TO UNBOUNDED)。
2. の中に WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION、定義する OCCURS DEPENDING ON オブジェクト。
3. PROCEDURE DIVISION で、以下の手順で無制限グループを処理します。
 - a. OCCURS DEPENDING ON オブジェクトを設定します。
 - b. LENGTH 特殊レジスターまたは LENGTH 組み込み関数を使用して、グループの合計サイズを計算します。
 - c. CALL ステートメントを使用して、Language Environment サービス CEEGTST などのストレージ割り振りサービス呼び出しします。グループの合計長に十分なメモリーを割り振ります。このメモリーに対するポインターが必要になります (CEEGTST サービスからポインターが返されます)。
 - d. SET ステートメントを使用して、アドレス可能性を設定します。例えば、SET ADDRESS OF *group* TO *pointer* などです。

4. 以下の規則に従って、無制限テーブルとそれを含む無制限グループを使用します。

- テーブルを参照可能であればどこでも、COBOL 構文内の無制限テーブルを参照できます。
- 英数字グループまたは国別グループを参照可能であればどこでも、COBOL 構文内の無制限グループを参照できますが、以下の例外があります。
 - CALL ステートメント内の BY CONTENT 引数として無制限グループを指定することはできません。
 - PROCEDURE DIVISION RETURNING 句で無制限グループを *data-name-2* として指定することはできません。
 - LENGTH 組み込み関数に対する引数を除き、組み込み関数への引数として無制限グループを指定することはできません。

関連参照

96 ページの『例: XML 文書を構文解析するための無制限テーブルの使用』

例: 無制限テーブルに対する ALLOCATE および FREE ストレージ

(Enterprise COBOL for z/OS 言語解説書)

可変長テーブル (Enterprise COBOL for z/OS 言語解説書)

OCCURS DEPENDING ON 節

(Enterprise COBOL for z/OS 言語解説書)

例: XML 文書を構文解析するための無制限テーブルの使用

反復エレメント数が不明な XML 文書を構文解析する際には、無制限テーブルを使用することを考慮してください。

以下のいずれかの方法を使用できます。

- 想定されるエレメント数を事前に決定します。エレメント数を決定する方法の 1 つに、XML 文書を 2 回構文解析する方法があります。初回の構文解析時に、対応する OCCURS UNBOUNDED DEPENDING ON オブジェクト内の各無制限エレメントのオカレンス数を数えます。次に、これらの計算された値を使用してデータ項目のストレージを割り振り、そのペイロードを処理するために XML 文書の 2 回目の構文解析を行います。
- 初期サイズを選び、テーブルの拡張を許可します。任意の制限を設ける方が効率的かもしれない OCCURS UNBOUNDED DEPENDING ON 過去の経験に基づいてオブジェクトを分類し、ドキュメントを直接解析してそのコンテンツを処理します。無制限エレメントのそれぞれについて、現在の制限を超えようとしているかどうかを確認します。超えそうな場合は、対応する配列にさらにストレージを割り振り、データを旧配列から拡張された配列にコピーし、その後で旧配列のストレージを解放します。

次の例は、最初の方法を示しています。XML スキーマの例を見て、エレメント B と C に maxOccurs 値 unbounded があること、そのためこれらのエレメントがエレメント G 内のシーケンスで無限に繰り返される可能性があることに注意してください。XML 文書の例では、実際にはエレメント B が 3 回、エレメント C が 5 回現れています。

XML 処理プログラムの例では、最初の XML PARSE ステートメントの処理プロシーチャーは、単にエレメント B と C の出現回数を計算するだけです。必要なストレージを割り振った後、プログラムは XML ペイロードを処理するために 2 番目の XML PARSE ステートメントを実行します。

XML スキーマ

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://example.org"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="G">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="A" type="xsd:string" maxOccurs="1" />
        <xsd:element name="B" type="xsd:int" maxOccurs="unbounded" />
        <xsd:element name="C" type="xsd:int" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML 文書

```
<?xml version="1.0" encoding="UTF-8"?>
<p:G xmlns:p="http://example.org" >
  <A>Hello</A>
  <B>1</B>
  <B>2</B>
  <B>3</B>
  <C>1</C>
  <C>2</C>
  <C>3</C>
  <C>4</C>
  <C>5</C>
</p:G>
```

XML 処理プログラム

```
Identification division.
Program-id. XMLProc.
Data division.
Working-storage section.
01 NB pic S9(9) binary value zero.
01 NC pic S9(9) binary value zero.
01 Gptr pointer.
01 Gsize pic 9(9) binary.
01 Heap0 pic 9(9) binary value zero.
Linkage section.
01 XML-Doc pic X(500000).
01 G.
  02 A pic x(5).
  02 B pic s9(9) occurs 1 to unbounded depending on NB.
  02 C pic s9(9) occurs 1 to unbounded depending on NC.
Procedure division using XML-Doc.
XML parse XML-Doc processing procedure CountElements
Move length of G to Gsize
Call "CEEGTST" using Heap0 Gsize Gptr omitted
Set address of G to Gptr
XML parse XML-doc processing procedure acquireContent
...
Goback.
CountElements.
If xml-event = 'START-OF-ELEMENT'
  Evaluate xml-text
  When 'B'
    Add 1 to NB
  When 'C'
    Add 1 to NC
  When other
    Continue
  End-evaluate
End-if.
End program XMLProc.
```

関連タスク

[95 ページの『無制限テーブルおよび無制限グループの処理』](#)

第 5 章 プログラム・アクションの選択と反復

このタスクについて

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

99 ページの『プログラム・アクションの選択』

107 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。

このタスクについて

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

99 ページの『アクションの選択項目のコーディング』

103 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 (*Enterprise COBOL for z/OS 言語解説書*)

EVALUATE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

アクションの選択項目のコーディング

使用 IF . . . ELSE 2 つの処理アクションの選択をコーディングします。(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ようにするには、EVALUATE ステートメントを使用します。

このタスクについて

```
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
```

注：次の例にあるように、NEXT SENTENCE と CONTINUE は後続のピリオドの位置次第では非常に異なる場合があります。

```
IF condition-r
  statement-1
ELSE
  CONTINUE or NEXT SENTENCE
END-IF
*> CONTINUE goes to statement-2
  statement-2
  statement-3.
*> NEXT SENTENCE goes to statement-4
  statement-4
```

詳細については NEXT SENTENCE、見る IF 文の中に *Enterprise COBOL for z/OS* 言語解説書。

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

[100 ページの『ネストされた IF ステートメントの使用』](#)

[101 ページの『EVALUATE ステートメントの使用』](#)

[103 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。

このタスクについて

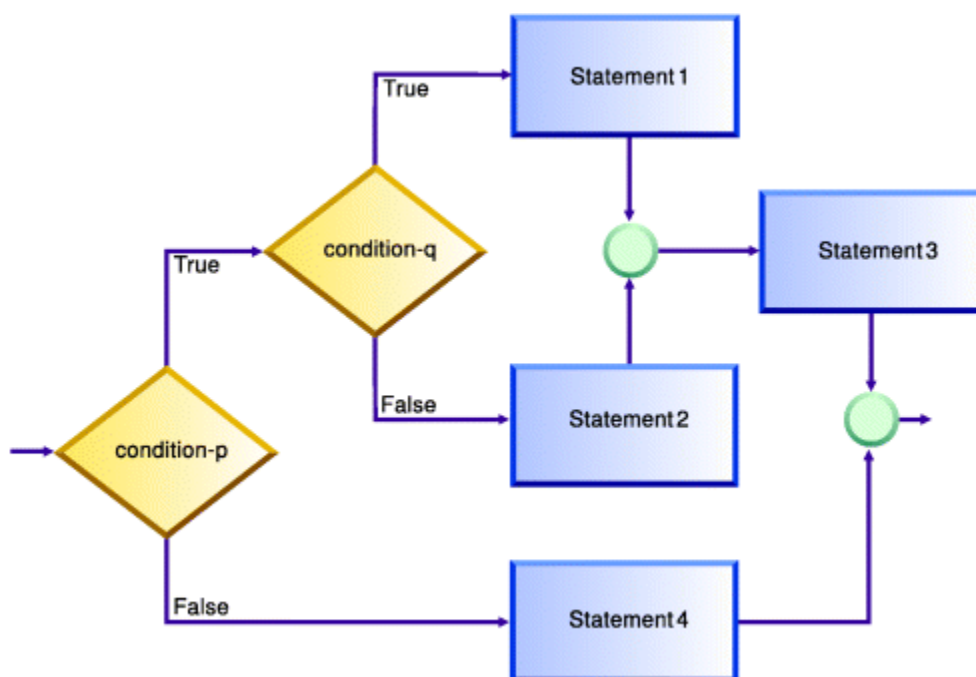
ただし、ネストされた IF ステートメントを多用しないでください。 明示範囲終了符号および字下げが役に立つとはいえ、ロジックをたどるのが困難になる可能性があります。 プログラムが 2 つを超える値について変数をテストしなければならない場合には、おそらく EVALUATE を使用する方が適切です。

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

[99 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 (*Enterprise COBOL for z/OS* 言語解説書)

EVALUATE ステートメントの使用

あなたは EVALUATE 声明ネストされた一連の IF 声明複数の条件をテストし、それぞれに異なるアクションを指定します。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。

このタスクについて

また、EVALUATE 複数の条件が同じ処理につながるようにするステートメント、次の例に示すように:

[102 ページの『例: THRU 句を使用した EVALUATE』](#)

[102 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。

[103 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に、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

[99 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 (*Enterprise COBOL for z/OS 言語解説書*)

一般比較条件 (*Enterprise COBOL for z/OS 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 それは選択対象;1、2、そして 3 THRU 6 は選択オブジェクト:

```
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*

この例では、ALSO 2つの選択対象を区切るフレーズ (True ALSO True) を選択し、各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します (たとえば、When A + B < 10 Also C = 10)。

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.
    Source-Computer. IBM-390. 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。

このタスクについて

以下の条件を指定できます。

- 次のような比較条件
 - 数字の比較
 - 英数字比較
 - DBCS 比較
 - 国別の比較
- 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 ALPHABETIC-LOWER
 - IS ALPHABETIC-UPPER
 - IS DBCS
 - IS 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
- 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。例: NOT (A IS EQUAL TO B)
 - 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

[104 ページの『スイッチおよびフラグ』](#)

関連タスク

[105 ページの『スイッチおよびフラグの定義』](#)

[106 ページの『スイッチとフラグのリセット』](#)

[60 ページの『非互換データの検査 \(数値のクラス・テスト\)』](#)

[148 ページの『国別 \(UTF-16\) データの比較』](#)

[169 ページの『有効な DBCS 文字に関するテスト』](#)

関連参照

一般比較条件 (*Enterprise COBOL for z/OS* 言語解説書)

クラス条件 (*Enterprise COBOL for z/OS* 言語解説書)

条件名項目の規則 (*Enterprise COBOL for z/OS* 言語解説書)

符号条件 (*Enterprise COBOL for z/OS* 言語解説書)

複合条件 (*Enterprise COBOL for z/OS* 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

[105 ページの『スイッチおよびフラグの定義』](#)

[106 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。

このタスクについて

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。

[105 ページの『例: スイッチ』](#)

[105 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".
```

の level-88 説明によると、Transaction-EOF は事実上いつ Transaction-EOF-Switch 値は「y」です。参照 Transaction-EOF の中に PROCEDURE DIVISION テストと同じ条件を表す Transaction-EOF-Switch = "y"。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  
End-if
```

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、メイン・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-Main-Record-Paragraph
  When Change-Transaction
    Perform Update-Existing-Record-Paragraph
  When Delete-Transaction
    Perform Delete-Main-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。

このタスクについて

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" "O" "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。

[106 ページの『例: スイッチをオンに設定する』](#)

[107 ページの『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

条件名値を条件変数に移行する 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以下に例示します。

例えば、SET 次の例のステートメントは、次のステートメントをコーディングするのと同じ効果があります。Move "y" to Transaction-EOF-Switch:

```

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

条件名値を条件変数に移行する 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以下に例示します。

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。

このタスクについて

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

[107 ページの『インラインまたはライン外 PERFORM の選択』](#)

[108 ページの『ループのコーディング』](#)

[109 ページの『テーブルのループ処理』](#)

[110 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 (*Enterprise COBOL for z/OS 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。

このタスクについて

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、別の手順への分岐と暗黙の戻りが必要になります。実行された手順が、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・フローの中でもう一度実行されます。この追加の実行を回避するためには、手順を通常の順次フローの外側 (例えば、GOBACK の後) に置くか、または手順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。

108 ページの『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

使用 PERFORM . . . TIMES 指定された回数だけプロシージャを実行するステートメント。

このタスクについて

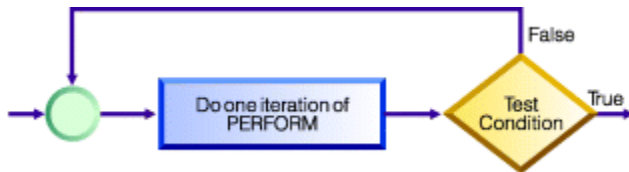
```
PERFORM 010-PROCESS-ONE-MONTH 12 TIMES
INSPECT . . .
```

上記の例では、制御が PERFORM ステートメントに達すると、手順 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。

使用 **PERFORM . . . UNTIL** 選択した条件が満たされるまでプロシージャを実行するステートメント。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 . . .
```

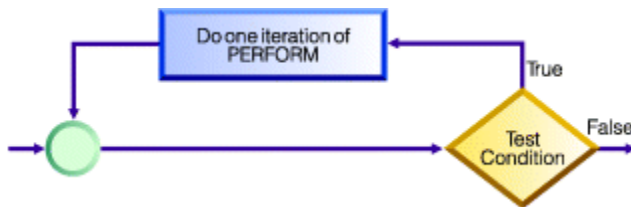
使用 **PERFORM . . . WITH TEST AFTER . . . UNTIL** プロシージャを少なくとも 1 回実行し、後続の実行の前にテストする場合は、ステートメントを使用します。このステートメントは、do-until 構造と同等です。



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 . . .
```

コントロールが **PERFORM** 声明、条件 **MONTH GREATER THAN 12** 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

あなたは **PERFORM . . . VARYING** テーブルを初期化するステートメント。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。

このタスクについて

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、段落のグループ、または単一のセクションまたはセクションのグループを実行するには、PERFORM . . . THRU 声明。

このタスクについて

使用すると PERFORM . . . THRU ステートメント、段落をコード化する-EXIT 一連の段落の終了点を明確に示すステートメント。

関連タスク

94 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

EXIT PERFORM または EXIT PERFORM CYCLE ステートメント

(Enterprise COBOL for z/OS 言語解説書)

EXIT PARAGRAPH または EXIT SECTION ステートメント

(Enterprise COBOL for z/OS 言語解説書)

第 6 章 スtring の処理

このタスクについて

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、以下のことを行うことができます。

- データ項目の結合または分割。
- スル終了Stringの操作 (文字のカウントや移動など)。
- 通常的位置 (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

- [111 ページの『データ項目の結合 \(STRING\)』](#)
- [113 ページの『データ項目の分割 \(UNSTRING\)』](#)
- [116 ページの『スル終了Stringの取り扱い』](#)
- [117 ページの『データ項目のサブStringの参照』](#)
- [120 ページの『データ項目の計算および置換 \(INSPECT\)』](#)
- [122 ページの『データ項目の変換 \(組み込み関数\)』](#)
- [126 ページの『データ項目の評価 \(組み込み関数\)』](#)
- [131 ページの『第 7 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 ステートメントを複数の MOVE ステートメントの代わりに使用できます。

このタスクについて

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。
- USAGE UTF-8 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 UTF-8 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。

[112 ページの『例: STRING ステートメント』](#)

関連タスク

[266 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 (*Enterprise COBOL for z/OS 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.bSMITHbbbbbb
444bSPRINGbST.,bCHICAGO,bILL.bbbbbbb
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.
```

なぜなら POINTER 分野 LINE-POS4 の値は、STRING ステートメントが実行されると、データは受信フィールドに移動されます RPT-LINE 文字位置 4 から始まります。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。

品	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 ステートメントを複数の MOVE ステートメントの代わりに使用できます。

このタスクについて

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。

- 送信フィールドで区切り文字の 1 つが検出されると、現在の受信フィールドの受信が停止され、次のフィールド（存在する場合）の受信が開始されます。（DELIMITED BY フレーズ）
- 送信フィールドで区切り文字が検出されると、現在の受信フィールドが受信を停止するフィールド（DELIMITER IN フレーズ）
- 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)

- 送信データ項目の終わりに達する前に受信フィールドがすべて入力された場合に実行されるアクション (ON OVERFLOW フレーズ)

送信データ項目と区切り文字 DELIMITED BY フレーズは、アルファベット、英数字、英数字編集、DBCS、国別、国別編集のカテゴリである必要があります。、または UTF-8。

受信データ項目は、アルファベット、英数字、数字、DBCS、国別のいずれかになります。、または UTF-8。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
- USAGE UTF-8 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 UTF-8 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

[114 ページの『例: UNSTRING ステートメント』](#)

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[266 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 (*Enterprise COBOL for z/OS 言語解説書*)

データのクラスおよびカテゴリ (*Enterprise COBOL for z/OS 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).
```

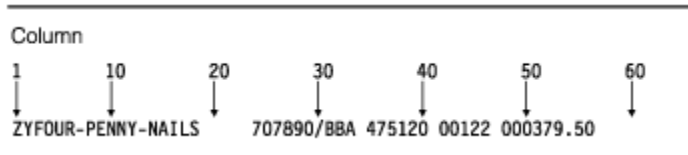
```

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。



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。

1. 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。
2. 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。
3. 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。
4. 桁 31 から 33 (88A) 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。
5. 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。上位桁の位置はゼロ (0)。
区切り文字は SPACE ただし、区切り文字の受信領域として定義されたフィールドがないため、位置 47
のスペースはバイパスされます。
- 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が処理され、INV-RCD 調査されていないが、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

さまざまな仕組みを使用して、ヌル終了ストリング (例えば C プログラムとの間で受け渡されるストリング) を構成し、操作することができます。

このタスクについて

例えば、以下のことを行うことができます。

- 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。
- ループ内のフィールド内の各文字をチェックします (PERFORM 参照修飾子を使用してフィールド内の各文字を調べることができます。source-field (I:1)。

117 ページの『例: ヌル終了ストリング』

関連タスク

629 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 (*Enterprise COBOL for z/OS* 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。

このタスクについて

注: UTF-8 でエンコードされた文字ストリング引数のサブストリングを取得するには、[163 ページの『組み込み関数を使用した UTF-8 エンコード・データの処理』](#)で説明されているように、USUBSTR 関数を使用してください。

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、コロンで分離された 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) 必要なサブストリングの長さ (文字位置)

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。

あなたはできるの部分文字列を参照する USAGE NATIONAL 参照修飾子を使用して、国別グループ、国別編集データ項目、数値編集データ項目、国別浮動小数点データ項目、および国別 10 進データ項目を含むデータ項目を作成します。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す。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。

[119 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。

[120 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超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。

の SSRANGE コンパイラ オプションは範囲外の参照修飾子を検出し、実行時メッセージで違反をフラグ付けします。

関連概念

[118 ページの『参照修飾子』](#)

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[76 ページの『テーブル内の項目の参照』](#)

関連参照

[446 ページの『SSRANGE』](#)

参照変更 (*Enterprise COBOL for z/OS 言語解説書*)

関数定義 (*Enterprise COBOL for z/OS 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。例:

```
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)
```

[119 ページの『例: 参照修飾子としての演算式』](#)

[120 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割り当て \(ACCEPT\)』](#)

[117 ページの『データ項目のサブストリングの参照』](#)

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

関連参照

参照変更 (*Enterprise COBOL for z/OS 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、次のコードフラグメントは、Customer-Record データ項目に移動される WS-name。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。例:

```
Move Customer-Record(Function Integer(Function Sqrt(I)): ) to WS-name
```

関連参照

INTEGER (*Enterprise COBOL for z/OS 言語解説書*)

INTEGER-PART (*Enterprise COBOL for z/OS 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。

このタスクについて

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、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、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、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。

121 ページの『例: INSPECT ステートメント』

関連概念

132 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 (*Enterprise COBOL for z/OS 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。先頭のゼロの回数 (0) が発生するデータ項目は、COUNTR。文字 C の最初のインスタスの後に続く文字 A の最初のインスタスは、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
"Twas BR	0	"Twas 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 ? (存在する場合) 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
      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    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、16 進数字または 2 進数字に) 変換できます。また、16 進文字ストリングまたはビット文字ストリングを英数字データ項目に変換することもできます。

このタスクについて

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。

[121 ページの『例: INSPECT ステートメント』](#)

関連タスク

[122 ページの『大/小文字の変更 \(UPPER-CASE、LOWER-CASE\)』](#)

[123 ページの『逆順への変換 \(REVERSE\)』](#)

[123 ページの『数値への変換 \(NUMVAL、NUMVAL-C、NUMVAL-F\)』](#)

[124 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』](#)

[125 ページの『16 進データまたはビット・データへの変換 \(HEX-OF、BIT-OF\)』](#)

[125 ページの『16 進データまたはビット・データからの変換 \(HEX-TO-CHAR、BIT-TO-CHAR\)』](#)

大/小文字の変更 (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。

このタスクについて

```
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。

注: UPPER-CASE および LOWER-CASE 組み込み関数は、UTF-8 エンコード・データを含む英数字引数をサポートしません。

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 \(ACCEPT\)』](#)

[36 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。

このタスクについて

```
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 の中の文字の順序を逆にします。開始値が JOHNSONbbb 文を実行した後の値は bbbNOSNH0J、どこ b 空白を表します。

関連概念

[132 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-F)

NUMVAL、NUMVAL-C、および NUMVAL-F 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。

このタスクについて

```
01 R      Pic x(20)  Value "- 1234.5678".  
01 S      Pic x(20)  Value " $12,345.67CR".  
01 T      Pic x(20)  Value "+ 12.345678E+2".  
01 Total  Usage is Comp-1.  
...  
Compute Total = Function Numval(R) + Function Numval-C(S) + Function Numval-F(T)
```

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-F は、上の例で示されているように、引数に指数値が含まれているときに使用します。文字ストリングの前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、NUMVAL-C そして NUMVAL-F 互換モードでは long (64 ビット) 浮動小数点値を返し、拡張精度を返します。(128 ビット)拡張モードでの浮動小数点値。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)。内部的に NUMVAL は長精度浮動小数点計算を使用して、与えられた数を出力に変換しますので、NUMVAL、NUMVAL-C、または NUMVAL-F の引数が 15 桁を超える場合には、ARITH(EXTEND) コンパイラ・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。そうしないと、予期せぬ仕方で結果が精度を失うことがあります。

NUMVAL、NUMVAL-C、または NUMVAL-F を使用する場合は、数値データを固定形式で静的に定義したり、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。例:

```
+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

[53 ページの『数値データの形式』](#)

[58 ページの『データ形式の変換』](#)

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

関連参照

[58 ページの『変換および精度』](#)

[381 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。

このタスクについて

例えば、以下のコードは EBCDIC スtring を ASCII String に変換します。

```
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

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

16 進データまたはビット・データへの変換 (HEX-OF、BIT-OF)

HEX-OF 組み込み関数または BIT-OF 組み込み関数を使用して、任意の型のデータを 16 進数字または 2 進数字に変換できます。

このタスクについて

HEX-OF 組み込み関数は、あらゆるタイプのデータを、変換対象のデータの基礎となるバイト値を 16 進形式で表す、人間が読み取れる 16 進数字の String (「0」から「9」、「A」から「F」、および「a」から「f」) に変換するために使用できます。出力 16 進 String の長さ (バイト単位) は、入力引数 String の長さ (バイト単位) の 2 倍です。

例えば、FUNCTION HEX-OF('Hello, world!') は「C8859393966B40A6969993845A」を返します。

注: 最初の 2 つの 16 進数字「C8」は、文字「H」の EBCDIC エンコードに対応します。

HEX-OF 組み込み関数に対する引数には、リテラル、データ項目、または組み込み関数の結果を指定できます。

BIT-OF 組み込み関数は、あらゆるタイプのデータを、変換対象のデータの基礎となるバイト値をビット・String 形式で表す、人間が読み取れる 2 進数字の String (「0」または「1」) に変換するために使用できます。出力ビット・String の長さ (バイト単位) は、入力引数 String の長さ (バイト単位) の 8 倍です。

例えば、FUNCTION BIT-OF('Hello, world!') は
「11001000100001011001001110010011100101100110101101011010000001010011010010110
10011001100100111000010001011010」を返します。

注: 出力 String の最初の 8 文字「11001000」は、16 進値 x'C8' に対応します。この値は、上に示した HEX-OF 組み込み関数の出力に一致し、文字「H」の EBCDIC エンコードに対応します。

BIT-OF 組み込み関数に対する引数には、リテラル、データ項目、または組み込み関数の結果を指定できます。

関連参照

BIT-OF (*Enterprise COBOL for z/OS* 言語解説書)

HEX-OF (*Enterprise COBOL for z/OS* 言語解説書)

16 進データまたはビット・データからの変換 (HEX-TO-CHAR、BIT-TO-CHAR)

HEX-TO-CHAR 組み込み関数または BIT-TO-CHAR 組み込み関数を使用して、16 進文字 String (文字「0」から「9」、「A」から「F」、および「a」から「f」で構成されます) またはビット文字 String (文字「0」と「1」で構成されます) を英数字データ項目に変換できます。

このタスクについて

HEX-TO-CHAR

HEX-TO-CHAR 組み込み関数は、16 進文字ストリング (文字「0」から「9」、「A」から「F」、および「a」から「f」) で構成される文字ストリングを、入力文字ストリング内の 16 進数字に対応するバイトで構成される英数字文字ストリングに変換するために使用できます。

例えば、

```
MOVE 'FFAABB' TO MY-HEX-DATA
```

FUNCTION HEX-TO-CHAR(MY-HEX-DATA) は、値が x'FFAABB' の文字ストリングを返します。

HEX-TO-CHAR 組み込み関数に対する引数には、英数字リテラル、英数字データ項目、または英数字グループ項目を指定できます。引数の長さは、2 バイトの倍数でなければなりません。

BIT-TO-CHAR

BIT-TO-CHAR 組み込み関数は、文字「0」と「1」で構成される文字ストリングを、入力文字ストリング内の「0」と「1」の文字シーケンスで示されるビット・パターンに対応するバイトで構成される英数字文字ストリングに変換するために使用できます。

例えば、

```
MOVE '1111001000000110' TO MY-BIT-DATA
```

FUNCTION BIT-TO-CHAR(MY-BIT-DATA) は、値が x'F206' の文字ストリングを返します。

BIT-TO-CHAR 組み込み関数に対する引数には、英数字リテラル、英数字データ項目、または英数字グループ項目を指定できます。引数の長さは、8 バイトの倍数でなければなりません。

関連参照

BIT-TO-CHAR (*Enterprise COBOL for z/OS 言語解説書*)

HEX-TO-CHAR (*Enterprise COBOL for z/OS 言語解説書*)

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。

このタスクについて

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、そして ORD-MIN 一連のデータ項目の中で最大と最小の項目を見つける。USAGE NATIONAL データ項目
- LENGTH データ項目の長さを調べるには、USAGE NATIONAL データ項目、そして BYTE-LENGTHDBCS データ項目を含むデータ項目の長さをバイト単位で調べる
- 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

[132 ページの『Unicode および言語文字のエンコード』](#)

関連タスク

[127 ページの『照合シーケンスに関する単一文字の評価』](#)

[127 ページの『最大または最小データ項目の検出』](#)

[129 ページの『データ項目の長さの検出』](#)

[130 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

ある特定の 1 文字 (英字または英数字) が照合シーケンス内に出現する位置を調べるには、その文字を引数として指定して 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。

このタスクについて

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。例:

```
INITIALIZE Customer-Name REPLACING ALPHABETIC BY Function Char(65)
```

文字に関連付けられた序数は、文字の 16 進値の数値 (10 進数) と同じではありません。例えば、EBCDIC 照合シーケンスを使用している場合、序数 X'00' はゼロではなく 1 です。同様に、序数 X'FF' は 255 ではなく 256 です。そのため、EBCDIC 照合シーケンス範囲を使用している場合に ORD 組み込み関数から返された序数は、0 から 255 (有効な EBCDIC 文字の 16 進値の 10 進数値) ではなく、1 から 256 です。

関連参照

CHAR (*Enterprise COBOL for z/OS 言語解説書*)

ORD (*Enterprise COBOL for z/OS 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、照合シーケンスに従って評価します。

このタスクについて

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 ".
```

次の文は、VALLEJObbbb 最初の 10 文字の位置に Customer-record、どこ b 空白を表します:

```
Move Function Max(Arg1 Arg2 Arg3) To Customer-record(1:10)
```

使用した場合 MIN 代わりに、THOMASbbbb 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。もしあなたが ORD-MAX 関数を前例のように使用すると、数値関数は送信元として有効ではないため、コンパイラはエラーメッセージを発行します。MOVE 声明。前の例と同じ引数を使用して、ORD-MAX 次のように使用できます。

```
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した場合、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

[61 ページの『算術の実行』](#)

[94 ページの『組み込み関数を使用したテーブル項目の処理』](#)

[128 ページの『英数字または国別関数によって戻される可変結果』](#)

関連参照

MAX (*Enterprise COBOL for z/OS* 言語解説書)

MIN (*Enterprise COBOL for z/OS* 言語解説書)

ORD-MAX (*Enterprise COBOL for z/OS* 言語解説書)

ORD-MIN (*Enterprise COBOL for z/OS* 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。

このタスクについて

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。
- 文字列「fbbbb'は R3、どこ b 空白を表します。(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。
- 文字列「gbbbbbbbbb'は R3。(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。

- 文字列 NX"0066 0020 0020 0020 0020"（「f」の国別文字に相当する *bbbb'*、どこ *b*（空白を表す）は、ここでは読みやすくするためにスペースを追加した 16 進表記で示されており、R3。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え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

[127 ページの『最大または最小データ項目の検出』](#)

[61 ページの『算術の実行』](#)

関連参照

MAX (*Enterprise COBOL for z/OS* 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。また、BYTE-LENGTH 関数を使用して、項目の長さをバイト単位で判別することもできます。

このタスクについて

LENGTH 組み込み関数

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、顧客名を入れるレコード内のフィールドにデータ項目を移動する例を示しています。

```
Move Customer-name To Customer-record(1:Function Length(Customer-name))
```

BYTE-LENGTH 組み込み関数

BYTE-LENGTH 関数は、国別項目、UTF-8 項目、英数字項目、または DBCS リテラルの長さを、引数の長さに等しい整数としてバイト単位で返します。

LENGTH OF 特殊レジスター

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返します。コーディング `Function Length(Customer-name)` または `LENGTH OF Customer-name` 英数字項目の場合も同じ結果を返します。Customer-name バイト単位。BYTE-LENGTH 関数は、すべての引数タイプについて、LENGTH OF 特殊レジスターと同じ結果を返します。

あなたは LENGTH そして BYTE-LENGTH 機能算術式が許可されている場合にのみ使用できます。ただし、LENGTH OF より多様なコンテキストで特別なレジスターになります。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。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

61 ページの『算術の実行』

84 ページの『可変長テーブルの作成 (DEPENDING ON)』

94 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

BYTE-LENGTH (*Enterprise COBOL for z/OS* 言語解説書)

LENGTH (*Enterprise COBOL for z/OS* 言語解説書)

LENGTH OF (*Enterprise COBOL for z/OS* 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。

このタスクについて

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、秒の小数部を扱いません。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 (*Enterprise COBOL for z/OS* 言語解説書)

第 7 章 国際環境でのデータの処理

Enterprise COBOL では、実行時の文字データとして UTF-16 と UTF-8 の Unicode エンコードがサポートされます。UTF-16 は、固定幅の Unicode エンコード方式で、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。

このタスクについて

以下の COBOL 機能を使用して、国別データ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:
 - 文字データ型は、USAGE NATIONAL 条項と PICTURE 国別、国別編集、または数値編集のカテゴリのデータを定義する句
 - 数値データ型は、USAGE NATIONAL 条項と PICTURE 数値データ項目を定義する句（国別小数点）または外部浮動小数点データ項目（国別浮動小数点項目）
 - 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 に英数字または 2 バイト文字セット (DBCS) 文字列を USAGE NATIONAL (UTF-16)
 - DISPLAY-OF 国別文字列を USAGE DISPLAY で選択したコードページ (EBCDIC、ASCII、EUC、または UTF-8)
 - 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ー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:
 - CODEPAGE を使用すると、プログラムで英数字および DBCS データに使用するコード・ページを指定できます。
 - 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。

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。

UTF-8 は、データ伝送およびワールド・ワイド・ウェブ関連のデータ形式 (HTML や JSON など) でよく使用される可変長の Unicode エンコードです。UTF-8 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、UTF-8 データ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できます。

- データ型およびリテラル:
 - 文字データ型。USAGE UTF-8 節や、カテゴリー UTF-8 のデータを定義する PICTURE 節で定義します。

- リテラル接頭部 U または UX で指定される UTF-8 リテラル
 - U リテラルには特別な Unicode エスケープシーケンス \u が含まれる場合があります。うーんまたは \U00 うーん個々の Unicode コードポイントを指定する。うーんそしてあああああ 16 進数のシーケンスとして指定された Unicode コードポイント値です。
- 形象定数 ALL *utf-8-literal*
- 形象定数 QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO。これらは、UTF-8 文字コンテキストで使用されるときには UTF-8 文字 (UTF-8) 値を持ちます。
- COBOL ステートメントおよび UTF-8 データに関する関連参照に示されている COBOL ステートメント
 - 組み込み関数
 - NATIONAL-OF 変換する UTF-8 文字列を USAGE NATIONAL (UTF-16)
 - DISPLAY-OF 変換する UTF-8 文字列を USAGE DISPLAY 選択したコードページ (EBCDIC、ASCII、EUC) で国別データを変換し、USAGE UTF-8 (UTF-8)
 - 組み込み関数および UTF-8 データに関する関連参照に示されているその他の組み込み関数
 - の GROUP-USAGE UTF-8 句は、以下のみを含むグループを定義します。USAGE UTF-8 データ項目であり、基本カテゴリのように動作する UTF-8 ほとんどの操作の項目

英数字または国別データ項目から UTF-8 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 UTF-8 データ項目へ移動させるとき、またはこれらの項目を UTF-8 データ項目と比較するとき、コンパイラは (ほとんどの場合に) この変換を実行します。

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

[141 ページの『国別グループ』](#)

関連タスク

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

[162 ページの『UTF-16 \(国別\) データ・タイプを使用して UTF-8 データを処理』](#)

[167 ページの『中国語 GB 18030 データの処理』](#)

[148 ページの『国別 \(UTF-16\) データの比較』](#)

[168 ページの『DBCS サポート用のコーディング』](#)

[867 ページの『2 バイト文字セット \(DBCS\) データの変換』](#)

関連参照

[135 ページの『COBOL ステートメントと国別データ』](#)

[138 ページの『組み込み関数と国別データ』](#)

[385 ページの『CODEPAGE』](#)

[421 ページの『NSYMBOL』](#)

データのクラスおよびカテゴリー (*Enterprise COBOL for z/OS* 言語解説書)

データ・カテゴリーと PICTURE ルール

(*Enterprise COBOL for z/OS* 言語解説書)

MOVE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

一般比較条件 (*Enterprise COBOL for z/OS* 言語解説書)

Unicode および言語文字のエンコード

Enterprise COBOL 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關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。Enterprise COBOL 用途 UTF-16 (CCSID 1200) ビッグエンディアン形式国別リテラルやデータ項目の表現として USAGE NATIONAL。Enterprise COBOL 用途 UTF-8 (CCSID 1208) の表現として UTF-8 リテラルとデータ項目は USAGE UTF-8。

UTF-8 ASCII 不変文字の a～z、A～Z、0～9、および '@、. + - = / * ()' などの特定の特殊文字を表します。として UX'nn' どころ X'nn' ASCII での文字の表現です。UTF-16 これらの文字は次のように表される。NX'00nn'。

例えば、文字列 'ABC' で表される UTF-16 として NX'004100420043'。で UTF-8、'ABC' は次のように表される。X'414243'。

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 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。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 Enterprise COBOL 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。

関連タスク

[142 ページの『国別 \(Unicode\) 表現との変換』](#)

関連参照

[148 ページの『文字データの保管』](#)

文字セットとコード・ページ (*Enterprise COBOL for z/OS 言語解説書*)

COBOL での国別データ (Unicode) の使用

Enterprise COBOL では、いくつかの方法で国別 (UTF-16) データを指定できます。

このタスクについて

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)

さらに、明示的または暗黙的に次のデータ項目のみを含む国別グループを定義することもできます。USAGE NATIONAL ほとんどの操作では、基本カテゴリの国別データ項目と同じように動作します。

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

[132 ページの『Unicode および言語文字のエンコード』](#)

[141 ページの『国別グループ』](#)

関連タスク

[134 ページの『国別データ項目の定義』](#)

[134 ページの『国別リテラルの使用』](#)

[140 ページの『国別文字形象定数の使用』](#)

[141 ページの『国別数値データ項目の定義』](#)

[145 ページの『国別グループの使用』](#)

[142 ページの『国別 \(Unicode\) 表現との変換』](#)

[148 ページの『国別 \(UTF-16\) データの比較』](#)

関連参照

148 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 (Enterprise COBOL for z/OS 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。

このタスクについて

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、PICTURE 次の記号のそれぞれを少なくとも 1 つ含む句。

- 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。数値編集データ項目は、USAGE NATIONAL 数値編集項目を使用するのと同じように、USAGE DISPLAY。

また、データ項目を数値編集として定義するには、BLANK WHEN ZERO 数値として定義されている基本項目の節 PICTURE 句。

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

51 ページの『数値データの表示』

関連参照

421 ページの『NSYMBOL』

BLANK WHEN ZERO 節 (Enterprise COBOL for z/OS 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。

このタスクについて

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、次の文も、UTF-16 文字 'AB' (16 進数のビットパターンではない) C1C2) に Y、どこ Y と定義されている USAGE NATIONAL:

```
Move X"C1C2" to Y
```

国別リテラルは使用できません。SPECIAL-NAMES 段落またはプログラム名として使用します。オブジェクト指向メソッドに名前を付けるには、国別リテラルを使用できます。METHOD-ID 段落またはメソッド名を指定する INVOKE 声明。

関連タスク

25 ページの『リテラルの使用』

関連参照

421 ページの『NSYMBOL』

国別リテラル (Enterprise COBOL for z/OS 言語解説書)

COBOL ステートメントと国別データ

国のデータは、PROCEDURE DIVISION およびコンパイラ指示ステートメントを以下の表に示します。

表 15. COBOL ステートメントおよび国別データ			
COBOL ステートメント	国別にできるもの	コメント	詳細の参照先
ACCEPT	<i>identifier-1, identifier-2</i>	<i>identifier-1</i> はから変換されたネイティブコードページは、CODEPAGE コンパイルオプション入力が CONSOLE。	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。		62 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>		623 ページの『データの受け渡し』

表 15. COBOL ステートメントおよび国別データ (続き)			
COBOL ステートメント	国別にできるもの	コメント	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。		62 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>		481 ページの『第 21 章 コンパイラ指示ステートメント』
DISPLAY	<i>identifier-1</i>	<i>identifier-1</i> が EBCDIC に変換されるのは、CONSOLE mnemonic-name が直接的または間接的に指定されている場合のみです。	36 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。		62 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> 。	指定する場合 REPLACING NATIONAL または REPLACING NATIONAL-EDITED、 <i>identifier-2</i> または <i>literal-1</i> でなければなりません移動の送信オペランドとして有効 <i>identifier-1</i> 。	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。 (<i>identifier-2</i> 、TALLYING 整数データ項目、USAGE NATIONAL。	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。	120 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> 。		763 ページの『メソッドの呼び出し (INVOKE)』
JSON PARSE	<i>identifier-2</i> (ターゲット・データ項目)、 <i>identifier-3</i> (NAME ID)、 <i>literal-1</i> (NAME 代替)、 <i>identifier-4</i> (SUPPRESS ID)	<i>identifier-1</i> は、国別データ項目としてサポートされません。	673 ページの『第 36 章 JSON 入力の処理』

表 15. COBOL ステートメントおよび国別データ (続き)			
COBOL ステートメント	国別にできるもの	コメント	詳細の参照先
MERGE	マージ・キー	COLLATING SEQUENCE 句は適用されません。	252 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。	32 ページの『基本データ項目への値の割り当て (MOVE)』 33 ページの『グループ・データ項目への値の割り当て (MOVE)』
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。		62 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。	92 ページの『二分探索 (SEARCH ALL)』
SORT	ソート・キー	COLLATING SEQUENCE 句は適用されません。	252 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。 (<i>identifier-4</i> 、POINTER 整数データ項目、USAGE NATIONAL。	<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。	111 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。		62 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ます、それぞれのリテラルは国別でなければなりません。	113 ページの『データ項目の分割 (UNSTRING)』

表 15. COBOL ステートメントおよび国別データ (続き)			
COBOL ステートメント	国別にできるもの	コメント	詳細の参照先
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)		727 ページの『第 39 章 XML 出力の生成』
XML PARSE	<i>identifier-1</i> (XML 文書)	XML-NTEXT 特殊レジスタには、構文解析時に国別文字文書フラグメントが入ります。XML-NNAMESPACE そして XML-NNAMESPACE-PREFIX 特殊レジスタには、関連付けられた名前空間識別子と名前空間プレフィックス (存在する場合) が国別文字で格納されます。	687 ページの『第 38 章 XML 入力の処理』

関連タスク

[49 ページの『数値データの定義』](#)

[51 ページの『数値データの表示』](#)

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

[148 ページの『国別 \(UTF-16\) データの比較』](#)

関連参照

[385 ページの『CODEPAGE』](#)

データのクラスおよびカテゴリー (*Enterprise COBOL for z/OS* 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 16. 組み込み関数および国別文字データ		
組み込み関数	関数のタイプ	詳細の参照先
BIT-OF	英数字	BIT-OF (<i>Enterprise COBOL for z/OS</i> 言語解説書)
BYTE-LENGTH	整数	129 ページの『データ項目の長さの検出』
CONTENT-OF	英数字、国別、UTF-8、整数、または数値	CONTENT-OF (<i>Enterprise COBOL for z/OS</i> 言語解説書)
DISPLAY-OF	英数字	143 ページの『国別の英数字への変換 (DISPLAY-OF)』

表 16. 組み込み関数および国別文字データ (続き)		
組み込み関数	関数のタイプ	詳細の参照先
FORMATTED-CURRENT-DATE, FORMATTED-DATE, FORMATTED-TIME, FORMATTED-DATETIME	英数字、国別、または UTF-8	• FORMATTED-CURRENT-DATE (<i>Enterprise COBOL for z/OS</i> 言語解説書) • FORMATTED-DATE (<i>Enterprise COBOL for z/OS</i> 言語解説書) • FORMATTED-TIME (<i>Enterprise COBOL for z/OS</i> 言語解説書) • FORMATTED-DATETIME (<i>Enterprise COBOL for z/OS</i> 言語解説書)
HEX-OF	英数字	HEX-OF (<i>Enterprise COBOL for z/OS</i> 言語解説書)
INTEGER-OF-FORMATTED-DATE	整数	INTEGRE-OF-FORMATTED-DATE (<i>Enterprise COBOL for z/OS</i> 言語解説書)
LENGTH	整数	129 ページの『データ項目の長さの検出』
LOWER-CASE, UPPER-CASE	国別	122 ページの『大/小文字の変更 (UPPER-CASE、LOWER-CASE)』
NUMVAL, NUMVAL-C, NUMVAL-F	数字	123 ページの『数値への変換 (NUMVAL、NUMVAL-C、NUMVAL-F)』
MAX, MIN	国別	127 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	127 ページの『最大または最小データ項目の検出』
REVERSE	英数字または国別文字	REVERSE (<i>Enterprise COBOL for z/OS</i> 言語解説書)
SECONDS-FORM-FORMATTED-TIME	数字	SECONDS-FROM-FORMATTED-TIME (<i>Enterprise COBOL for z/OS</i> 言語解説書)
TEST-FORMATTED-DATETIME	整数	TEST-FORMATTED-DATETIME (<i>Enterprise COBOL for z/OS</i> 言語解説書)
TEST-NUMVAL, TEST-NUMVAL-C, TEST-NUMVAL-F	整数	• TEST-NUMVAL (<i>Enterprise COBOL for z/OS</i> 言語解説書) • TEST-NUMVAL-C (<i>Enterprise COBOL for z/OS</i> 言語解説書) • TEST-NUMVAL-F (<i>Enterprise COBOL for z/OS</i> 言語解説書)
TRIM	英数字または国別文字	TRIM (<i>Enterprise COBOL for z/OS</i> 言語解説書)
ULENGTH	整数	ULENGTH (<i>Enterprise COBOL for z/OS</i> 言語解説書)
UPOS	整数	UPOS (<i>Enterprise COBOL for z/OS</i> 言語解説書)
USUBSTR	英数字または国別文字	USUBSTR (<i>Enterprise COBOL for z/OS</i> 言語解説書)
USUPPLEMENTARY	整数	USUPPLEMENTARY (<i>Enterprise COBOL for z/OS</i> 言語解説書)
UVALID	整数	UVALID (<i>Enterprise COBOL for z/OS</i> 言語解説書)
UWIDTH	整数	UWIDTH (<i>Enterprise COBOL for z/OS</i> 言語解説書)

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

[49 ページの『数値データの定義』](#)

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

関連参照

引数 (*Enterprise COBOL for z/OS 言語解説書*)

データのクラスおよびカテゴリー (*Enterprise COBOL for z/OS 言語解説書*)

組み込み関数 (*Enterprise COBOL for z/OS 言語解説書*)

国別文字形象定数の使用

比喩的な定数を使うことができます ALL 国語のリテラル国民的性格を必要とする文脈において。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。

このタスクについて

比喩的な定数を使うことができます QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO 国別の特徴を必要とする文脈では、例えば MOVE ステートメント、暗黙の移動、または国別オペランドを持つ関係条件。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。

国別文字が必要となるコンテキストで形象定数 QUOTE を使用するとき QUOTE コンパイラー・オプションが有効であれば、値は NX'0022' となります。APOST コンパイラー・オプションが有効であれば、値は NX'0027' となります。

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。

制限: 使用してはいけません HIGH-VALUE または、HIGH-VALUE あるデータ表現から別のデータ表現への値の変換をもたらす方法 (例えば、USAGE DISPLAY そして USAGE NATIONAL)。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 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)。次の例を考えてみましょう。

```
01 natl-data PIC NN Usage National.
01 alph-data PIC XX.
. . .
MOVE HIGH-VALUE TO natl-data, alph-data
IF natl-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

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

[148 ページの『国別 \(UTF-16\) データの比較』](#)

関連参照

形象定数 (*Enterprise COBOL for z/OS 言語解説書*)

DISPLAY-OF (*Enterprise COBOL for z/OS 言語解説書*)

Support for Unicode: Using Unicode Services

国別数値データ項目の定義

データ項目を定義するには USAGE NATIONAL 国別文字で表現された数値データを保持する句 (UTF-16)。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。

このタスクについて

国別小数点項目を定義するには、PICTURE 記号のみを含む句 9、P、S、そして V。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

[49 ページの『数値データの定義』](#)

[51 ページの『数値データの表示』](#)

関連参照

SIGN 節 (*Enterprise COBOL for z/OS 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、次のように記述される基本カテゴリの国別項目として再定義されたものとして処理されます。PIC N(m)、どこメートルは、国家 (UTF-16) 文字をグループに追加します。

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。このような操作 (例えば、MOVE CORRESPONDING そして INITIALIZE) 国グループ内の基本項目を認識または処理します。

可能であれば、英数字グループの代わりに国別グループを使用してください。USAGE NATIONAL アイテム。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:グループ項目は、USAGE NATIONAL グループレベルでの便利な略語として USAGE グループ内の各基本データ項目の。そのようなグループはないしかし、国別グループではなく英数字グループであり、移動や比較などの多くの操作では、基本データ項目のように動作します。USAGE DISPLAY (ただし、データの編集や変換は行われません)。

関連タスク

[33 ページの『グループ・データ項目への値の割り当て \(MOVE\)』](#)

[111 ページの『データ項目の結合 \(STRING\)』](#)

[113 ページの『データ項目の分割 \(UNSTRING\)』](#)

[120 ページの『データ項目の計算および置換 \(INSPECT\)』](#)

[145 ページの『国別グループの使用』](#)

関連参照

GROUP-USAGE 節 (*Enterprise COBOL for z/OS* 言語解説書)

国別 (Unicode) 表現との間の変換

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。

このタスクについて

あなたはできる暗黙的にアルファベットを変換し、英数字、UTF-8、DBCS、または整数データを国別データに変換するには、MOVE 声明。暗黙的な変換は、他の COBOL 文でも行われます。IF 声明英数字データ項目をデータ項目には USAGE NATIONAL。

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、CODEPAGE コンパイラ・オプション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

[142 ページの『英数字、DBCS、および整数から国別への変換 \(MOVE\)』](#)

[143 ページの『英数字または DBCS から国別への変換 \(NATIONAL-OF\)』](#)

[143 ページの『国別の英数字への変換 \(DISPLAY-OF\)』](#)

[144 ページの『デフォルト・コード・ページのオーバーライド』](#)

[148 ページの『国別 \(UTF-16\) データの比較』](#)

関連参照

[385 ページの『CODEPAGE』](#)

[144 ページの『変換例外』](#)

英数字、DBCS、および整数から国別への変換 (MOVE)

M 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。

このタスクについて

次の種類のデータをカテゴリー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
- UTF-8
- 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
- 浮動小数点数（浮動小数点数）を表示 USAGE DISPLAY)
- 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

- [32 ページの『基本データ項目への値の割り当て \(MOVE\)』](#)
- [33 ページの『グループ・データ項目への値の割り当て \(MOVE\)』](#)
- [51 ページの『数値データの表示』](#)
- [168 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 (*Enterprise COBOL for z/OS 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、UTF-8、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。ソースコードページを次のように指定します二番目ソースが有効なコードページとは異なるコードページでエンコードされている場合は引数とともに CODEPAGE コンパイラオプション。

このタスクについて

[144 ページの『例: 国別データとの間の変換』](#)

関連タスク

- [162 ページの『UTF-16 \(国別\) データ・タイプを使用して UTF-8 データを処理』](#)
- [167 ページの『中国語 GB 18030 データの処理』](#)
- [170 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

[385 ページの『CODEPAGE』](#)
NATIONAL-OF (*Enterprise COBOL for z/OS 言語解説書*)

国別の英数字への変換 (DISPLAY-OF)

使用 DISPLAY-OF 国のデータを英数字 (USAGE DISPLAY) 指定したコードページで表される文字列 2 番目の引数として。

このタスクについて

2 番目の引数を省略すると、出力コードページは有効だったのは CODEPAGE ソースをコンパイルしたときのコンパイラ オプション。

1 バイト文字セット (SBCS) 文字と DBCS 文字が組み合わせられた 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に SBCS 文字と DBCS 文字が混在していることがあります。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。

[144 ページの『例: 国別データとの間の変換』](#)

関連タスク

[162 ページの『UTF-16 \(国別\) データ・タイプを使用して UTF-8 データを処理』](#)

[167 ページの『中国語 GB 18030 データの処理』](#)

関連参照

DISPLAY-OF (*Enterprise COBOL for z/OS* 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、CODEPAGE オプション値として指定された CCSID 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。

このタスクについて

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、CODEPAGE コンパイラ・オプションで有効なコード・ページとは異なる場合、暗黙の変換を伴う操作 (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、CODEPAGE コンパイラ・オプションで指定された EBCDIC コード・ページが使用されることを想定しています。

関連参照

[385 ページの『CODEPAGE』](#)

変換例外

国別データと英数字データとの間の暗黙変換または明示変換が失敗して、重大度 3 の言語環境条件が生成されることがあります。

暗黙的または明示的に指定したコード・ページが有効なコード・ページではない場合、失敗することがあります。

ターゲットの CCSID に対になる文字が存在しない文字については、変換例外は発生しません。このような文字はターゲット・コード・ページの置換文字に変換されます。

関連参照

[385 ページの『CODEPAGE』](#)

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。

```
CBL CODEPAGE(00037)
* . . .
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, 00875)
        to Data-in-Unicode
* . . . process Data-in-Unicode here . . .
      Move function Display-of(Data-in-Unicode, 00875)
        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、CCSID 00875 (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

```

上記の MOVE ステートメントは、Data-in-Greek を Unicode 表現に変換しますが、これは CCSID 00037 (米国 英語) から UTF-16 への変換に基づいています。Data-in-Greek は CCSID 00875 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。

CODEPAGE コンパイラー・オプションを正しく CCSID 00875 に設定することができた場合 (つまり、プログラムの残りの部分も EBCDIC データをギリシャ語で処理した場合) は、上記と同じ例を次のようにしてコーディングすることができます。

```

      CBL CODEPAGE(00875)
* . . .
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

```

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。グループには、明示的または暗黙的に USAGE NATIONAL。

このタスクについて

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。あ GROUP-USAGE NATIONAL 条項は暗黙的に Group-1、そして USAGE NATIONAL は、以下の従属項目に暗示されています。Group-1。Month、DayOf、そして Year 国別の小数点項目であり、Amount 数値編集項目であり、USAGE NATIONAL。

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。（もし Alpha-Group-1 指定された USAGE NATIONAL グループレベルでは、USAGE NATIONAL それぞれの従属項目について暗示される。Group-1。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。国別グループ内のすべての基本項目は、明示的または暗黙的に次のように記述する必要があります。USAGE NATIONAL、国別グループ内のすべてのグループ項目は、明示的または暗黙的に次のように記述する必要があります。GROUP-USAGE NATIONAL。

関連概念

[141 ページの『国別グループ』](#)

関連タスク

[146 ページの『国別グループを基本項目として使用』](#)

[147 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 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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 06/23/2010 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 2010.

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 06232010 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

[33 ページの『グループ・データ項目への値の割り当て \(MOVE\)』](#)

[150 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』](#)

[147 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

MOVE ステートメント (*Enterprise COBOL for z/OS* 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。

このタスクについて

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 17. 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	コメント
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、国別グループ内の基本項目と同様に処理されます。USAGE NATIONAL 英数字グループ内。
EXEC SQL ステートメントのホスト変数	国別グループ項目をホスト変数として指定してください。	国別グループ項目は、実質的には、グループ項目に従属するホスト変数セットの省略表現です。
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。	国グループ内の基本項目は、国グループ内の基本項目と同様に初期化されます。USAGE NATIONAL 英数字グループ内。
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になります。
FROM のフレーズ XML GENERATE 声明	国別グループ項目を指定する FROM 規則に従ってグループとして処理することを意味するフレーズ XML GENERATE 声明。	国別グループ内の基本項目は、国別グループ内の基本項目と同様に処理されます。USAGE NATIONAL 英数字グループ内。

関連タスク

[31 ページの『構造の初期化 \(INITIALIZE\)』](#)

[79 ページの『テーブルの初期化 \(INITIALIZE\)』](#)

[32 ページの『基本データ項目への値の割り当て \(MOVE\)』](#)

[33 ページの『グループ・データ項目への値の割り当て \(MOVE\)』](#)

[129 ページの『データ項目の長さの検出』](#)

[727 ページの『XML 出力の生成』](#)

[561 ページの『SQL ステートメントでの国別グループ項目の使用』](#)

関連参照

修飾 (*Enterprise COBOL for z/OS* 言語解説書)

RENAMES 節 (*Enterprise COBOL for z/OS* 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 18. 英数字データ、DBCS データ、および国別データのエンコード方式およびサイズ			
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	EBCDIC	EBCDIC DBCS	UTF-16BE ¹
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. 英数字または DBCS データに適用可能な EBCDIC コード・ページを指定するには、CODEPAGE コンパイラ・オプションを使用します。 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す。 • COBOL 文字 A から Z、a から z、0 から 9、スペース、+ - * / = \$, ; . " () > < : ' ! • EBCDIC または ASCII コード・ページから変換されるすべての文字			

関連概念

[132 ページの『Unicode および言語文字のエンコード』](#)

国別 (UTF-16) データの比較

国別で比較できます (UTF-16) データ、つまり国別リテラルとデータ項目は、USAGE NATIONAL (国別クラスか数値クラスかを問わず)、明示的または暗黙的に他の種類のデータと関連付ける関係条件において。

このタスクについて

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

国別データ項目と他のデータ項目の比較について詳しくは、関連参照を参照してください。

関連タスク

[149 ページの『2 つのクラス国別オペランドの比較』](#)

[149 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』](#)

[150 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』](#)

[150 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』](#)

[150 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 (*Enterprise COBOL for z/OS* 言語解説書)

一般比較条件 (*Enterprise COBOL for z/OS* 言語解説書)

国別比較 (*Enterprise COBOL for z/OS* 言語解説書)

グループ比較 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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
- カテゴリー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。

同じ長さを持つ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。

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

[141 ページの『国別グループ』](#)

関連タスク

[145 ページの『国別グループの使用』](#)

関連参照

国別比較 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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 (*Enterprise COBOL for z/OS* 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。

このタスクについて

数値オペランドの代数値は、その値に関係なく比較できます。USAGE 。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

[141 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 (*Enterprise COBOL for z/OS* 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、UTF-8、DBCS、または数字編集) のいずれかの文字値と比較できます。

このタスクについて

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

[140 ページの『国別文字形象定数の使用』](#)

関連参照

国別比較 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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 (Enterprise COBOL for z/OS 言語解説書)

COBOL での UTF-8 データ (Unicode) の使用

UTF-8 は、1 つから 4 つまでの 8 ビット・バイトを使用して有効な各 Unicode コード・ポイントをエンコードする可変幅の Unicode エンコードです。UTF-8 には多くの望ましい特性があります。例えば、ASCII に対する後方互換性があること、UTF-16 よりコンパクトな Unicode データ表現が多いこと、エンディアンネスに依存しないことなどが挙げられます。UTF-8 は HTML および関連言語の推奨エンコードであり、ワールド・ワイド・ウェブで最もよく使用されているデータ・エンコードです。

Enterprise COBOL における UTF-8 のサポートについて詳しくは、関連参照を参照してください。

関連参照

151 ページの『UTF-8 データ項目の定義』

153 ページの『UTF-8 リテラルの使用』

154 ページの『COBOL ステートメントと UTF-8 データ項目』

155 ページの『UTF-8 データ項目に対する組み込み関数サポート』

156 ページの『Db2 に関する考慮事項』

UTF-8 データ項目の定義

Enterprise COBOL は、UTF-8 データ項目の定義、移動、および比較のためのネイティブ・サポートを提供しています。

Enterprise COBOL for z/OS では、異なる 3 つのタイプの UTF-8 データ項目がサポートされています。

• 固定文字長 UTF-8 データ項目

このタイプの UTF-8 データ項目が定義されるのは、PICTURE 節に 1 つ以上の「U」文字、または 1 つの「U」記号と、その後に続く反復因数が含まれていて、PICTURE 節の BYTE-LENGTH 句も、DYNAMIC LENGTH 節も指定されない場合です。

PICTURE 節に含まれる各「U」文字は 1 つの UTF-8 文字に相当します。これは、Enterprise COBOL において 1 つの Unicode コード・ポイントと同等のものとして扱われます。文字の UTF-8 エンコードの長さはさまざまであり、常に 1 バイトから 4 バイトまでです。

次のコード・フラグメントは、2 つの異なる固定文字長 UTF-8 データ項目定義を示しています。

```
01 u1 pic u(10).    *> fixed character-length UTF-8 data item holding 10 UTF-8 characters (40
bytes reserved)
01 u2 pic uuuu.     *> fixed character-length UTF-8 data item holding 4 UTF-8 characters (16
bytes reserved)
```

固定文字長 UTF-8 データ項目の場合、メモリー内のデータ項目用に予約されているバイトの数は $4 \times n$ です。n は項目の定義で指定されている文字の数です。UTF-8 エンコードの長さが変化する性質により、n 個の文字を長さ n の UTF-8 データ項目に移動した後でも、データを保持するために $4 \times n$ 個の予約バイトがすべて必要であるとは限らないことに注意してください。これはデータ内の各文字のサイズによって異なります。

移動中は常に、固定文字長 UTF-8 データ項目には、データ項目の最大バイト長まで UTF-8 ブランク (x'20') が埋め込まれます。切り捨てが固定文字長 UTF-8 データ項目で行われるときは文字境界で行われます。

固定文字長 UTF-8 データ項目が送り出し側として使用されるときは必ず、その項目のバイト長は実行時に既知の固定文字長に基づいて計算されるため、操作で使用する文字の数が、項目定義に示されている文字の数と同じになります。

• 固定バイト長 UTF-8 データ項目

このタイプの UTF-8 データ項目が定義されるのは、PICTURE 節に 1 つの「U」文字が含まれていて、PICTURE 節の BYTE-LENGTH 句が指定されている場合です。この句は、データ項目が、正確に n 個の有効な UTF-8 バイトで構成される固定バイト長 UTF-8 項目であることを示します。ただし、UTF-8 エンコードの長さが変化する性質により、データ項目に含まれる文字の数は常に可変であり、各文字のサイズに依存しますが、必ず $\lceil 4/n \rceil, n$ の範囲に収まらなければなりません。ここで、 $\lceil \cdot \rceil$ は、引数よりも大きい最小の整数を戻す数学関数です。

次のコード・フラグメントは、2 つの異なる固定バイト長 UTF-8 データ項目定義を示しています。

```
01 u1 pic u byte-length 10.  *> fixed byte-length UTF-8 data item
```

固定バイト長 UTF-8 データ項目に切り捨てが必要となる場合、切り捨ては文字境界で行われ、データ項目には常に UTF-8 スペース (x'20') が指定のバイト長 n まで埋め込まれます。

固定バイト長 UTF-8 データ項目が送り出し側として使用されるときは必ず、その項目のバイト長は常に n であるとみなされます。

固定バイト長 UTF-8 データ項目は、Db2[®] 製品、DFSORT 製品、および MQ 製品との互換性のために用意されています。これらの製品はいずれも固定バイト長 UTF-8 データ・ストリングに対するサポートを提供しています。例えば、Unicode 表の CHAR(n) 列に対応する Db2 ホスト変数として COBOL UTF-8 データ項目を使用するには、そのデータ項目に PICTURE 節の BYTE-LENGTH 句が含まれていなければなりません。同様に、DFSORT は固定バイト長を持つ UTF-8 ソート・キー部分のみをサポートするため、ソート・キーとマージ・キーに対しては COBOL の固定バイト長 UTF-8 データ項目を強くお勧めします。

• 動的長 UTF-8 データ項目

UTF-8 データ項目は、DYNAMIC LENGTH 節を使用して宣言できます。UTF-8 データ項目のバイト長は変化するため、この節は UTF-8 データ項目に無理なく適合します。この場合、データ項目における文字の数の制限はなく、バイト数が制限されるのは、DYNAMIC LENGTH 節の LIMIT 句が指定されているときに限られます。

DYNAMIC LENGTH 節で定義された UTF-8 データ項目において LIMIT 句が原因で切り捨てが必要になった場合、切り捨ては UTF-8 文字レベルで行われます。

このようなデータ項目に対する切り捨ては、LIMIT 句が原因で必要となる場合、文字境界で行われます。

動的長 UTF-8 データ項目が送り出し側として使用されるときは必ず、その項目のバイト長は常にその項目の現行実行時バイト長です。

例:

```
01 u1 pic u dynamic length limit 10.  *> dynamic-length UTF-8 data item
```

「U」pic 記号で宣言されたデータ項目には、次の規則が適用されます。

- UTF-8 データ項目は基本データ項目にすることができます。
- UTF-8 データ項目はファイル・レコードなどのグループで指定できる。ただし、データ項目が動的長データ項目の場合を除く。動的長データ項目はファイル・レコードでサポートされていないため。
- グループ項目は、GROUP-USAGE 節の UTF-8 句を使用することで UTF-8 項目にすることができます。

注：GROUP-USAGE UTF-8 節で定義されたグループに含めることができるのは、PICTURE 節の BYTE-LENGTH 句で定義された UTF-8 項目のみです。他のクラスのデータ項目は許可されません。

- 条件名 (レベル 88 項目) に関連付けられている条件変数は UTF-8 にすることができます。
- UTF-8 データ項目に対する VALUE 節は、英数字リテラル、国別リテラル、および UTF-8 リテラルを受け入れる。英数字の場合、リテラルは自動的に EBCDIC から UTF-8 に変換される。
- UTF-8 サポートに以下のタイプのデータ項目に対するサポートは含まれない。
 - UTF-8 編集、UTF-8 数字編集
 - UTF-8 10 進数
 - UTF-8 外部浮動小数点

注: USAGE UTF-8 節は「U」pic 記号で宣言されたデータ項目に対するデータ定義にのみ指定できます。つまり、数値項目は USAGE UTF-8 では定義できません。

次のコード・フラグメントは、有効/無効な UTF-8 データ項目定義を示しています。

有効な UTF-8 データ項目定義の例:

01 u1 pic u(10).	*> fixed character-length item, usage utf-8
implied	
01 u2 pic u(10) usage utf-8.	*> fixed character-length item, usage utf-8
specified explicitly	
77 u3 pic uuu usage utf-8.	*> fixed character-length item
77 u4 pic u dynamic length usage utf-8.	*> dynamic length item
01 u5 pic u(5) value u'abcde'.	*> fixed character-length item, VALUE clause
applied	
01 u6 pic u byte-length 5.	*> fixed byte-length

無効な UTF-8 データ項目定義の例:

01 u1 pic uuu,uuu,uuu usage utf-8.	*> utf-8 edited not supported
01 u2 pic 9(9) usage utf-8.	*> utf-8 decimal not supported
77 u3 pic 999,999 usage utf-8.	*> utf-8 numeric edited not supported
77 u4 pic 999e+99 usage utf-8.	*> utf-8 external float not supported
01 u5 pic u(10) byte-length 20.	*> cannot include repetition factor and byte-length phrase
01 u6 pic uuu byte-length 5.	*> cannot include repetition factor and byte-length phrase
01 u7 pic u byte-length 5 dynamic length.	*> only one of byte-length and dynamic length allowed
length phrase	

UTF-8 リテラルの使用

2 つのタイプの UTF-8 リテラルがサポートされています。

1. 基本 UTF-8 リテラル:

- `u'character-data'`
 - `character-data` は EBCDIC から UTF-8 に変換されます。
 - `character-data` には 2 バイト EBCDIC 文字を含めることができますが、その文字はシフトアウト文字とシフトイン文字で区切らなければなりません。
 - 基本 UTF-8 リテラルで表すことができる Unicode コード・ポイントの最大数は、各 UTF-8 文字のサイズによって異なる可能性があります。ただし、切り捨てが行われる前に、UTF-8 変換後の最大値として 160 バイトが許可されます。
 - 文字データ には、以下の Unicode エスケープ・シーケンスを入れることができます。
 - `¥uhhhh`: それぞれの `h` は、「0」から「9」まで、「a」から「f」まで、および「A」から「F」までの範囲にある 16 進数字を表します。この Unicode エスケープ・シーケンスは、基本多言語面 (BMP) からの Unicode コード・ポイント (つまり、U+0000 から U+FFFF までの範囲にある Unicode コード・ポイント) を表します。
 - `\U00hhhhhh`: それぞれの `h` は、「0」から「9」まで、「a」から「f」まで、および「A」から「F」までの範囲にある 16 進数字を表します。この Unicode エスケープ・シーケンスは、補助面からのコードポイント (すなわち、U+10000 から U+10FFFF までの範囲にある Unicode コード・ポイント (絵文字シンボルなど)) を含む任意の正当な Unicode コード・ポイントを表すことができます。

注:

- コード・ポイント U+D800 から U+DFFF は、UTF-16 が使用するサロゲート・ペアの高位半分と低位半分のために予約されます。UTF-8 では、これらの Unicode コード・ポイントの正当なエンコードはありません。したがって、UTF-8 リテラル内で `¥uD800` から `¥U0000DFFF` まで、および `¥U0000D800` から `¥U0000DFFF` までを Unicode エスケープ・シーケンスとして指定することはできません。
- UTF-8 リテラル内の `¥uhhhh` や `¥U00hhhhhh` という形式の文字のストリングが Unicode エスケープ・シーケンスとして解釈されないようにするには、エスケープ文字

「¥」が文字どおりに解釈されるようにそれ自体を「¥」でエスケープします。こうすることでシーケンス ¥¥u00E9 は Unicode エスケープ・シーケンスとして扱われなくなります。

- Unicode エスケープ・シーケンスは、基本 UTF-8 リテラルで指定される場合は必ず、コンパイラによって、そのエスケープ・シーケンスに対応する Unicode コード・ポイントの UTF-8 エンコードで置き換えられます。このことは、EBCDIC 文字のみを使用してリテラルにおいて汎用 Unicode コード・ポイントを表すために役立ちます。例えば、u'caf¥u00E9' はストリング「café」を表します。

2. 16 進 UTF-8 リテラル

• ux'hexadecimal-digits'

- hexadecimal-digits は、UTF-8 リテラル値としてそのまま使用するためにバイト・シーケンスに変換されます。
- 最小 2 桁から最大 320 桁までの 16 進数を使用できます。

注: 16 進数字で表されるバイトのシーケンスは、正規の UTF-8 バイト・シーケンスが含まれていることを確認するために検証されます。

COBOL ステートメントと UTF-8 データ項目

このタスクについて

以下の COBOL ステートメントは UTF-8 項目を直接サポートします。

ALLOCATE または FREE

LINKAGE SECTION で宣言された UTF-8 データ項目は、動的に割り振ったり解放したりできます。

EVALUATE および IF

条件内で UTF-8 データ項目を比較できます。UTF-8 データ項目同士を比較するには、バイナリー・バイト単位比較を使用します。

UTF-8 データ項目は、クラス英字、クラス英数字、クラス国別、またはクラス UTF-8 の項目とのみ比較できます。

EBCDIC から UTF-8 への変換、および UTF-16 から UTF-8 への変換は、比較時に必要に応じて自動的に行われます。

INITIALIZE

UTF-8 データ項目のカテゴリー・デフォルトは x'20' (UTF-8 スペース) です。

JSON 生成と JSON 解析

カテゴリのデータ項目 UTF-8 JSON GENERATE および JSON PARSE 操作の重要な部分として使用できます。

MERGE および SORT

カテゴリー UTF-8 のデータ項目は、ソート操作およびマージ操作の主要部分として使用できます。

COBOL での UTF-8 データの比較はすべて、バイナリー・バイト単位比較を使用して行われます。これにより、UTF-8 ストリング・セットの順序は、ストリング内の全コード・ポイントが基本多言語面 (BMP) からのものであれば、同じ Unicode コード・ポイントを表す同様の国別ストリング・セットと同じ順序になります。

MOVE

基本移動規則:

- カテゴリー UTF-8 送り出し側は、クラスおよびカテゴリーが国別の項目、またはクラスおよびカテゴリーが UTF-8 の項目にのみ移動できます。
- カテゴリー UTF-8 項目は、クラス英字、クラス英数字、クラス国別、またはクラス UTF-8 の項目のみを受け取ることができます。

注: これには、数字編集、英数字編集、国別編集、国別数字編集などの項目が含まれます。

埋め込みと切り捨ては、必要に応じて常に UTF-8 文字 (つまり Unicode コード・ポイント) レベルで行われます。

STRING と UNSTRING

データは、基本命令の MOVE 文の規則に従って、送信フィールドから受信フィールドへ転送されます。UTF-8-to-UTF-8 動きます。

以下の COBOL ステートメントは UTF-8 引数をサポートしていません。

- ACCEPT
- INSPECT
- XML GENERATE
- XML PARSE

ヒント: UTF-8 データ項目をまだネイティブにサポートしていない INSPECT のようなステートメントで UTF-8 データ項目を使用する必要がある場合、UTF-8 データ項目を最初にナショナルデータ項目に移動し、ステートメントでナショナル項目を代わりに使用することができます。

関連タスク

[162 ページの『UTF-16 \(国別\) データ・タイプを使用して UTF-8 データを処理』](#)

UTF-8 データ項目に対する組み込み関数サポート

UTF-8 データは引数のクラスおよびカテゴリーであり、組み込み関数の戻りの型でもあります。

このタスクについて

以下の組み込み関数は UTF-8 引数をサポートしています。

- BIT-OF
- BYTE-LENGTH
- CONTENT-OF¹
- DISPLAY-OF
- FORMATTED-CURRENT-DATE¹
- FORMATTED-DATE¹
- FORMATTED-DATETIME¹
- FORMATTED-TIME¹
- HEX-OF
- INTEGER-OF-FORMATTED-DATE
- LENGTH
- LOWER-CASE¹
- NATIONAL-OF
- SECONDS-FROM-FORMATTED-TIME
- TEST-FORMATTED-DATETIME
- TRIM¹
- ULENGTH
- UPOS
- UPPER-CASE¹
- USUBSTR¹
- USUPPLEMENTARY
- UVALID

- UWIDTH

1. この組み込み関数は、入力引数に UTF-8 データ項目が含まれている場合にカテゴリ UTF-8 の結果を返します。

関連参照

組み込み関数 (*Enterprise COBOL for z/OS 言語解説書*)

Db2 に関する考慮事項

Db2 ホスト変数のコード・ページは、CODEPAGE ファイル、SQLCCSID ファイル、および DSNHDECP ファイルなどさまざまな要因によって管理されます。ただし、ホスト変数が UTF-8 として宣言された場合、そのホスト変数はオーバーライドされ、Db2 コプロセッサは項目の CCSID が 1208 であると想定します。

Db2 ホスト変数として使用できるのは、PICTURE 節の BYTE-LENGTH 句を使用して宣言された UTF-8 データ項目のみです。この場合、ホスト変数が Db2 表の CHAR 列に対応しているのであれば、CHAR 列のバイト長と、PICTURE 節の BYTE-LENGTH 句に指定されたバイトの数を一致させて、予期せずにデータが切り捨てられたり埋め込まれたりしないようにします。

注: Db2 プリコンパイラーは、UTF-8 データ項目を含む COBOL プログラムとは使用できません。

UTF-8 文字形象定数の使用

UTF-8 文字を必要とするコンテキストでは、形象定数の ALL *utf-8-literal* を使用できます。ALL *utf-8-literal* は、UTF-8 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。

このタスクについて

UTF-8 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、UTF-8 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 UTF-8 文字 (UTF-8) 値を表します。

UTF-8 文字が必要となるコンテキストで形象定数 QUOTE を使用するとき QUOTE コンパイラー・オプションが有効であれば、値は UX'22' となります。APOST コンパイラー・オプションが有効であれば、値は UX'27' となります。

比喩定数を使う場合 HIGH-VALUE 要求される文脈において UTF-8 文字、その価値は UTF-8 キャラクター UX'F48FBFBF' Unicode コードポイントに対応 U+10FFFF ただし、HIGH-VALUE が固定バイト長の移動または比較操作で使用される場合は除く。UTF-8 長さが 4 バイトの倍数ではないデータ項目。その場合、HIGH-VALUE が最後の 3、2、または 1 バイトに移動されるか、または比較されると、UTF-8 データ項目の場合、HIGH-VALUE の値は UX'EFBFBF' (U+FFFF)、UX'DFBF' (U+07FF)、そして UX'7F' (U+007F)、それぞれ。

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 UX'00' です。

制約事項: HIGH-VALUES に設定された UTF-8 データ項目を、HIGH-VALUES に設定された英数字データ項目または国別データ項目と比較する場合、それらは等しいものとして評価されません。これは、HIGH-VALUE あるデータ型の値が対応するデータ型に変換されない HIGH-VALUE 異なるデータ型の値。

関連タスク

[159 ページの『UTF-8 \(Unicode\) 表現との間の変換』](#)

[160 ページの『UTF-8 データの比較』](#)

関連参照

形象定数 (*Enterprise COBOL for z/OS 言語解説書*)

DISPLAY-OF (*Enterprise COBOL for z/OS 言語解説書*)

Support for Unicode: Using Unicode Services

参照修飾子と UTF-8 データ項目

国別データ項目と同様、参照修飾子は UTF-8 データ項目に適用できます。UTF-8 参照修飾子の開始位置と長さは、バイト単位ではなく文字単位 (Unicode コード・ポイント) で指定されるので、UTF-8 サブストリングを生成するときに UTF-8 文字のバイト長の違いを気にする必要はありません。ただし、この可変という UTF-8 文字の性質により、特別な考慮事項がいくつかあります。

このタスクについて

- 参照変更の UTF-8 データ項目の開始バイト位置は、コンパイル時に決定できません。開始文字位置がリテラルとして指定されている場合でも、実行時に開始文字位置から決定する必要があります。

例 1

```
01 u1 pic u(10) value u'\u00e9cran'. *> écran
:
display u1(3:2)
```

この例では、u1(3:2) はデータ項目の 4 バイト目である文字位置 3 から始まります。これは、「é」が UTF-8 の 2 バイト・エンコードで、「c」と「r」の両方が UTF-8 の 1 バイト・エンコードであるからです。

例 2

```
01 u1 pic u(10) value u'ecran'.
:
display u1(3:2)
```

この例では、u1(3:2) はデータ項目の 3 バイト目である文字位置 3 から始まります。これは、「e」、「c」、および「r」はすべて UTF-8 の 1 バイト・エンコードであるからです。

- UTF-8 参照変更の実際のバイト長は、コンパイル時に決定できません。参照修飾子の文字長がリテラルとして指定されている場合でも、実行時に文字長から決定する必要があります。

例 3

```
01 u1 pic u(10) value u'caf\u00e9'. *> café
:
display u1(3:2)
```

この例では、u1(3:2) は UTF-8 の 2 文字ですが、長さは 3 バイトです。これは、「f」が UTF-8 の 1 バイト・エンコードで、「é」が UTF-8 の 2 バイト・エンコードであるからです。

例 4

```
01 u1 pic u(10) value u'cafe'.
:
display u1(3:2)
```

この例では、u1(3:2) は UTF-8 の 2 文字ですが、長さは 2 バイトです。これは、「f」と「e」が両方とも UTF-8 の 1 バイト・エンコードであるからです。

- MOVE ステートメントの受け取り側を参照変更の UTF-8 データ項目にすると、参照変更で指定したサブストリングを構成するバイト数が、移動前と移動後で変わる可能性があります。例えば、サブストリングの長さが 4 文字である場合に、移動前はそれらの 4 文字が 12 バイトで表されていたのに、移動後は 4 バイトだけになったり 16 バイトにまで増えていたりする可能性があります。これは、移動の UTF-8 の送り側によって決まります。このような状況が発生し、新しいサブストリングのバイト数が元のバイト数より少なくなると、サブストリングより後に位置する残りのデータが自動的に左にシフトされます。また、新しいサブストリングのバイト数が元のバイト数より多くなると、残りのバイトが右にシフトされます。後者の場合、基のデータ項目の残りの部分の文字が切り捨てられる可能性があります。ただし、切り捨てが発生するのは、基の UTF-8 データ項目が PICTURE 節の BYTE-LENGTH 句で定義されていた場合のみです。固定文字長の UTF-8 項目の場合は、切り捨てを行わずに右シフトに対応できるだけの十分なスペースが必ずあります。

例 5

```
01 u1 pic u(13) value '01\u00e9, 01\u00e9, 01\u00e9'. *> 01é, 01é, 01é
:
move 'abcdef' to u1(3:6)
```

この例の u1(3:6) は「é, 01é」を表します。これは 8 バイト長です。移動後には、この参照変更で表される 8 バイトのサブストリングが「abcdef」に置き換えられます。これは UTF-8 では 6 バイト長に過ぎません。つまり、基のデータ項目において参照変更部分の直後に位置する残りの文字「, 01é」が、移動の結果、2 バイト分、メモリー内で左にシフトされます。

例 6

```
01 u1 pic u(13) value '01\u00e9, 01\u00e9, 01\u00e9'. *> 01é, 01é, 01é
:
move '\u00e9\u00e9\u00e9\u00e9\u00e9\u00e9' to u1(3:6)
```

この例の u1(3:6) は「é, 01é」を表します。これは 8 バイト長です。移動後には、この参照変更で表される 8 バイトのサブストリングが「éééééé」に置き換えられます。これは UTF-8 では 12 バイト長です。つまり、基のデータ項目において参照変更部分の直後に位置する残りの文字「, 01é」が、移動の結果、4 バイト分、メモリー内で右にシフトされます。

関連参照

[118 ページの『参照修飾子』](#)

UTF-8 グループ

UTF-8 のグループは GROUP-USAGE UTF-8 節を使用して指定されます。これらのグループには固定バイト長を含めることができます UTF-8 データ項目（定義される項目）PIC U BYTE-LENGTH *n*）または動的長さデータ項目（PIC U DYNAMIC）。固定文字長 UTF-8 データ項目は許可されません。グループに固定バイト長のみが含まれる場合 UTF-8 データ項目の場合、ほとんどの場合、グループは基本カテゴリとして再定義されたかのように処理されます。UTF-8 アイテムの説明 PIC U BYTE-LENGTH *m*、どこ *m* グループ内のすべての項目の合計バイト長です。

ただし、UTF-8 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。このような操作（例えば、MOVE CORRESPONDING そして INITIALIZE）内の基本項目を認識または処理する UTF-8 グループ。

UTF-8 グループの現在の制限、つまり、固定バイト長の UTF-8 データ項目しか含められないという制限が許容されるユース・ケースでは、UTF-8 グループを使用すると、UTF-8 データの処理において、英数字グループ内の UTF-8 データの処理と比べて以下のような複数の利点があります。

- UTF-8 グループを、USAGE UTF-8 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 UTF-8 文字が埋め込まれます。これに対して、UTF-8 文字を含む英数字グループを、UTF-8 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。
- UTF-8 グループを、USAGE UTF-8 を持つもっと短いデータ項目に移動させると、UTF-8 グループは UTF-8 文字境界で切り捨てられます。これに対して、UTF-8 文字を含む英数字グループを、UTF-8 文字を含むもっと短い英数字グループに移動させると、UTF-8 文字の 2 バイト間で切り捨てが起こります。

USAGE UTF-8 グループ: グループ項目は、USAGE UTF-8 グループレベルでの便利な略語として USAGE グループ内の各基本データ項目の。そのようなグループはない国別グループではなく英数字グループであり、移動や比較などの多くの操作では、基本データ項目のように動作します。USAGE DISPLAY（ただし、データの編集や変換は行われません）。

関連タスク

[33 ページの『グループ・データ項目への値の割り当て \(MOVE\)』](#)

関連参照

GROUP-USAGE 節 (*Enterprise COBOL for z/OS* 言語解説書)

UTF-8 (Unicode) 表現との間の変換

暗黙的または明示的にデータ項目を UTF-8 表現に変換できます。

このタスクについて

MOVE ステートメントを使用すると、英字データまたは国別データを UTF-8 データに暗黙的に変換できます。暗黙的な変換は、他の COBOL 文でも行われます。IF 英数字データ項目と、USAGE UTF-8。

DISPLAY-OF 組み込み関数を使用すると、明示的に、UTF-16 データを UTF-8 データに、UTF-8 データを英数字データに変換できます。NATIONAL-OF 組み込み関数を使用して、明示的に、UTF-8 データを UTF-16 データに変換することもできます。DISPLAY-OF 組み込み関数を使用して UTF-16 を UTF-8 に変換する場合は、出力コード・ページ 1208 を指定する必要があります。

関連タスク

[159 ページの『英数字または国別から UTF-8 への変換 \(MOVE\)』](#)

[160 ページの『UTF-8 から国別への変換 \(NATIONAL-OF\)』](#)

[160 ページの『国別から UTF-8 への変換と UTF-8 から英数字への変換 \(DISPLAY-OF\)』](#)

[144 ページの『デフォルト・コード・ページのオーバーライド』](#)

[160 ページの『UTF-8 データの比較』](#)

関連参照

[385 ページの『CODEPAGE』](#)

[144 ページの『変換例外』](#)

DISPLAY-OF (*Enterprise COBOL for z/OS* 言語解説書)

NATIONAL-OF (*Enterprise COBOL for z/OS* 言語解説書)

英数字または国別から UTF-8 への変換 (MOVE)

MOVE ステートメントを使用して、データを UTF-8 表現に暗黙的に変換できます。

このタスクについて

次の種類のデータをカテゴリー UTF-8 に移動させることができ、そのようにしてデータを UTF-8 表現に変換できます。

- 英字
- 英数字
- 英数字編集
- 数字編集
- 国別
- 国別編集
- USAGE NATIONAL の数字編集

UTF-8 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 (*Enterprise COBOL for z/OS* 言語解説書) 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 UTF-8 データ項目 UTF8-Data に移動させます。

```
01  UTF8-Data  Pic U(2) Usage UTF-8.  
    Move "AB" to UTF8-Data
```

上記の MOVE ステートメントの実行後、UTF8-Data には、英数字「AB」の UTF-8 表現である UX'4142' が入ります。

USAGE UTF-8 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-8 スペース 文字 (UX'20') が使用されます。切り捨てが必要な場合、それは UTF-8 文字位置の境界で行われます。

関連タスク

[32 ページの『基本データ項目への値の割り当て \(MOVE\)』](#)

[33 ページの『グループ・データ項目への値の割り当て \(MOVE\)』](#)

関連参照

MOVE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

UTF-8 から国別への変換 (NATIONAL-OF)

NATIONAL-OF 組み込み関数を使用して、明示的に UTF-8 データ項目を国別データ項目に変換できます。

このタスクについて

関連参照

NATIONAL-OF (*Enterprise COBOL for z/OS* 言語解説書)

国別から UTF-8 への変換と UTF-8 から英数字への変換 (DISPLAY-OF)

国別データを UTF-8 に変換する場合や、UTF-8 データを、2 番目の引数として指定したコード・ページで表現された英数字 (USAGE DISPLAY) 文字ストリングに変換するには、DISPLAY-OF 組み込み関数を使用してください。

このタスクについて

国別データを UTF-8 に変換する場合は、DISPLAY-OF の 2 番目の引数である出力コード・ページが、コード・ページ 1208 になるように指定する必要があります。

UTF-8 を英数字に変換する場合に 2 番目の引数を省略すると、出力のコード・ページは、ソースのコンパイル時に CODEPAGE コンパイラー・オプションで有効にされたコード・ページになります。1 バイト文字セット (SBCS) 文字と DBCS 文字が組み合わせられた 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に SBCS 文字と DBCS 文字が混在していることがあります。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。

関連参照

DISPLAY-OF (*Enterprise COBOL for z/OS* 言語解説書)

UTF-8 データの比較

UTF-8 データ、すなわち USAGE UTF-8 を持つデータ項目および UTF-8 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。

このタスクについて

以下のステートメントで、UTF-8 データを使用する条件式をコード化できます。

- EVALUATE
- IF
- PERFORM
- SEARCH

UTF-8 データ項目と他のデータ項目の比較について詳しくは、関連参照を参照してください。

関連タスク

[161 ページの『2 つのクラス UTF-8 オペランドの比較』](#)

[161 ページの『UTF-8 と他の文字ストリング・オペランドとの比較』](#)

[161 ページの『UTF-8 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 (*Enterprise COBOL for z/OS* 言語解説書)

一般比較条件 (*Enterprise COBOL for z/OS* 言語解説書)

国別比較 (*Enterprise COBOL for z/OS* 言語解説書)

グループ比較 (*Enterprise COBOL for z/OS* 言語解説書)

2 つのクラス UTF-8 オペランドの比較

クラス UTF-8 の 2 つのオペランドの文字値を比較できます。

このタスクについて

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。

- UTF-8 グループ
- 基本カテゴリー UTF-8 データ項目

オペランドの 1 つは、代わりに UTF-8 リテラルまたは UTF-8 組み込み関数にすることができます。

同じ長さを持つ 2 つのクラス UTF-8 オペランドを比較するときに、各オペランドの対応する文字が等しい場合、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-8 スペース文字 (UX'20') が埋め込まれているものとして扱われます。

PROGRAM COLLATING SEQUENCE 節は、2 つのクラス UTF-8 オペランドの比較には影響しません。

関連参照

UTF-8 比較 (*Enterprise COBOL for z/OS* 言語解説書)

UTF-8 と他の文字ストリング・オペランドとの比較

UTF-8 リテラルまたはクラス UTF-8 データ項目の文字値を、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、または数字編集、国別、国別編集、USAGE NATIONAL の数字編集) のいずれかの文字値と比較できます。

このタスクについて

これらのオペランドは、基本 UTF-8 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-8) 表現へ変換され、2 つの UTF-8 文字オペランドの比較が進行します。

関連タスク

[140 ページの『国別文字形象定数の使用』](#)

関連参照

UTF-8 比較 (*Enterprise COBOL for z/OS* 言語解説書)

UTF-8 データ・オペランドと英数字グループ・オペランドの比較

UTF-8 リテラル、UTF-8 グループ項目、または USAGE UTF-8 の任意の基本データ項目を、英数字グループまたは国別グループと比較できます。

このタスクについて

どちらのオペランドも変換されません。UTF-8 オペランドは、UTF-8 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、英数字または国別グループ・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 U"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、UTF-8 リテラル U"AB123" の 5 バイトが、Group-XN の内容と 1 バイトずつ比較されます。項目を比較した結果、等しくないと評価されると、「NOT EQUAL」と表示されます。

関連参照

グループ比較 (*Enterprise COBOL for z/OS 言語解説書*)

UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理

USAGE UTF-8 データ項目を使用して UTF-8 データを処理するという推奨方法の代わりに、UTF-8 データを英数字データ項目に保管してから、国別データ項目で UTF-16 に変換することで処理することもできます。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。

このタスクについて

USAGE UTF-8 データ項目を使用して UTF-8 データを処理するという推奨方法の代わりに、UTF-8 データを英数字データ項目に保管してから、国別データ項目で UTF-16 に変換することで処理することもできます。

国別データは UTF-16 でエンコードされます。この場合、一般的に検出されるほぼすべての文字に 1 つのエンコード・ユニットが使用されます。このプロパティでは、国別データに対して参照変更などのストリング操作を使用できます。UTF-8 エンコードを保持したほうが利便性が良い場合は、Unicode 組み込み関数を使用してデータ処理を支援してください。詳細については、[163 ページの『組み込み関数を使用した UTF-8 エンコード・データの処理』](#)を参照してください。

ASCII または EBCDIC データを UTF-8 に変換するには、以下の手順に従ってください。

手順

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。

タスクの結果

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

[117 ページの『データ項目のサブストリングの参照』](#)

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

組み込み関数を使用した UTF-8 エンコード・データの処理

データのエンコードを UTF-8 に保持したほうが利便性が良い場合は、UTF-8 データのテストおよび処理を容易にするために Unicode 組み込み関数を使用してください。

このタスクについて

以下の組み込み関数を使用できます。

UVALID

UTF-8 文字データが適切な形式であることを検査します。

USUPPLEMENTARY

データを国別に変換し、各文字が単一 16 ビット・エンコード・ユニットで表現可能であることが重要な場合は、USUPPLEMENTARY 関数を使用して、有効な UTF-8 文字ストリングに Unicode 補足コード・ポイントが含まれているかどうかを判別します。つまり、U+FFFF より大きい Unicode スカラー値を持つコード・ポイントは、UTF-8 での 4 バイト表現を必要とします。

USUBSTR

UTF-8 文字ストリングのサブストリングを参照するための参照変更に代わる利便性の高い手段を提供します。参照変更では計算されたバイト位置とバイト・カウントを必要とするのに対し、USUBSTR は文字位置と長さの引数を必要とします。

補助関数により、有効な UTF-8 文字ストリングに関する追加情報を指定できます。

ULENGTH

ストリング内の Unicode コード・ポイントの総数を判別する

UPOS

n 番目の Unicode コード・ポイントのストリング内のバイト位置を判別する

UWIDTH

ストリング内の n 番目の Unicode コード・ポイントの幅 (バイト) を判別する

以下のコード・フラグメントは、UTF-8 の妥当性検査と、補助関数の使用を示しています。

```
checkUTF-8-validity.  
  Compute u = function UVALID(UTF-8-testStr)  
  If u not = 0  
  Display 'checkUTF-8-validity failure:'  
  Display '  The UTF-8 representation is not valid,'  
    'starting at byte ' u '.'  
  Compute v = function ULENGTH(UTF-8-testStr(1:u - 1))  
  Compute u = function UPOS(UTF-8-testStr v)  
  Compute w = function UWIDTH(UTF-8-testStr v)  
  Display '  The ' v 'th and last valid code point starts '  
    'at byte ' u ' for ' w ' bytes.'  
End-if.
```

次のストリングでは、x'F5' から x'FF' の範囲内の値をとりうるバイトがないため、x'F5' で始まるシーケンスは有効な UTF-8 ではありません。

```
x'6162D0B0E4BA8CF5646364'
```

このストリングの checkUTF-8-validity からの出力は以下のとおりです。

```
checkUTF-8-validity failure:  
  The UTF-8 representation is not valid, starting at byte 08.  
  The 04th and last valid code point starts at byte 05 for 03 bytes.
```

以下のコード・フラグメントは、UTF-8 での 4 バイト表現を必要とする、Unicode 補足コード・ポイントの存在検査を示しています。

```
checkUTF-8-sup.  
  Compute u = function USUPPLEMENTARY(UTF-8-testStr)
```

```

If u not = 0
  Display ' checkUTF-8-sup hit:'
  Compute v = function ULENGTH(UTF-8-testStr(1:u - 1))
  Compute w = function UWIDTH(UTF-8-testStr v + 1)
  Display ' The ' v 'th code point of the string'
  Display ' , starting at byte ' u ', '
  Display ' is a Unicode supplementary code point, '
  Display ' width ' w ' bytes.'
End-if.

```

次のストリングでは、シーケンス x'F0908C82' は補足文字です (範囲 x'F0' から x'F4' のバイトで始まる任意の有効な UTF-8 シーケンス)。

```
x'6162D0B0E4BA8CF0908C826364'
```

このストリングの checkUTF-8-sup からの出力は以下のとおりです。

```

checkUTF-8-sup hit:
  The 04th code point of the string, starting at byte 08,
  is a Unicode supplementary code point, width 04 bytes.

```

関連参照

[385 ページの『CODEPAGE』](#)

例: UTF-8 の名前からイニシャルを派生させる

下のプログラムでは、Unicode 関数を使用して、チェコ語の名前のテーブルから作曲家のイニシャルを派生させます。この例は、Unicode 組み込み関数を解説することを目的としているため、必ずしもこの作業を行う最も効率的な方法というわけではありません。このプログラムは、作曲家名を UTF-8 で処理しますが、プログラムのソースおよび出力を意味のある表示にするために、データを EBCDIC で開始および終了します。コンパイラー・オプション CODEPAGE(1153) により、Unicode への、および Unicode からの変換時に、名前が正しく解釈されます。

initials プログラム

```
Process codepage(1153)
*-----*
* For a table of Czech composer names represented in UTF-8,      *
* determine and print out the initials of each name.              *
*-----*
Identification division.
  Program-id. initials.
Data division.
  Working-storage section.
    1 utilityVariables.
      2 UTF-8-space pic x value x'20'.
      2 UTF-8-hyphen pic x value x'2D'.
      2 UTF-8-ch pic xxx.
      2 i comp pic 9.
      2 j comp pic 99.
      2 hex pic x(160).
    1 EBCDICNameData.
      2 pic x(40) value 'Antonín Leopold Dvořák'.
      2 pic x(40) value 'Leoš Janáček'.
      2 pic x(40) value 'Rafael Jeroným Kubelík'.
      2 pic x(40) value 'Pavel Křížkovský'.
      2 pic x(40) value 'Jan Václav Hugo Voříšek'.
    1 redefines EBCDICNameData.
      2 EBCDICName pic x(40) occurs 5 times.
    1 UTF-8-nameData.
      2 composer pic x(40) occurs 5 times.
    1 composerInitials.
      2 occurs 5.
      3 cInitSize comp pic 99.
      3 cInit pic x(8).
    1 state pic 9.
      88 seekingInitial value 0.  *> Skip space and hyphen
      88 seekingSeparator value 1. *> Skip all but space and hyphen
```


initials プログラム、続き

```
Procedure division.
  main.
    Display 'Compute composer initials...'
    Initialize composerInitials
    Perform test before varying i from 1 by 1 until i > 5
    * Start by translating each composer name from EBCDIC to UTF-8.
      Move function display-of
        (function national-of(EBCDICname(i)) 1208)
      to composer(i)
    * Test each character of the name; skip leading spaces, etc.
      Set seekingInitial to true
      Move 1 to cInitSize(i)
      Perform varying j from 1 by 1
        until j > function ULENGTH(composer(i))
        Move function USUBSTR(composer(i) j 1) to UTF-8-ch
    * Initial found. Save in buffer, then skip to next space/hyphen.
      If seekingInitial and
        UTF-8-ch not = UTF-8-Space and UTF-8-Hyphen
        String function USUBSTR(composer(i) j 1)
          delimited by size
          into cInit(i) with pointer cInitSize(i)
        Set seekingSeparator to true
      End-if
    * Space/hyphen found; skip spaces or hyphens to next initial.
      If seekingSeparator and
        (UTF-8-ch = UTF-8-Space or UTF-8-Hyphen)
        Set seekingInitial to true
      End-if
    End-perform
    * Adjust string pointer to number of initials found.
      Subtract 1 from cInitSize(i)
    End-perform
    * Print out the UTF-8 initials, translated to EBCDIC, and
    * also in hexadecimal, using program ToHex (listed later).
      Perform test before varying i from 1 by 1 until i > 5
      Call 'toHex' using hex cInit(i) value cInitSize(i)
      Display '  #' i ': ' function display-of(
        function national-of(cInit(i) (1:cInitSize(i)) 1208))
        ' (x''' hex(1:2 * cInitSize(i)) ''')'
      End-perform.
    Goback.
  End program initials.
```

initials プログラムからの出力

```
Compute composer initials...
#1: ALD (x'414C44')
#2: LJ (x'4C4A')
#3: RJK (x'524A4B')
#4: PK (x'504B')
#5: JVHV (x'4A564856')
```

toHex プログラム

```
Identification division.
  Program-id. toHex.
Data division.
  Working-storage section.
    1 hexv.
      2 pic x(32) value '000102030405060708090A0B0C0D0E0F'.
      2 pic x(32) value '101112131415161718191A1B1C1D1E1F'.
      2 pic x(32) value '202122232425262728292A2B2C2D2E2F'.
      2 pic x(32) value '303132333435363738393A3B3C3D3E3F'.
      2 pic x(32) value '404142434445464748494A4B4C4D4E4F'.
      2 pic x(32) value '505152535455565758595A5B5C5D5E5F'.
      2 pic x(32) value '606162636465666768696A6B6C6D6E6F'.
      2 pic x(32) value '707172737475767778797A7B7C7D7E7F'.
```



```

2 pic x(32) value '808182838485868788898A8B8C8D8E8F'.
2 pic x(32) value '909192939495969798999A9B9C9D9E9F'.
2 pic x(32) value 'A0A1A2A3A4A5A6A7A8A9AAABACADAFAF'.
2 pic x(32) value 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'.
2 pic x(32) value 'C0C1C2C3C4C5C6C7C8C9CACBCCDCCECF'.
2 pic x(32) value 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'.
2 pic x(32) value 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'.
2 pic x(32) value 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'.
1 redefines hexv.
2 hex pic xx occurs 256 times.
Local-storage section.
1 i pic 9(4) binary.
1 j pic 9(4) binary value 0.
1 jx redefines j.
2 pic x.
2 jxd pic x.
Linkage section.
1 ostr.
2 ostrv pic xx occurs 1024 times.
1 istr.
2 istrv pic x occurs 1024 times.
1 len pic 9(9) binary.
Procedure division using ostr istr value len.
  If len > 1024
    Display '>> Error: length ' len ' greater than toHex '
      'supported maximum of 1024.'
    Stop run
  End-if
  Perform with test before varying i from 1 by 1 until i > len
    Move 0 to j
    Move istrv(i) to jxd
    Add 1 to j
    Move hex(j) to ostrv(i)
  End-perform
  Goback
.
End program toHex.

```

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。

このタスクについて

GB 18030 文字は、UTF-16 またはコード・ページ CCSID 1392 でエンコードできます。コード・ページ 1392 は、1 文字に 1、2、または 4 バイトを使用する ASCII マルチバイト・コード・ページです。GB 18030 文字のサブセットは、中国語 ASCII コード・ページ (CCSID 1386) または中国語 EBCDIC コード・ページ (CCSID 1388) でエンコードできます。

Enterprise COBOL は GB 18030 を明示的にはサポートしていませんが、いくつかの方法で GB 18030 文字の処理をサポートします。以下を実行できます。

- DBCS データ項目を使用して、CCSID 1388 で表される GB 18030 文字を処理します。
- 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 (CCSID 1388 または 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 データを処理する必要がある場合は、まず入力データを国別データ項目内で 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、コード・ページ 1388 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-EBCDIC pic X(16) value "オリンピック运动会".
01 Chinese-GB18030-String pic X(16).
01 UnicodeString pic N(14).
* * *
* Move function National-of(Chinese-EBCDIC, 1388) to UnicodeString
* Process data in Unicode
  Move function Display-of(UnicodeString, 1388) to Chinese-GB18030-String

```

関連タスク

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

[168 ページの『DBCS サポート用のコーディング』](#)

関連参照

[148 ページの『文字データの保管』](#)

DBCS サポート用のコーディング

IBM エンタープライズ COBOL z/OS 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。

このタスクについて

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 (DBCS 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
- DBCS コンパイラー・オプション

関連タスク

[168 ページの『DBCS データの定義』](#)

[169 ページの『DBCS リテラルの使用』](#)

[169 ページの『有効な DBCS 文字に関するテスト』](#)

[170 ページの『DBCS データを含む英数字データ項目の処理』](#)

[867 ページの『2 バイト文字セット \(DBCS\) データの変換』](#)

関連参照

[392 ページの『DBCS』](#)

DBCS データの定義

使用 PICTURE そして USAGE 条項に定義する DBCS データ項目。DBCS データ項目は PICTURE シンボル G、G そして B、または N。各 DBCS 文字位置の長さは 2 バイトです。

このタスクについて

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。使用する場合は PICTURE シンボル N ただし、USAGE 句、USAGE DISPLAY-1 または USAGE NATIONAL の設定に応じて暗示される NSYMBOL コンパイラオプション。

DBCS 項目の定義で 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。

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。

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。

このタスクについて

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* '

引用符 (") またはアポストロフィ (') を DBCS リテラルの区切り文字として使用する。の設定 APOST または QUOTE コンパイラオプション。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。

シフトアウト (SO) 制御文字 X'0E' 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'0F' 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、英数字リテラルに含まれる DBCS 文字のストリングは、SO および SI 文字で区切る必要があり、SO および SI 文字がシフト・コードとして認識されるためには DBCS コンパイラー・オプションが有効になっている必要があります。

DBCS 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。

DBCS 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、そのような仕方で処理されません。

関連タスク

[26 ページの『形象定数の使用』](#)

関連参照

[379 ページの『APOST/QUOTE』](#)

[392 ページの『DBCS』](#)

[421 ページの『NSYMBOL』](#)

DBCS リテラル (*Enterprise COBOL for z/OS 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。

このタスクについて

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、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、文字を検査することで行われます。

関連タスク

[103 ページの『条件式のコーディング』](#)

関連参照

クラス条件 (*Enterprise COBOL for z/OS 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。

このタスクについて

すなわち、以下のステップを実行してください。

手順

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。

タスクの結果

関連タスク

[111 ページの『データ項目の結合 \(STRING\)』](#)

[113 ページの『データ項目の分割 \(UNSTRING\)』](#)

[117 ページの『データ項目のサブストリングの参照』](#)

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

第 8 章 ファイルの処理

このタスクについて

データの処理は、どのプログラムにも不可欠な部分です。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、結果を示します。

情報のソースおよび結果のターゲットとしては、以下の 1 つ以上の項目を使用できます。

- 別のプログラム
- 階層型データベースまたはリレーショナル・データベース
- サブシステム・ソフトウェアからのメッセージ
- 直接アクセス記憶装置
- 磁気テープ
- プリンター
- 端末
- カード読取装置または穿孔装置

外部装置上に存在する情報は、物理レコードまたはブロックに格納されている場合があります。レコードまたはブロックは、入力または出力操作時にシステムによって 1 つの単位として処理される情報の集まりです。

COBOL プログラムは、物理レコードを直接処理しません。論理レコードを処理します。論理レコードは、1 つの完全な物理レコードであることもあるし、1 つの物理レコードの一部であることもあるし、1 つ以上の物理レコードの一部または全部を含むこともあります。COBOL プログラムは、論理レコードを、それらが定義されたとおりに正確に処理します。

COBOL では、論理レコードの集合をファイル (プログラムで処理できる情報の列) と言います。

関連概念

[171 ページの『ファイル編成および入出力装置』](#)

関連タスク

[173 ページの『ファイル編成およびアクセス・モードの選択』](#)

[175 ページの『ファイルの割り振り』](#)

[176 ページの『入出力エラーの検査』](#)

ファイル編成および入出力装置

入出力装置によって異なりますが、ファイル編成には、順次、行順次、索引付き、または相対があります。プログラムを設計するときに、使用する装置およびファイル・タイプを決定してください。

以下のファイル編成を選択することができます。

順次ファイル編成

レコードの配置は、ファイル作成時のレコードの入力順によって決まります。それぞれのレコード (最初のレコードは除く) には固有の先行レコードがあり、それぞれのレコード (最後のレコードは除く) には固有の後続レコードがあります。いったん確立されると、これらの関係は変わりません。

順次ファイルについて認められるアクセス (レコード伝送) モードは順次のみです。

行順次ファイルの編成

行順次ファイルは z/OS UNIX ファイルシステム内にある順次ファイルで、データとして文字のみを含んでいます。各レコードは改行文字で終了します。

行順次ファイルについて認められるアクセス (レコード伝送) モードは順次のみです。

索引付きファイル編成

ファイル内のそれぞれのレコードには特殊フィールドが含まれ、そのフィールドの内容がレコード・キーを形成します。このキーの位置は各レコードで同じです。ファイルの論理配置は、ファイルの索引コンポーネントがレコード・キーによって順序付けて確立します。ファイル内のレコードの実際の物理配置は、COBOL プログラムにとっては重要ではありません。

索引付きファイルは、レコード・キーに加えて、代替索引を使用することもできます。これらのキーにより、レコードに関して別の論理順序付けを使用して、ファイルにアクセスできます。

索引付きファイルについて認められるアクセス (レコード伝送) モードは、順次、ランダム、または動的です。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。索引付きファイルの読み取り/書き込みをランダムに行う場合、その順序はプログラマー指定の方式に従います。索引付きファイルの読み取り/書き込みを動的に行う場合、その順序は使用されている入出力ステートメントによって決定され、順次またはランダム、あるいはその両方になります。

相対ファイル編成

ファイルのレコードは、ファイルの始まりからの相対位置によって識別されます。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。

相対ファイルについて認められるアクセス (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。

IBM エンタープライズ COBOL z/OS では、レコードの保管および入出力装置からの取り出しに関するオペレーティング・システムへの要求は、QSAM と VSAM の 2 つのアクセス方式、および z/OS UNIX ファイルシステムによって処理されます。

データの保管にどの装置タイプを使用するかによって、使用できるファイル編成の選択に影響する可能性があります。ファイル編成オプションで最も柔軟に選択できるのは、直接アクセス記憶装置の場合です。順次専用の装置の場合、編成オプションは制限されますが、有用な他の特性 (テープの可搬性など) があります。

順次専用装置

端末装置、プリンター、カード読取装置、および穿孔装置は、一度に 1 行を処理するので、ユニット・レコード装置と呼ばれます。したがって、ユニット・レコード装置との間で読み取りまたは書き込みを行うときは、プログラムで一度にレコードを 1 つずつ順次に処理しなければなりません。

テープでは、レコードは順次に並べられるので、プログラムでそれらを順次に処理しなければなりません。テープ・ファイルを処理するときは、QSAM 物理順次ファイルを使用してください。テープのレコードは固定長でも可変長でも構いません。

直接アクセス・ストレージ

直接アクセス・ストレージ・デバイスには多くのレコードが格納できます。これらの装置に保管されるファイルのレコード配置によって、プログラムがデータを処理できる方法が決まります。直接アクセス装置を使用すると、以下の複数のファイル編成タイプが使用できるため、プログラムにおける柔軟性が高まります。

- 順次 (VSAM または QSAM)
- 行順次 (z/OS ユニックス)
- 索引付き (VSAM)
- 相対 (VSAM)

関連タスク

[175 ページの『ファイルの割り振り』](#)

[177 ページの『第 9 章 QSAM ファイルの処理』](#)

[201 ページの『第 10 章 VSAM ファイルの処理』](#)

[237 ページの『第 12 章 行順次ファイルの処理』](#)

[173 ページの『ファイル編成およびアクセス・モードの選択』](#)

ファイル編成およびアクセス・モードの選択

アプリケーションで使用するファイル編成およびアクセス・モードを決定する際に使用できる指針があります。

このタスクについて

ファイル編成の選択時に以下の指針を考慮してください。

- アプリケーションがレコード (固定長でも可変長でも) に順次にアクセスのみを行い、既存のレコード間にレコードを挿入しない場合には、QSAM または VSAM 順次ファイルが最も単純なタイプです。
- 印刷可能文字および特定の制御文字のみを含むレコードに順次にアクセスする z/OS UNIX ファイルシステム アプリケーションを開発している場合には、行順次ファイルが最も適しています。
- アプリケーションが (レコードが固定長でも可変長でも) 順次アクセスとランダム・アクセスの両方を必要とする場合には、VSAM 索引付きファイルが最も柔軟性のあるタイプです。
- アプリケーションがレコードをランダムに挿入および削除する場合には、相対ファイルが適しています。

アクセス・モードの選択時には、以下の指針を考慮してください。

- ファイルのかなりの部分がアプリケーションの中で参照または更新される場合には、順次アクセスの方が、ランダムまたは動的アクセスより速くなります。
- アプリケーションの実行ごとに処理されるレコードの割合が小さい場合は、ランダム・アクセスまたは動的アクセスを使用します。

表 19. COBOL ファイルのファイル編成、アクセス・モード、レコード・フォーマットの要約					
ファイル編成	順次アクセス	ランダム・アクセス	動的アクセス	固定長	可変長
QSAM (物理順次)	X			X	X
行順次	X			X ¹	X
VSAM 順次 (ESDS)	X			X	X
VSAM 索引付き (KSDS)	X	X	X	X	X
VSAM 相対 (RRDS)	X	X	X	X	X
1. データ自体は可変長形式ですが、COBOL 固定長レコードに読み込んだり、固定長レコードから書き出すことができます。					

関連参照

[173 ページの『入出力コーディングの形式』](#)

[238 ページの『行順次ファイルの制御文字』](#)

入出力コーディングの形式

入出力コーディングの一般形式を、以下の例に示します。ユーザー提供情報についての説明は、コードの後に示しています。

```
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 PICTURE 99.
. . .
PROCEDURE DIVISION.
   OPEN iomode filename                      (9)
   READ filename
   WRITE recordname
   CLOSE filename
   STOP RUN.

```

以下は、上のコードでのユーザー提供情報の説明です。

(1) *filename*

任意の正規 COBOL 名。SELECT 節と FD 項目、および READ、OPEN、CLOSE の各ステートメントでは、同じファイル名を使用しなければなりません。さらに、START または DELETE ステートメントを使用する場合にも、ファイル名が必要になります。この名前は、必ずしも、システムに認識されているデータ・セット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。

(2) *assignment-name*

選択した任意の名前 (COBOL およびシステムの命名規則に従うもの)。名前は、ユーザー定義語である場合には 1 から 30 文字の長さにすることができ、リテラルである場合には 1 から 160 文字の長さにすることができます。コーディングする名前の一部割り当て名で DD 声明では、ALLOCATE コマンド (TSO)、または環境変数として (例えば、export 指示) (z/OS ユニックス)。

(3) *org*

組織は SEQUENTIAL、LINE SEQUENTIAL、INDEXED、または RELATIVE。この節は、QSAM ファイルについては任意指定です。

(4) *access*

アクセス・モードは、SEQUENTIAL、RANDOM、または DYNAMIC にすることができます。順次ファイル処理の場合 (行順次ファイルを含む) は、この節を省略することができます。

(5) *file-status*

COBOL ファイル状況キー。ファイルステータスキーは、2 文字の英数字カテゴリまたは国別カテゴリ項目、あるいは 2 桁のゾーン 10 進数 (USAGE DISPLAY) または国別小数点 (USAGE NATIONAL) アイテム。

(6) *recordname*

WRITE または REWRITE ステートメントで使用されるレコードの名前。

(7) *fieldlength*

フィールドの論理長。

(8) *type*

ファイルのレコード形式。レコード記入項目をレベル 01 記述以上に分ける場合は、各エレメントを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*

INPUT または OUTPUT モード。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のみを行う場合は、OUTPUT または EXTEND 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 (ただし、LINE SEQUENTIAL 編成の場合を除きます)。

関連タスク

[177 ページの『第 9 章 QSAM ファイルの処理』](#)

[201 ページの『第 10 章 VSAM ファイルの処理』](#)

[237 ページの『第 12 章 行順次ファイルの処理』](#)

ファイルの割り振り

z/OS または z/OS ユニックス アプリケーション内のどのタイプのファイル (順次、行順次、索引付き、または相対) についても、DD 名または環境変数名のいずれかを使用して外部名を定義することができます。外部名とは、ASSIGN 節の *assignment-name* 内の名前です。

このタスクについて

ファイルが z/OS UNIX ファイルシステム 内にある場合は、DD 定義または環境変数のいずれかを使用して、PATH キーワードでパス名を指定することによって、ファイルを定義することができます。

環境変数名は大文字でなければなりません。その値として使用できる属性は、定義中のファイルの編成によって異なります。

外部名は 2 つの方法のいずれかで定義できるので、COBOL 実行時には、以下のステップを使用してファイルの定義が調べられます。

手順

1. DD 名が明示的に割り振られていれば、それが使用されます。定義は、JCL の DD ステートメント、TSO/E からの ALLOCATE コマンド、またはユーザー開始の動的割り振りから取ることができます。
2. DD 名が明示的に割り振られていないが、同じ名前の環境変数が設定されている場合は、その環境変数の値が使用されます。

ファイルは、環境変数で指定された属性を使用して動的に割り振られます。最小限、PATH() または DSN() オプションのいずれかを指定しなければなりません。オプションおよび属性はすべて大文字でなければなりません (ただし、PATH オプションの *path-name* サブオプションは例外で、これには大/小文字の区別があります)。DSN() オプションに一時データ・セット名を指定することはできません。

次のいずれかの場合には、ファイル状況コード 98 が出されます。

- 環境変数の内容 (ヌルまたはすべてブランクの値を含む) が無効である。
- ファイルの動的割り振りが失敗する。
- ファイルの動的割り振り解除が失敗する。

COBOL 実行時には、各 OPEN ステートメントで環境変数の内容が検査されます。同じ外部名を持つファイルが前の OPEN ステートメントによって動的に割り振られており、その OPEN 以後に環境変数の内容が変更された場合は、実行時に、前の割り振りが動的に割り振り解除され、環境変数に現在設定されているオプションを使用してファイルが再度割り振られます。環境変数の内容が変更されていない場合、実行時には現行の割り振りが使用されます。

3. DD 名も環境変数も定義されていない場合、次のようになります。
 - a) 割り振りが QSAM ファイルに対するものであり、CBLQDA ランタイム・オプションが有効であれば、適格ファイルに対して CBLQDA 動的割り振り処理が行われます。この種の「暗黙の」動的割り振りは、実行単位が存続する間持続され、再割り振りすることはできません。
 - b) そうでない場合、割り振りは失敗します。

タスクの結果

COBOL 実行時には、暗黙の CBLQDA 割り振りを除いて、実行単位の終了時にすべての動的割り振りが割り振り解除されます。

関連タスク

- [580 ページの『環境変数の設定およびアクセス』](#)
- [191 ページの『QSAM ファイルの定義および割り振り』](#)
- [188 ページの『QSAM ファイルの動的作成』](#)
- [222 ページの『VSAM ファイルの割り振り』](#)

入出力エラーの検査

それぞれの入力または出力ステートメントが実行された後、操作が成功したか失敗したかを示す値にファイル状況キーが更新されます。

このタスクについて

FILE STATUS 節を使用して、それぞれの入力または出力ステートメントの後にファイル状況キーを検査し、ゼロ以外のファイル状況コードが戻されている場合にはエラー処理プロシージャを呼び出してください。VSAM ファイルでは、FILE STATUS 追加の VSAM ステータス コード情報を取得するための句。

入出力操作でのエラーを処理する別の方法として、ERROR (EXCEPTION と同義) 宣言をコーディングする方法があります。

関連タスク

[267 ページの『入出力操作でのエラーの処理』](#)

[270 ページの『ERROR 宣言のコーディング』](#)

[271 ページの『ファイル状況キーの使用』](#)

第 9 章 QSAM ファイルの処理

待機順次アクセス方式 (QSAM) ファイルは、キーなしのファイルであり、レコードは入力順に次々に配置されます。

このタスクについて

プログラムでは、これらのファイルを順次にのみ処理することができ、レコードを、それらがファイルに入っているのと同じ順序で検索します (READ ステートメントを使用して)。各レコードは先行レコードの後に置かれます。プログラムで QSAM ファイルを処理するには、次のような COBOL 言語ステートメントを使用します。

- ENVIRONMENT DIVISION および DATA DIVISION 内で QSAM ファイルを識別および記述する
- これらのファイル内のレコードを PROCEDURE DIVISION。

レコードの作成後は、ファイル内でレコードの長さや位置を変えることはできず、また削除することもできません。しかし、直接アクセス・ストレージ・デバイス上の QSAM ファイルは (REWRITE を使用して) 更新することができます (ただし z/OS UNIX ファイルシステムでは不可)。

QSAM ファイルは、テープ、直接アクセス・ストレージ・デバイス (DASD)、ユニット・レコード装置、および端末装置上に置くことができます。QSAM 処理は、テーブルおよび中間記憶域に最適です。

QSAM を使用して、z/OS UNIX ファイルシステム内のバイト・ストリーム・ファイルにアクセスすることもできます。これらのファイルは、レコード構造を持たない 2 進数のバイト単位の順次ファイルです。COBOL プログラムでコーディングしたレコード定義と、読み取りおよび書き込みに使用する変数の長さによって、転送されるデータの量が決まります。

関連概念

z/OS DFSMS: Using Data Sets (アクセス方式)

関連タスク

[177 ページの『COBOL での QSAM ファイルおよびレコードの定義』](#)

[186 ページの『QSAM ファイルの入出力ステートメントのコーディング』](#)

[190 ページの『QSAM ファイルのエラーの処理』](#)

[191 ページの『QSAM ファイルの操作』](#)

[198 ページの『QSAM を使用する z/OS ユニックスファイルへのアクセス』](#)

[198 ページの『テープ上の QSAM ASCII ファイルの処理』](#)

COBOL での QSAM ファイルおよびレコードの定義

COBOL プログラムのファイルを QSAM ファイルとして定義して、ファイルを外部ファイル名と関連付けるには、FILE-CONTROL 記入項目を使用します。

このタスクについて

外部ファイル名 (DD 名または環境変数名) とは、ファイルがオペレーティング・システムに認識されるときに使用される名前です。次の例では、COMMUTER-FILE-MST は、プログラムがファイルに使用する名前であり、COMMUTR は外部名です。

```
FILE-CONTROL.  
  SELECT COMMUTER-FILE-MST  
  ASSIGN TO S-COMMUTR  
  ORGANIZATION IS SEQUENTIAL  
  ACCESS MODE IS SEQUENTIAL.
```

ASSIGN 節 *name* には、ファイルが QSAM ファイルである文書の外部名の前に S- を含めることができます。両方とも ORGANIZATION そして ACCESS MODE 句はオプションです。

関連タスク

[178 ページの『レコード形式の指定』](#)

[184 ページの『ブロック・サイズの設定』](#)

レコード形式の指定

DATA DIVISION の FD 項目では、レコード・フォーマットおよびレコードをブロック化するかどうかの標識をコーディングしてください。関連するレコード記述項目では、*record-name* およびレコード長を指定します。

このタスクについて

RECORDING MODE 節に、レコード・フォーマット F、V、S、または U をコーディングできます。COBOL は、RECORD 節から、またはファイルの FD 記入項目に関連付けられたレコード記述から、レコード形式を判別します。レコードをブロックしたい場合は、BLOCK CONTAINS 条項の FD エントリ。

次の例は、固定長レコードを持つファイルの場合に FD 記入項目がどのようなになるかを示しています。

```
FILE SECTION.  
FD  COMMUTER-FILE-MST  
   RECORDING MODE IS F  
   BLOCK CONTAINS 0 RECORDS  
   RECORD CONTAINS 80 CHARACTERS.  
01  COMMUTER-RECORD-MST.  
    05  COMMUTER-NUMBER          PIC  X(16).  
    05  COMMUTER-DESCRIPTION     PIC  X(64).
```

S の記録モードは、z/OS UNIX ファイルシステム 内のファイルではサポートされません。上記の例はこのようなファイルに適しています。

関連概念

[178 ページの『論理レコード』](#)

関連タスク

[179 ページの『固定長フォーマットの要求』](#)

[180 ページの『可変長フォーマットの要求』](#)

[181 ページの『スパン形式の要求』](#)

[183 ページの『不定形式の要求』](#)

[177 ページの『COBOL での QSAM ファイルおよびレコードの定義』](#)

関連参照

[13 ページの『FILE SECTION 記入項目』](#)

論理レコード

COBOL では、論理レコード という用語を、z/OS QSAM とは多少異なる仕方で使用します。

形式 V および形式 S ファイルの場合、QSAM 論理レコードではレコードのユーザー・データ部分の前に、COBOL 論理レコードの定義には含まれていない 4 バイトの接頭部が入ります。

形式 F および形式 U のファイル、および z/OS UNIX ファイルシステムのバイト・ストリーム・ファイルの場合、QSAM 論理レコードと COBOL 論理レコードの定義は同じです。

本書では、QSAM 論理レコード は QSAM 定義を指し、論理レコード は COBOL 定義を指しています。

関連参照

[179 ページの『形式 F レコードのレイアウト』](#)

[181 ページの『形式 V レコードのレイアウト』](#)

[183 ページの『形式 S レコードのレイアウト』](#)

[184 ページの『形式 U レコードのレイアウト』](#)

固定長フォーマットの要求

固定長レコードは形式 F です。この形式を明示的に要求するには、RECORDING MODE F を使用します。

このタスクについて

RECORDING MODE 節は省略することができます。コンパイラは記録モードを次のように決定します。F 最大の長さが level-01 ファイルに関連付けられたレコードは、BLOCK CONTAINS 条項に違反し、次のいずれかのアクションを実行します。

- RECORD CONTAINS *integer* 節 (形式 1 RECORD 節) を使用して、レコード長 (バイト単位) を示します。
この節を使用すると、ファイルは必ずレコード長 *integer* の固定長形式になります (別の長さを指定した複数のレベル 01 レコード記述がファイルに関連付けられている場合でも)。
- RECORD CONTAINS *integer* 節を省略するが、ファイルに関連付けられているすべてのレベル 01 レコード記述項目を、同じ固定サイズで、OCCURS DEPENDING ON 節を含まないようにコーディングする。この固定サイズがレコード長になります。

非ブロック化形式 F ファイルでは、論理レコードはブロックと同じになります。

ブロック化形式 F ファイルでは、ブロック内の論理レコードの数 (ブロック化因数) はファイル内の (最後のブロックを除く) すべてのブロックで一定です。最後のブロックは少ないことがあります。

z/OS UNIX ファイルシステム 内のファイルはブロック化されることはありません。

関連概念

[178 ページの『論理レコード』](#)

関連タスク

[180 ページの『可変長フォーマットの要求』](#)

[181 ページの『スパン形式の要求』](#)

[183 ページの『不定形式の要求』](#)

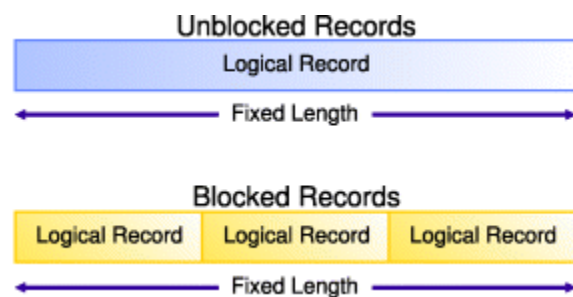
[178 ページの『レコード形式の指定』](#)

関連参照

[179 ページの『形式 F レコードのレイアウト』](#)

形式 F レコードのレイアウト

形式 F の QSAM レコードのレイアウトを以下に示します。



関連概念

[178 ページの『論理レコード』](#)

関連タスク

[179 ページの『固定長フォーマットの要求』](#)

z/OS DFSMS: Using Data Sets (固定長レコード・フォーマット)

関連参照

[181 ページの『形式 V レコードのレイアウト』](#)

[183 ページの『形式 S レコードのレイアウト』](#)

[184 ページの『形式 U レコードのレイアウト』](#)

可変長フォーマットの要求

可変長レコードは、形式 V または形式 D です。形式 D レコードは、ASCII テープ・ファイル上の可変長レコードです。形式 D レコードは、形式 V レコードと同じ方法で処理されます。

このタスクについて

使用 RECORDING MODE V 両方のための。RECORDING MODE 節は省略することができます。ファイルに関連付けられている最大のレベル 01 レコードの長さが、BLOCK CONTAINS 節でコーディングされているブロック・サイズを超えておらず、かつ次のいずれかを行った場合は、コンパイラーは記録モードを V と見なします。

- RECORD IS VARYING 節 (形式 3 RECORD 節) を使用する。

値を指定する場合 *integer-1* そして *integer-2* (RECORD IS VARYING FROM *integer-1* TO *integer-2*)、最大レコード長は、*integer-2* コード化された長さに関係なく、level-01 ファイルに関連付けられたレコード記述エントリ。整数サイズは、レコードのデータ項目の USAGE にかかわらず、最小および最大レコード長 (バイト数) を示します。

integer-1 および *integer-2* を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

- 使用 RECORD CONTAINS *integer-1* TO *integer-2* 条項 (format-2 RECORD 句)。 *integer-1* および *integer-2* を、ファイルに関連したレベル 01 レコード記述項目の最小長および最大長 (バイト数) と一致させてください。最大レコード長は *integer-2* の値です。
- RECORD 節を省略するが、サイズが異なるか OCCURS DEPENDING ON 節を含む、複数のレベル 01 レコード (ファイルに関連付けられる) をコーディングする。

最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

形式 V ファイルに READ INTO ステートメントを指定すると、そのファイルに関して読み取られたレコード・サイズが、コンパイラーによって生成される MOVE ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。例えば、READ ステートメントによって読み込まれる形式 V レコードに MOVE ステートメントを指定すると、移動されるレコードのサイズは、そのレベル 01 レコード記述に相当します。

形式 V ファイルに対する READ ステートメントを指定し、その後レベル 01 レコードの MOVE を続けて指定すると、実際のレコード長が使用されません。プログラムは、レベル 01 レコード記述で記述されたバイト数を移動しようとします。このバイト数が実際のレコード長を超え、プログラムによってアドレッシング可能な領域外に及ぶと、結果は予測できません。レベル 01 レコード記述によって記述されたバイト数が、読み取られた物理レコードより短い場合は、レベル 01 記述を超えるバイトは切り捨てられます。可変長レコードの実際の長さを見つけるには、ファイル定義 (FD) の RECORD 節の形式 3 で *data-name-1* を指定してください。

関連タスク

[179 ページの『固定長フォーマットの要求』](#)

[181 ページの『スパン形式の要求』](#)

[183 ページの『不定形式の要求』](#)

[178 ページの『レコード形式の指定』](#)

関連参照

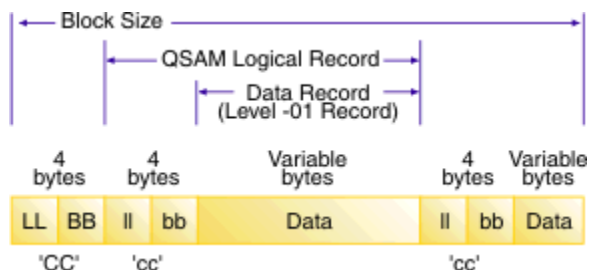
[13 ページの『FILE SECTION 記入項目』](#)

[181 ページの『形式 V レコードのレイアウト』](#)

Enterprise COBOL for z/OS 移行ガイド (
VS COBOL II ランタイムからの移行)

形式 V レコードのレイアウト

形式 V QSAM レコードでは、データの前に制御フィールドがあります。QSAM 論理レコード長は、プログラム内で定義されたレコード長に (制御フィールド用の) 4 バイトを加算することによって決まります。ただし、レコードの記述およびレコード長にこれらの 4 バイトを含めてはなりません。



CC

各ブロックの最初の 4 バイトには、制御情報が入ります。

LL は、ブロックの長さ (**CC** フィールドを含む) を指定する 2 バイトを表します。

BB は、システム使用に予約されている 2 バイトを表します。

cc

各論理レコードの最初の 4 バイトには、制御情報が入ります。

ll は、論理レコード長 (**cc** フィールドを含む) を指定する 2 バイトを表します。

bb は、システム使用に予約されている 2 バイトを表します。

ブロック長は次のように決定されます。

- 非ブロック化形式 V レコード: CC + cc + データ部分
- ブロック化形式 V レコード: CC + 各レコードの cc + 各レコードのデータ部分

オペレーティングシステムは、ファイルが書き込まれるときに制御バイトを提供します。制御バイトフィールドは、DATA DIVISION あなたのプログラムの。COBOL は、制御バイトを収容するのに十分な大きさの入出力バッファを割り振ります。バッファ内のこれらの制御フィールドをプログラムの中で使用することはできません。可変長レコードがユニット・レコード装置に書き込まれるときには、制御バイトは印刷も穿孔もされません。しかし、それらは、その他の外部ストレージ装置や、ストレージのバッファ域には入れられます。V モード・レコードを入力バッファから WORKING-STORAGE 域に移動すると、それらのレコードは制御バイトなしで移動されます。

z/OS UNIX ファイルシステム 内のファイルはブロック化されることはありません。

関連概念

[178 ページの『論理レコード』](#)

関連タスク

[180 ページの『可変長フォーマットの要求』](#)

関連参照

[179 ページの『形式 F レコードのレイアウト』](#)

[183 ページの『形式 S レコードのレイアウト』](#)

[184 ページの『形式 U レコードのレイアウト』](#)

スパン形式の要求

スパン・レコードとは、形式 S のレコードです。スパン・レコードとは、1 つまたは複数の物理ブロックに含めることができる QSAM 論理レコードのことです。

このタスクについて

磁気テープまたは直接アクセス装置に割り当てられる QSAM ファイルのスパン・レコードには、RECORDING MODE S をコーディングできます。z/OS UNIX ファイルシステム のファイルについてはスパ

ン・レコードを要求してはなりません。RECORDING MODE 節は省略することができます。コンパイラは記録モードを次のように決定します。S 最大レコード長が(バイト単位)プラス 4 は、BLOCK CONTAINS 句。

プログラム内でフォーマット S が指定されたファイルの場合、コンパイラは、フォーマット V に使用されるのと同じ規則を使用して最大レコード長を判定します。この長さは、RECORD 節の使用状況に基づいています。

フォーマット S レコードを含んでいるファイルを作成する場合に、レコードがブロックの残りのスペースより大きい場合、COBOL はレコードの 1 セグメントを書き込んでブロックを埋めます。レコードの残りの部分は、その長さに応じて、次の 1 つ以上のブロックに格納されます。COBOL は、32,760 バイトまでの長さの QSAM スパン・レコードをサポートします。

形式 S レコードを持つファイルを検索するときには、プログラムは完全なレコードだけを検索することができます。

形式 S ファイルのメリット: 形式 S レコードを持つファイルを定義すると、外部ストレージを効率よく利用できる上に、論理レコード長を使用してファイルを編成することができます。

- 直接アクセス装置でトラック容量を効率的に使用するためにブロック長を設定することができます。
- 論理レコード長を装置依存の物理ブロック長に合わせて調整する必要がありません。1 つの論理レコードが 2 つ以上の物理ブロックにわたることができます。
- 異なるタイプの直接アクセス・ストレージ間で論理レコードを転送したいときに、柔軟性が増します。

しかし、形式 S ファイルを処理するためには追加のオーバーヘッドが必要になります。

形式 S ファイルおよび READ INTO: 形式 S ファイルに READ INTO ステートメントを指定した場合、コンパイラは、そのファイルの読み取りにのみ使用するレコードのサイズを使用する MOVE ステートメントを生成します。読み取られたレコードが、レベル 01 レコード記述に対応していない場合には、期待どおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。

関連概念

[178 ページの『論理レコード』](#)

[182 ページの『スパン・ブロック化および非ブロック化ファイル』](#)

関連タスク

[179 ページの『固定長フォーマットの要求』](#)

[180 ページの『可変長フォーマットの要求』](#)

[183 ページの『不定形式の要求』](#)

[178 ページの『レコード形式の指定』](#)

関連参照

[13 ページの『FILE SECTION 記入項目』](#)

[183 ページの『形式 S レコードのレイアウト』](#)

スパン・ブロック化および非ブロック化ファイル

スパン・ブロック化された QSAM ファイルは、それぞれが 1 つ以上の論理レコードまたは論理レコードのセグメントを含むブロックから構成されます。スパン非ブロック化ファイルは、それぞれが 1 つの論理レコードまたは論理レコードの 1 つのセグメントを含む物理ブロックから構成されます。

スパン・ブロック化ファイルでは、論理レコードは固定長であっても可変長であっても構わず、そのサイズは物理ブロック・サイズと等しくても、より小さくても、より大きくても構いません。論理レコードと物理ブロックのサイズの間に必要とされる関係はありません。

スパン非ブロック化ファイルでは、論理レコードは固定長であっても可変長であっても構いません。物理ブロックが 1 つの論理レコードを含んでいる場合、ブロック長は論理レコード・サイズによって判別されます。論理レコードをセグメンテーションしなければならない場合、システムは、常に、できる限り大きな物理ブロックを書き込みます。論理レコード全体が 1 つのトラックに収まらない場合、システムは論理レコードをセグメンテーションします。

関連概念

[178 ページの『論理レコード』](#)

関連タスク

[181 ページの『スパン形式の要求』](#)

形式 S レコードのレイアウト

以下で説明するように、スパン・レコードの前には制御フィールドが置かれます。



各ブロックの前には、4 バイトのブロック記述子フィールド (図中の「BDF」) が置かれます。ブロック記述子フィールドは、各物理ブロックの始まりに 1 つだけあります。

ブロック内のレコードの各セグメントの前には、そのセグメントがレコード全体である場合でも、4 バイトのセグメント記述子フィールド (図中の「SDF」) が付加されます。セグメント記述子フィールドは、ブロック内の各レコード・セグメントにつき 1 つあります。セグメント記述子フィールドは、そのセグメントが最初、最後、または中間のいずれかであるかも示します。

これらのフィールドは、DATA DIVISION に記述しないため、COBOL プログラムで使用することはできません。

関連タスク

[181 ページの『スパン形式の要求』](#)

関連参照

[179 ページの『形式 F レコードのレイアウト』](#)

[181 ページの『形式 V レコードのレイアウト』](#)

[184 ページの『形式 U レコードのレイアウト』](#)

不定形式の要求

形式 U レコードは、未定義または未指定の特性を持ちます。形式 U を使用すると、形式 F または形式 V 仕様に適合しないブロックを処理することができます。

このタスクについて

形式 U ファイルを使用するときは、ストレージの各ブロックが 1 つの論理レコードになります。形式 U のファイルを読み取るとブロック全体が 1 レコードとして返されます。形式 U のファイルに書き込むと 1 レコードが 1 ブロックとして書き込まれます。コンパイラは、次のコードを実行する場合にのみ、記録モードを U と判定します。RECORDING MODE U。

異なるレコード形式で書き込まれたファイルの更新または拡張には、形式 U を使用しないことをお勧めします。フォーマット U を使用して、別のフォーマットで作成されたファイルを更新する場合、データ・セット・ラベルの RECFM 値が変更されるか、またはデータ・セットに別のフォーマットで作成されたレコードが入ることがあります。

レコード長は、RECORD 節が使用される方法に基づいて、プログラムの中で判別されます。

- を使用する場合 RECORD CONTAINS 整数条項 (format-1RECORD 句) レコード長は整数長さに関係なく level-01 ファイルに関連付けられたレコード記述エントリ。整数サイズは、そのデータ・セット項目の USAGE にかかわらず、レコード中のバイト数を示します。
- RECORD IS VARYING 節 (形式 3 RECORD 節) を使用する場合、integer-1 および integer-2 をコーディングするかどうかに基づいて、レコード長が決まります。

コーディングする場合 integer-1 そして integer-2 (RECORD IS VARYING FROMinteger-1TOinteger-2)、最大レコード長は integer-2 長さに関係なく level-01 ファイルに関連付けられたレコード記述エントリ。整数サイズは、レコードのデータ項目の USAGE にかかわらず、最小および最大レコード長 (バイト数) を示します。

integer-1 および integer-2 を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

- を使用する場合 RECORD CONTAINS *integer-1* TO *integer-2* 条項 (format-2 RECORD 条項)、*integer-1* として *integer-2* 最小長と最大長を一致させるバイト単位の level-01 ファイルに関連付けられたレコード記述エントリの最大レコード長は *integer-2* 価値。
- RECORD 節を省略する場合、最大レコード長は、ファイルに関連付けられた最大のレベル 01 レコード記述項目のサイズであると判別されます。

Format-U ファイルと READ INTO:指定する場合 READ INTO フォーマット U ファイル用のステートメントの場合、コンパイラは MOVE そのファイルで読み取ったレコードのサイズを使用するステートメント。読み取られたレコードが、レベル 01 レコード記述に対応していない場合には、期待どおりの結果が得られないことがあります。MOVE ステートメントに関するその他の規則はすべて適用されます。

関連タスク

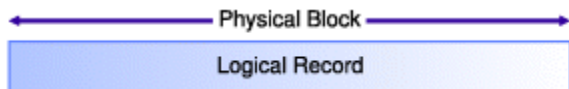
- [179 ページの『固定長フォーマットの要求』](#)
- [180 ページの『可変長フォーマットの要求』](#)
- [181 ページの『スパン形式の要求』](#)
- [178 ページの『レコード形式の指定』](#)

関連参照

- [13 ページの『FILE SECTION 記入項目』](#)
- [184 ページの『形式 U レコードのレイアウト』](#)

形式 U レコードのレイアウト

形式 U では、外部ストレージの各ブロックは 1 つの論理レコードとして扱われます。レコード長やブロック長フィールドはありません。



関連概念

- [178 ページの『論理レコード』](#)

関連タスク

- [183 ページの『不定形式の要求』](#)

関連参照

- [179 ページの『形式 F レコードのレイアウト』](#)
- [181 ページの『形式 V レコードのレイアウト』](#)
- [183 ページの『形式 S レコードのレイアウト』](#)

ブロック・サイズの設定

COBOL では、BLOCK CONTAINS 節を使用して物理レコードのサイズを設定します。この節を省略すると、コンパイラはレコードがブロック化されないものと見なします。

このタスクについて

ディスクまたはテープ上の QSAM ファイルをブロック化すると、処理速度が増し、ストレージ要件を最小限にすることができます。z/OS UNIX ファイルシステム 内のファイル、PDSE メンバー、およびスプール・データ・セットをブロック化できますが、そのようにしてもシステムがデータを保管する方法に影響はありません。

ブロックサイズを明示的に設定した場合、BLOCK CONTAINS 句では、サイズはデバイスの最大ブロック サイズを超えてはなりません。BLOCK CONTAINS 節の CHARACTERS 句を指定する場合、レコード中のデータ項目の USAGE にかかわらず、サイズではレコード中のバイト数を示す必要があります。形式 F ファイルについて設定するブロック・サイズは、レコード長の整数倍でなければなりません。

プログラムでテープ上の QSAM ファイルを使用する場合は、少なくとも 12 から 18 バイトの物理ブロック・サイズを使用してください。そうでない場合、以下のいずれかのアクションの実行時にパリティ検査が行われると、ブロックはスキップオーバーされます。

- 12 バイトよりも小さいレコード・ブロックの読み取り
- 18 バイトよりも小さいレコード・ブロックの書き込み

一般に、ブロックが大きいとパフォーマンスは向上します。ブロック・サイズがわずかに数キロバイトの場合は、特に効率が低下します。最低でも数十キロバイトのブロック・サイズを選択してください。レコードのブロック化を指定してブロック・サイズを省略すると、装置の使用効率とデータ転送速度を考慮した最適なブロック・サイズがシステムによって選択されます。

z/OS によるブロック・サイズの決定: パフォーマンスを最大化するには、COBOL ソース・プログラム内でブロック化ファイルのブロック・サイズを明示的に設定しないでください。新しいブロック化データ・セットの場合は、z/OS にシステム決定のブロック・サイズを提供させるようにする方が簡単です。このフィーチャーを使用するには、以下のガイドラインに従ってください。

- コード `BLOCK CONTAINS 0` ソースプログラムでまたはコンパイルして `BLOCK0` オプション。詳細については `BLOCK0`、見る [383 ページの『BLOCK0』](#)。
- ソース・プログラムに `RECORD CONTAINS 0` をコーディングしない。
- コーディングしないでください `BLKSIZEJCL` の値 `DD` 声明。

明示的なブロック・サイズの設定: ブロック・サイズを明示的に設定したい場合は、以下の指針に従うと、プログラムが最も柔軟になります。

- コード `BLOCK CONTAINS 0` ソースプログラムでまたはコンパイルして `BLOCK0` オプション。
- `DD` 名定義 (`JCL DD` ステートメント) で、`BLKSIZE` 値をコーディングする。

z/OS における拡張形式データ・セットの場合、z/OS DFSMS は物理レコードに 32 バイトのブロック接尾部を追加します。ブロック・サイズを明示的に指定する (`JCL` または `ISPF` を介して) 場合には、このブロック接尾部のサイズをブロック・サイズに含めないでください。このブロック接尾部は、プログラムの中で使用することはできません。z/OS DFSMS は、ブロック接尾部を読み取るためのスペースを割り振ります。ただし、直接アクセス装置の 1 つのトラックに収まる拡張フォーマット・データ・セットのブロック数を計算するとき、ブロック・サイズにブロック接尾部のサイズを含める必要があります。

32760 より大きいブロックサイズを直接指定すると、`BLOCK CONTAINS` 句または間接的に使用して `BLOCK CONTAINS n RECORDS`、`OPEN` データセットをテープ上に定義しない限り、データセットはファイルステータスコード 90 で失敗します。

既存のブロック化データ・セットの場合は、次のようにすると最も簡単です。

- コード `BLOCK CONTAINS 0` ソースプログラムでまたはコンパイルして `BLOCK0` オプション。
- `DD` 名定義に `BLKSIZE` 値をコーディングしません。

`DD` 名定義から `BLKSIZE` を省くと、ブロック・サイズは、システムによってデータ・セット・ラベルから自動的に取得されます。

LBI の利用: 大きなブロック・サイズ用のラージ・ブロック・インターフェース (LBI) を使用して、テープ・データ・セットのパフォーマンスを向上させることができます。LBI が使用可能であれば、COBOL 実行時には、システム決定のブロック・サイズを使用するテープ・ファイルに対して、この機能が自動的に使用されます。LBI は、`JCL` または `BLOCK CONTAINS` 句。LBI を使用すると、ブロック・サイズが 32760 (磁気テープ装置でサポートされる場合) を超えても構いません。

LBI はどんな場合にでも使用されるわけではありません。以下の場合には、32760 を超えるブロック・サイズを使用しようとすると、コンパイル時に診断され、`OPEN` 障害となります。

- スパン・レコード
- `OPEN I-O`

32760 を超えるブロック・サイズを使用すると、別のシステムでテープを読み取れなくなる場合があります。32760 より大きなブロック・サイズで作成したテープは、32760 より大きなブロック・サイズをサポートする磁気テープ装置を持つシステムでしか読み取ることができません。ファイル、装置、またはオペレーティング・システム・レベルに関して、指定したブロック・サイズが大きすぎると、ランタイム・メッセージが出されます。

システム決定のブロック・サイズを 32760 に限定するには、どこにも BLKSIZE を指定せずに、以下のいずれかの項目を 32760 に設定してください。

- データ・セットの DD ステートメントの BLKSZLIM キーワード
- BLKSZLIM キーワードを使用した、データ・クラスの BLKSZLIM (システム・プログラマーによる設定が必要)
- キーワード TAPEBLKSZLIM を使用して、SYS1.PARMLIB の DEVSUPxx メンバーにあるシステムのブロック・サイズ限界 (システム・プログラマーによる設定が必要)

ブロック・サイズ限界は、以下の項目を検査してコンパイラーによって検出される最初のゼロ以外の値です。

BLKSIZE または BLKSZLIM 値をどのソースからも利用できないと、システムは BLKSIZE を 32760 に限定します。その後、32760 を超えるブロック・サイズを以下のいずれかの方法で使用可能にすることができます。

- 指定する BLKSZLIM32760 より大きい値 DD ファイルのステートメントと使用 BLOCK CONTAINS 0COBOL ソースで。
- DD ステートメントまたは COBOL ソースの BLOCK CONTAINS 節で、32760 より大きな値を BLKSIZE に指定する。

BLKSZLIM は、装置独立です。

ブロック サイズと DCB RECFM サブパラメータ: 下 z/OS、コード化することができます S または T オプションの DCB RECFM サブパラメータ:

- 使用 S (標準) オプション DCB RECFM 標準ブロックのみ (ファイルの最後のブロックを除き、ファイル内に切り捨てられたブロックや未入力のトラックがないブロック) を含むフォーマット F レコードのサブパラメータ。S は、テープ上のレコードに関してもサポートされます。これは、レコードが DASD またはテープ上にない場合には無視されます。

この標準ブロック・オプションを使用すると、特に直接アクセス装置の場合は、入出力のパフォーマンスが向上する可能性があります。

- QSAM ファイルの場合、T (トラック・オーバーフロー) オプションはもう有用ではありません。

関連タスク

[177 ページの『COBOL での QSAM ファイルおよびレコードの定義』](#)

[z/OS DFSMS: Using Data Sets](#)

関連参照

[13 ページの『FILE SECTION 記入項目』](#)

[383 ページの『BLOCK0』](#)

BLOCK CONTAINS 節 (*Enterprise COBOL for z/OS* 言語解説書)

QSAM ファイルの入出力ステートメントのコーディング

以下の入力および出力ステートメントをコーディングして、QSAM ファイルまたはバイトストリームファイルを処理することができます。z/OS UNIX ファイルシステム QSAM を使用:OPEN、READ、WRITE、REWRITE、そして CLOSE。

このタスクについて

OPEN

ファイルの処理を開始します。どの QSAM ファイルも、INPUT、OUTPUT、または EXTEND (装置の能力に応じて) としてオープンすることができます。

直接アクセス・ストレージ・デバイス上の QSAM ファイルは、I-O としてオープンすることもできます。z/OS ユニックスファイルを I-O として開くことはできません。そうしようとすると、ファイル状況は 37 になります。

READ

ファイルからレコードを読み取ります。順次処理では、プログラムは、レコードをファイルの作成時に入力されたのと同じ順序で1つずつ読み取ります。

WRITE

ファイル内にレコードを作成します。プログラムは、ファイルの終わりに新しいレコードを書き込みます。

REWRITE

レコードを更新します。ファイルを更新することはできません z/OS UNIX ファイルシステムを使用して REWRITE。

CLOSE

ファイルとプログラムの間の接続を解放します。

関連タスク

[187 ページの『QSAM ファイルのオープン』](#)

[188 ページの『QSAM ファイルの動的作成』](#)

[188 ページの『QSAM ファイルへのレコードの追加』](#)

[189 ページの『QSAM ファイルの更新』](#)

[189 ページの『QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み』](#)

[190 ページの『QSAM ファイルのクローズ』](#)

関連参照

OPEN ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

READ ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

WRITE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

REWRITE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

CLOSE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

ファイル状況キー (*Enterprise COBOL for z/OS 言語解説書*)

QSAM ファイルのオープン

プログラムで READ、WRITE、または REWRITE 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には、まず OPEN ステートメントを使用してそのファイルを開く必要があります。

このタスクについて

OPEN ステートメントが機能するのは、以下の両方の条件が真である場合です。

- ファイルが使用可能であるか、または動的に割り振られている。
- DD 名定義またはファイルのデータ・セット・ラベルにコーディングされた固定ファイル属性が、SELECT 節および FD 項目でそのファイル用にコーディングされた属性と一致している。

ファイル編成属性、コード・セット、最大レコード・サイズ、またはレコード・フォーマット (固定または可変) で不一致があると、ファイル状況コードは 39 になり、OPEN ステートメントは失敗します。z/OS UNIX ファイルシステムのファイルのオープン時には、最大レコード・サイズおよびレコード・フォーマットの不一致はエラーではありません。

固定長 QSAM ファイルの場合コーディングする場合 RECORD CONTAINS 0 の中に FD エントリでは、レコードサイズ属性が競合していません。レコード・サイズは DD ステートメントまたはデータ・セット・ラベルから取られ、OPEN ステートメントは正常に終了します。

プログラムの実行中にファイルが再度オープンされないように、CLOSE WITH LOCK をコーディングしてください。

テープ・ファイルを逆順に処理するには、OPEN ステートメントの REVERSED オプションを使用してください。ファイルは最後に位置付けられ、READ ステートメントでは、データ・レコードが逆順に読み取られます。REVERSED オプションがサポートされるのは、固定長レコードを持つファイルの場合だけです。

関連タスク

[188 ページの『QSAM ファイルの動的作成』](#)

[194 ページの『ファイル属性をプログラムと一致させる』](#)

関連参照

OPEN ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

QSAM ファイルの動的作成

ある QSAM ファイルがオペレーティング・システム上で利用可能になっていないときに、COBOL プログラムがそのファイルの作成を指定することがあります。ある特定環境では、ファイルが動的に作成されます。

このタスクについて

QSAM ファイルは、利用可能の上 z/OS 有効なパスワードを使用してオペレーティングシステムに識別された場合 DD 声明、export 環境変数のコマンド、または TSOALLOCATE 指示。そうしないと、ファイルは利用不能です。

DD ステートメントの DD 名のミススペルは、DD ステートメントが欠落しているのと同じであること、また、無効な値を指定した環境変数は、変数が設定されていないのと同じであることに注意してください。

ランタイム・オプション CBLQDA を使用しており、以下の状況のいずれか 1 つが存在していれば、その QSAM ファイルが暗黙的に作成されます。

- オプション・ファイルが EXTEND または I-O として開かれている。

オプション・ファイルとは、プログラムが実行されるたびに、必ずしも使用可能である必要はないファイルです。開かれるファイルを定義します INPUT、I-O、または EXTEND モードをオプションとしてコーディングすることで SELECT OPTIONAL 条項の FILE-CONTROL 段落。

- OPTIONAL 句に関係なく、そのファイルが OUTPUT 用にオープンされている。

ファイルは、インストール先で設定されたシステム・デフォルト属性と、プログラムの SELECT 節および FD 記入項目でコーディングされた属性を用いて割り振られます。

この暗黙の割り振りメカニズムを、環境変数を使用して行うファイルの明示的な動的割り振りとは混乱しないでください。明示的な動的割り振りでは、有効な環境変数が設定されている必要があります。CBLQDA サポートが使用されるのは、上述されたように QSAM ファイルが利用できず、有効な環境変数が設定されていない場合だけです。

下 z/OS、を使用して作成されたファイル CBLQDA オプションは一時的なデータセットであり、プログラムの実行後は存在しなくなります。

関連タスク

[187 ページの『QSAM ファイルのオープン』](#)

QSAM ファイルへのレコードの追加

QSAM ファイルに追加を行うためには、ファイルを EXTEND としてオープンし、WRITE ステートメントを使用して、ファイルの最後のレコードの直後にレコードを追加します。

このタスクについて

I-O としてオープンされたファイルにレコードを追加したい場合には、ファイルをいったんクローズしてから、EXTEND としてオープンしなければなりません。

関連参照

READ ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

WRITE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

QSAM ファイルの更新

QSAM ファイルを更新できるのは、それが直接アクセス・ストレージ・デバイスに存在する場合のみです。z/OS UNIX ファイルシステム のファイルを更新することはできません。

このタスクについて

次のようにして、既存のレコードを同じ長さの別のレコードで置き換えてください。

手順

1. ファイルを I-O としてオープンします。
2. 既存のレコードを更新するには、REWRITE を使用します。(REWRITE の前の最後のファイル処理ステートメントは、成功した READ ステートメントでなければなりません。)

タスクの結果

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。

関連参照

REWRITE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

QSAM ファイルのプリンターまたはスプール・データ・セットへの書き込み

COBOL には、印刷されるページのサイズを制御するため、およびレコードの垂直位置決めを制御するための言語ステートメントが用意されています。

このタスクについて

ページ・サイズの制御: 印刷されるページのサイズ (ページの上部マージンと下部マージンの間の行数および脚注域の行数) を制御するには、FD 記入項目の LINAGE 節を使用します。LINAGE 節を使用すると、COBOL では ADV コンパイラー・オプションも要求されたかのようにファイルを処理します。

を使用する場合 LINAGE 条項と組み合わせる WRITE BEFORE|AFTER ADVANCING *nn* LINES 設定する値には注意してください。ADVANCING *nn* LINES 句が使用されると、COBOL はまず、LINAGE-COUNTER と *nn* の合計を計算します。それ以降のアクションは、*nn* のサイズによって異なります。LINAGE-COUNTER が増加されると、END-OF-PAGE 命令句が実行されます。したがって、LINAGE-COUNTER は、END-OF-PAGE 句の実行時に、現行の脚注域ではなく次の論理ページを指していることがあります。

AT END-OF-PAGE または NOT AT END-OF-PAGE 命令句は、書き込み操作が正常に完了した場合にのみ実行されます。書き込み操作が成功しなかった場合には、制御が WRITE ステートメントの終わりに渡され、すべての条件句が省略されます。

レコードの垂直位置決めの制御: 印刷ページに書き込むそれぞれのレコードの垂直位置決めを制御するには、WRITE ADVANCING ステートメントを使用します。

BEFORE ADVANCING は、ページが送られる前にレコードを印刷します。AFTER ADVANCING は、ページが送られた後でレコードを印刷します。

ADVANCING の後に、ページが送られることになる行数を整数 (または *mnemonic-name* を指定した *identifier*) で指定してください。WRITE ステートメントで、ADVANCING 句を省略した場合、その効果は次のようにコーディングした場合と同じです。

```
AFTER ADVANCING 1 LINE
```

関連参照

WRITE ステートメント (Enterprise COBOL for z/OS 言語解説書)

QSAM ファイルのクローズ

プログラムを QSAM ファイルから切り離すには、CLOSE ステートメントを使用します。既にクローズされているファイルをクローズしようとすると、論理エラーになります。

このタスクについて

QSAM ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。

- 実行単位が正常に終了すると、実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 実行単位が異常終了した場合は、TRAP (ON) ランタイム・オプションが有効化されていれば、その実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 言語環境の条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが処理を再開する場合は、実行単位の中で COBOL プログラム (再度呼び出されて再入される可能性のある) のいずれかで定義されている、オープン状態のすべてのファイルがランタイムによって閉じられます。

(条件が処理された後で) プログラムが実行を再開する位置を変更することができます。このためには、言語環境の CEEMRCR 呼び出し可能サービスを使用して再開カーソルを移動するか、または C longjmp のような言語構造体を使用します。

- COBOL サブプログラムに CANCEL を使用すると、そのプログラムに定義されているオープン状態の非外部ファイルは実行時にクローズされます。
- INITIAL 属性を持つ COBOL サブプログラムが制御権を戻すと、そのプログラムに定義されているオープン状態の非外部ファイルは実行時にクローズされます。
- マルチスレッド・アプリケーションのスレッドが終了すると、同じスレッド内部からオープンした外部ファイルと非外部ファイルはいずれもクローズされます。

ファイルステータスキーデータ項目 DATA DIVISION これらの暗黙の CLOSE 操作は実行されますが、EXCEPTION/ERROR 宣言的である呼び出されません。

エラー: マルチスレッド・アプリケーションで QSAM ファイルをオープンした場合は、ファイルをオープンしたのと同じ実行スレッドからクローズする必要があります。異なるスレッドからファイルをクローズしようとすると、ファイル状況コードの条件 90 でクローズが失敗します。

関連参照

CLOSE ステートメント (Enterprise COBOL for z/OS 言語解説書)

QSAM ファイルのエラーの処理

入力ステートメントまたは出力ステートメントが失敗しても、COBOL がユーザーに代わって訂正処置を行うことはありません。重大エラーではない入出力エラーが発生した後で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。

このタスクについて

COBOL は、特定の QSAM 入出力エラーを代行受信して処理するために、以下の方法を提供します。

- ファイル終了句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 節
- INVALID KEY 句

FILE STATUS キーまたは宣言をコーディングしなかった場合に、重大な QSAM 処理エラーが起こると、メッセージが出され、言語環境条件が合図されます。ランタイム・オプション ABTERMENC (ABEND) が指定されていると、この条件の結果として、異常終了が引き起こされます。

を使用する場合 FILE STATUS 条項または EXCEPTION/ERROR 宣言型、コード EROPT=ACC の中に DCB の DD そのファイルのステートメント。そうしなければ、COBOL プログラムは、いくつかのエラー条件が発生した後で処理を継続することができなくなります。

を使用する場合 FILE STATUS 句を使用する場合は、必ずキーを確認し、その値に基づいて適切なアクションを実行してください。キーを調べなくても、プログラムが実行を継続できる可能性があります、予期したものではない結果になることがあります。

関連タスク

[267 ページの『入出力操作でのエラーの処理』](#)

QSAM ファイルの操作

COBOL プログラムで QSAM ファイルを操作するには、それらのファイルの定義、割り振り、取得を行うとともに、それらのファイル属性をプログラム内の属性に一致させます。また、ストライプ拡張フォーマット QSAM データ・セットを使用して、パフォーマンスの向上に役立てることができます。

このタスクについて

関連タスク

[191 ページの『QSAM ファイルの定義および割り振り』](#)

[193 ページの『QSAM ファイルの検索』](#)

[194 ページの『ファイル属性をプログラムと一致させる』](#)

[197 ページの『ストライプ拡張形式 QSAM データ・セットの使用』](#)

関連参照

[197 ページの『QSAM ファイル用のバッファの割り振り』](#)

QSAM ファイルの定義および割り振り

QSAM ファイルまたはバイトストリームファイルを定義できます。z/OS UNIX ファイルシステムのいずれかを使用して DD ステートメントまたは環境変数。これらのファイルの割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。

このタスクについて

環境変数を使用する場合、名前は大文字でなければなりません。次のいずれかの方法で、MVS データ・セットを指定します。

- DSN(*data-set-name*)
- DSN(*data-set-name(member-name)*)

data-set-name は完全に修飾されていなければならず、一時データ・セットにすることはできません (つまり、& で開始させないでください)。

制約事項: 環境変数を使用して PDS や PDSE を作成することはできません。

オプションとして、DSN の後に、以下の属性を任意の順序で指定することができます。

- 後処理の値 (NEW、OLD、SHR、または MOD)
- TRACKS または CYL
- SPACE(*nnn,mmm*)
- VOL(*volume-serial*)
- UNIT(*type*)

- KEEP、DELETE、CATALOG、または UNCATALOG
- STORCLAS(*storage-class*)
- MGMTCLAS(*management-class*)
- DATACLAS(*data-class*)

環境変数または DD 定義を使用して、z/OS UNIX ファイルシステム のファイルを定義することができます。そうするには、ASSIGN 節の外部名と一致する名前を使用して、以下のいずれかの項目を定義してください。

- PATH='absolute-path-name' および FILEDATA=BINARY を使用する DD 割り振り
- 値 PATH(*pathname*) を持つ環境変数 (ここで、*pathname* は / で始まる絶対パス名です)

以前の COBOL リリースとの互換性のため COBOL for OS/390 & VM2.2、指定することもできます FILEDATA=TEXT 使用する場合 DD 割り当て z/OS ユニックスファイルただし、この使用法は推奨されません。テキストファイル処理するには z/OS UNIX ファイルシステム、使用 LINE SEQUENTIAL 組織。QSAM を使用して z/OS UNIX ファイルシステムのテキスト・ファイル処理する場合は、環境変数を使用してファイルを定義することはできません。

QSAM ファイルを定義する場合には、以下のパラメーターを使用します。

表 20. QSAM ファイル割り振り		
目的	使用する DD パラメーター	使用する EV キーワード
ファイルに名前を付ける。	DSNAME (データ・セット名)	DSN
ファイルに割り振る入出力装置のタイプと数量を選択する。	UNIT	タイプの UNIT の場合のみ
ファイルが常駐するボリュームと、ボリュームの取り付けについて指示を与える。	VOLUME (またはシステムに出力ボリュームを選択させる)	VOL
ファイルに必要なスペースのタイプおよび量を割り振る。(直接アクセス・ストレージ・デバイスの場合のみ)	SPACE	スペースの量 (1 次と 2 次のみ) の場合は、SPACE。スペースのタイプの場合は、TRACKS?または CYL。
ファイルに関連付けられるラベルのタイプおよび内容の一部を指定する。	LABEL	該当なし
ジョブ・ステップの完了後にファイルをカタログするか、後続ステップに渡すか、保存するかを指示する。	DISP	NEW、OLD、SHR、MOD に、KEEP、DELETE、CATALOG、または UNCATALOG の指定を追加したもの。
追加したいすべてのデータ制御ブロック情報を完成させる。	DCB サブパラメーター	該当なし

QSAM ファイルの一部の情報は常に、FILE-CONTROL 段落、FD エントリー、およびその他の COBOL 節にコード化する必要があります。その他の情報は、出力ファイルについての DD ステートメントまたは環境変数でコーディングする必要があります。入力ファイルの場合、システムはファイル・ラベルから情報を取得することができます (標準ラベル・ファイルの場合)。入力ファイルについての DD ステートメントで DCB 情報が指定されると、それはデータ・セット・ラベル上の情報をオーバーライドします。例えば、新規の直接アクセス装置ファイルに割り振られるスペースの量は、DD ステートメントで SPACE パラメーターを使用して設定することができます。

QSAM ファイルの一部の特性は、COBOL 言語では表現できませんが、ファイルの DD ステートメントで DCB パラメーターを使用してコーディングすることができます。DCB パラメーターのサブパラメーターで、システムがデータ・セット定義を完成させるのに必要な、以下の項目を含む情報を指定してください。

- ブロックサイズ (BLKSIZE=)、もし BLOCK CONTAINS 0 RECORDS または BLOCK0 オプションはコンパイル時に指定されました (推奨)
- レコードの書き込みまたは読み取りにおいてエラーが発生した場合に実行されるオプション。
- TRACK OVERFLOW または標準ブロック。
- カード読取装置または穿孔装置の操作モード。

DD DUMMY にコーディングされた DCB 属性は、COBOL プログラムの FD 記入項目にコーディングされた属性をオーバーライドしません。

[582 ページの『例: 環境変数の設定とアクセス』](#)

関連タスク

[184 ページの『ブロック・サイズの設定』](#)

[177 ページの『COBOL での QSAM ファイルおよびレコードの定義』](#)

[175 ページの『ファイルの割り振り』](#)

関連参照

[383 ページの『BLOCK0』](#)

[193 ページの『QSAM ファイルを作成するためのパラメーター』](#)

[MVS プログラム管理: ユーザーズ・ガイドおよび解説書](#)

QSAM ファイルを作成するためのパラメーター

次に示す DD ステートメントのパラメーターは、QSAM ファイルを作成するために頻繁に使用されます。

```

DSNAME= [ dataset-name
DSN=     dataset-name(member-name)
          &&name
          &&name(member-name) ]

UNIT= ( name[,unitcount] )

VOLUME= ( [PRIVATE] [,RETAIN] [,vol-sequence-num] [,volume-count] ...
VOL=     ... [ ,SER=(volume-serial[,volume-serial]...) )
           [ ,REF= [ dsname
                     *.ddname
                     *.stepname.ddname
                     *.stepname.procstep.ddname ] ] )

SPACE= ( [TRK
          CYL
          average-record-length ] , (primary-quantity[,secondary-quantity][,directory-quantity]))

LABEL= ( [Data-set-sequence-number,] [ NL
                                       SL
                                       SUL ] [ ,EXPDT= [ yyddd
                                                         yyyy/ddd ] ]
         [ ,RETPD=xxxx ] )

DISP= ( [ NEW
         MOD ] [ ,DELETE
                 ,KEEP
                 ,PASS
                 ,CATLG ] [ ,DELETE
                             ,KEEP
                             ,CATLG ] )

DCB= ( subparameter-list )

```

関連タスク

[191 ページの『QSAM ファイルの定義および割り振り』](#)

QSAM ファイルの検索

QSAM ファイルは、カタログされていてもいなくても、ジョブ制御ステートメントまたは環境変数を使用して検索することができます。

このタスクについて

カタログされたファイル

ボリュームやスペースなど、すべてのデータ・セット情報は、カタログおよびファイル・ラベルに保管されています。コーディングする必要があるのは、データ・セット名と後処理方法です。DD ステートメントを使用する場合、これは DSNAME パラメーターと DISP パラメーターです。環境変数を使用する場合、これは DSN パラメーターとパラメーター OLD、SHR、または MOD のいずれか 1 つです。

カタログされていないファイル

一部の情報はファイル・ラベルに保管されていますが、*dsname* および後処理方法のほかにユニットおよびボリューム情報をコーディングしなければなりません。

JCL を使用しており、そのファイルを現行ジョブ・ステップまたは現行ジョブ内の前のジョブ・ステップで作成した場合は、前の DD ステートメントを参照してデータ・セット情報の大部分を得ることができます。ただし、DSNAME および DISP はコーディングする必要があります。

関連参照

[194 ページの『QSAM ファイルを検索するためのパラメーター』](#)

QSAM ファイルを検索するためのパラメーター

次に示す DD ステートメントのパラメーターは、前に作成したファイルを検索するために使用されます。

```
DSNAME= [ dataset-name  
DSN=    dataset-name(member-name)  
        *.ddname  
        *.stepname.ddname  
        &&name  
        &&name(member-name) ]  
  
UNIT=   ( name[,unitcount] )  
  
VOLUME= ( subparameter-list )  
VOL=  
  
LABEL=  ( subparameter-list )  
  
DISP=   ( [ OLD  
           SHR  
           MOD ] [ ,DELETE  
                  ,KEEP  
                  ,PASS  
                  ,CATLG  
                  ,UNCATLG ] [ ,DELETE  
                               ,KEEP  
                               ,CATLG  
                               ,UNCATLG ] )  
  
DCB=    ( subparameter-list )
```

関連タスク

[193 ページの『QSAM ファイルの検索』](#)

ファイル属性をプログラムと一致させる

DD ステートメントまたはデータ・セット・ラベルにコーディングされた固定ファイル属性と、SELECT 節および FD 項目でそのファイル用にコーディングされた属性が整合していない場合、プログラムの OPEN ステートメントが機能しないことがあります。

このタスクについて

ファイル編成、レコード・フォーマット (固定長または可変長)、レコード長、またはコード・セットの属性が一致していないと、ファイル状況コードが 39 になり、OPEN ステートメントが失敗します。z/OS UNIX ファイルシステムのファイルについては、例外があります。すなわち、レコード形式およびレコード長が一致していなくても、エラーとなりません。

一般的なファイル状況 39 の問題が起こらないようにするには、既存ファイルまたは新規ファイルの処理に関するガイドラインに従ってください。

ファイルを DD 声明または TSOALLOCATE コマンドを実行し、COBOL プログラムでファイルの作成を指定すると、Enterprise COBOL ファイルを動的に割り当てます。ファイルがオープンされるときには、プログラム内でコーディングされたファイル属性が使用されます。このため、ファイル属性の矛盾を心配する必要はありません。

JCL または環境変数内の情報は、データ・セット・ラベル内の情報をオーバーライドするので注意してください。

関連タスク

[195 ページの『既存ファイルの処理』](#)

[196 ページの『新規ファイルの処理』](#)

[187 ページの『QSAM ファイルのオープン』](#)

関連参照

[13 ページの『FILE SECTION 記入項目』](#)

既存ファイルの処理

プログラムで既存のファイルを処理するときには、COBOL プログラム内のファイルの記述をデータ・セットのファイル属性と矛盾しないようにコーディングします。最大レコード長を定義するには、以下の指針を使用してください。

このタスクについて

表 21. QSAM ファイルの最大レコード長	
形式:	指定:
V または S	データ・セットの長さ属性よりも 4 バイト小さくする。
F	データ・セットの長さ属性と同じ値。
U	データ・セットの長さ属性と同じ値。

プログラムで可変長(フォーマット V) レコードを定義する最も簡単な方法は、FD 項目で RECORD IS VARYING FROM *integer-1* TO *integer-2* 節を使用して、*integer-2* に適切な値を設定する方法です。急行基になる要素に関係なく、整数のサイズはバイト単位で表されます。USAGE レコード内のデータ項目。たとえば、データセットの長さ属性が 104 であると判定したとします。バイト (LRECL=104)。最大レコード長は、RECORD IS VARYING 条項からではなく、level-01 レコードの説明については、次のコードを使用してプログラム内で format-V ファイルを定義できます。

```
FILE SECTION.  
FD  COMMUTER-FILE-MST  
   RECORDING MODE IS V  
   RECORD IS VARYING FROM 4 TO 100 CHARACTERS.  
01  COMMUTER-RECORD-A    PIC X(4).  
01  COMMUTER-RECORD-B    PIC X(75).
```

前の例の既存のファイルが、形式 V ではなく形式 U である場合を想定します。104 バイトがすべてユーザー・データである場合、プログラム内で以下のコードを使用してファイルを定義することができます。

```
FILE SECTION.  
FD  COMMUTER-FILE-MST  
   RECORDING MODE IS U  
   RECORD IS VARYING FROM 4 TO 104 CHARACTERS.  
01  COMMUTER-RECORD-A    PIC X(4).  
01  COMMUTER-RECORD-B    PIC X(75).
```

プログラム内で固定長レコードを定義するには、RECORD CONTAINS 整数句、またはこの句を省略してすべてをコード化する level-01 レコードの説明を同じ固定サイズにします。どちらの場合も、データ・セットの長さ属性の値と等しい値を使用してください。実行時に同じプログラムを使用して異なるファイル进行处理する予定で、それらのファイルの固定長が異なる場合は、次のようにコーディングしてレコード長の競合を回避します。RECORD CONTAINS 0。

既存のファイルが ASCII データセット (DCB=(OPTCD=Q))、あなたは CODE-SET 条項の FD ファイルのエントリ。

関連タスク

[196 ページの『新規ファイルの処理』](#)

[179 ページの『固定長フォーマットの要求』](#)

[180 ページの『可変長フォーマットの要求』](#)

[183 ページの『不定形式の要求』](#)

[187 ページの『QSAM ファイルのオープン』](#)

関連参照

[13 ページの『FILE SECTION 記入項目』](#)

新規ファイルの処理

COBOL プログラムが、プログラム実行前に使用可能にされる新規ファイルにレコードを書き込む場合、DD ステートメント、環境変数、または割り振りでのファイル属性が、プログラム内の属性と矛盾しないようにしてください。

このタスクについて

通常、ファイルを事前定義する場合には、最小限のパラメーターをコーディングするだけで済みます。ただし、データ・セットの長さ属性を明示的に設定する必要がある場合 (例えば、ISPF 割り振りパネルを使用している場合、または DD ステートメントがバッチ・ジョブ用であり、その中でプログラムが RECORD CONTAINS 0 を使用する場合)、次の指針に従ってください。

- 形式 V および形式 S ファイルの場合は、プログラム内で定義されているものよりも 4 バイト大きい長さ属性を設定します。
- 形式 F および形式 U ファイルの場合は、プログラム内で定義されているものと同じ長さ属性を設定します。
- ファイルを OUTPUT としてオープンし、それをプリンターに書き込む場合は、ADV コンパイラー・オプションと、プログラムで使用する言語によって、コンパイラーがレコード長に紙送り制御文字のための 1 バイトを追加することがあります。そのような場合には、LRECL 値をコーディングする際に、追加されるバイトを長さに含めてください。

例えば、プログラムに可変長レコードを含むファイルの以下のコードが入っている場合、DD ステートメントまたは割り振りにおける LRECL 値は 54 になります。

```
FILE SECTION.  
FD  COMMUTER-FILE-MST  
   RECORDING MODE IS V  
   RECORD CONTAINS 10 TO 50 CHARACTERS.  
01  COMMUTER-RECORD-A   PIC X(10).  
01  COMMUTER-RECORD-B   PIC X(50).
```

関連タスク

[195 ページの『既存ファイルの処理』](#)

[179 ページの『固定長フォーマットの要求』](#)

[180 ページの『可変長フォーマットの要求』](#)

[183 ページの『不定形式の要求』](#)

[187 ページの『QSAM ファイルのオープン』](#)

[188 ページの『QSAM ファイルの動的作成』](#)

ストライプ拡張形式 QSAM データ・セットの使用

ストライプ拡張形式 QSAM データ・セットは、大容量のデータを含んだファイルを処理するアプリケーションや入出力操作に要する時間が全体のパフォーマンスに大きな影響を与えるアプリケーションにメリットをもたらすことがあります。

このタスクについて

ストライプ拡張形式 QSAM データ・セットは、複数のボリュームにわたる拡張形式の QSAM データ・セットであるため、並列データ・アクセスを可能にします。

QSAM ストライプ・データ・セットを使用することから最大限の効果を得るためには、z/OS DFSMS が必要とされる数のバッファを 16 MB 境界より上に割り振ることができなければなりません。QSAM ストライプ・データ・セットに割り振られるファイルを含むアプリケーションを開発するときには、これらの指示に従ってください。

- バッファを 16 MB 境界より上に割り振ることができないファイルについては、QSAM ストライプ・データ・セットの使用を避けます。
- ファイルについての FILE-CONTROL 記入項目で RESERVE 節を省略します。これによって、z/OS DFSMS がデータ・セットについての最適なバッファ数を判別できるようになります。
- プログラムをコンパイルするには、DATA(31)そして RENT コンパイラオプションを設定し、プログラムオブジェクト AMODE 31。
- ファイルが形式 F、形式 V、または形式 U レコードを持つ EXTERNAL ファイルである場合は、ALL31(ON)ランタイム・オプションを指定します。

すべてのストライプ・データ・セットは拡張フォーマット・データ・セットですが、すべての拡張フォーマット・データ・セットがストライプ・データ・セットというわけではないことに注意してください。

関連タスク

z/OS DFSMS: Using Data Sets

QSAM ファイル用のバッファの割り振り

z/OS DFSMS は、QSAM ファイルに合わせて、16 MB 境界より上または下にファイルの入出力データを保管するためのバッファを自動的に割り振ります。

ほとんどの QSAM ファイルのバッファは、16 MB 境界より上に割り振られます。例外は次のとおりです。

- AMODE 24 で実行されるプログラム。
- DATA(24) および RENT オプションを指定してコンパイルされたプログラム。
- NORENT オプションを使用してコンパイルされたプログラム。
- EXTERNAL ファイル (ALL31(OFF) ランタイム・オプションが指定されている場合)。ALL31(ON) ランタイム・オプションを指定するためには、実行単位内のすべてのプログラムが 31 ビット・アドレッシング・モードで実行可能でなければなりません。
- TSO 端末に割り振られるファイル。
- 形式 S (スパン) レコードを持つファイル (ファイルが以下のいずれかである場合)。
 - EXTERNAL ファイル (ALL31(ON) が指定されている場合でも)
 - I-O-CONTROL 段落の SAME RECORD AREA 節で指定されたファイル
 - I-O としてオープンされ、REWRITE ステートメントを使用して更新されるブロック化ファイル

関連概念

40 ページの『ストレージとそのアドレス可能性』

関連タスク

197 ページの『ストライプ拡張形式 QSAM データ・セットの使用』

QSAM を使用する z/OS ユニックスファイルへのアクセス

バイトストリームファイルを処理することができます。z/OS UNIX ファイルシステムとして ORGANIZATION SEQUENTIAL QSAM を使用したファイル。これを行うためには、DD 名または環境変数のいずれか 1 つを ASSIGN 節の *assignment-name* として指定してください。

このタスクについて

ddname

キーワード PATH= および FILEDATA=BINARY によってファイルを識別する DD 割り振り。

環境変数名

ファイルの z/OS UNIX ファイルシステム パスの実行時値を持つ環境変数

以下の制約事項に従ってください。

- スパン・レコード形式はサポートされません。
- OPEN I-O と REWRITE はサポートされていません。これらの操作の 1 つを行おうとすると、次のファイル状況条件の 1 つになります。
 - OPEN I-O からは 37
 - REWRITE からは 47 (ファイルを I-O として正常にオープンできなかったため)。

使用上の注意

- 次のタイプのどちらかの矛盾については、ファイル状況 39 (固定ファイル属性の矛盾) が強制されません。
 - レコード長の矛盾
 - レコード・タイプの矛盾 (固定と可変の違い)
- READ は、ファイルの最大論理レコード・サイズのバイト数を戻します (ただし、最後のレコードは別で、それより短い可能性があります)。

例えば、ファイル定義に、3、5、および 10 バイトの長さのレベル 01 レコード記述があり、3 つのレコード「abc」、「defgh」、「ijklmnopqr」をこの順に書き込むとしましょう。このファイルの最初の READ は「abcdefghij」を戻し、2 番目の READ は「klmnopqr」を戻し、3 番目の READ は AT END 条件となります。

リリースとの互換性のため IBM COBOL 以前 COBOL for OS/390 & VM2.2、指定することもできます FILEDATA=TEXT 使用する場合 DD 割り当て z/OS ユニックスファイルただし、この使用法は推奨されません。テキストファイルを処理するには z/OS UNIX ファイルシステム、使用 LINE SEQUENTIAL 組織。QSAM を使用して z/OS UNIX ファイルシステムのテキスト・ファイルを処理する場合は、環境変数を使用してファイルを定義することはできません。

関連タスク

175 ページの『ファイルの割り振り』

191 ページの『QSAM ファイルの定義および割り振り』

z/OS DFSMS: Using Data Sets (HFS データ・セットの使用)

テーブル上の QSAM ASCII ファイルの処理

プログラムが QSAM ASCII ファイルを処理する場合は、ASCII アルファベットを要求し、レコード形式を定義し、DD 名を定義する必要があります (JCL を使用)。

このタスクについて

さらに、プログラムが署名済み ASCII ファイルからの数値データ項目、数値データを、別々の符号を持つゾーン 10 進数項目として定義します。つまり、USAGE DISPLAY そして、SEPARATE のフレーズ SIGN 句。

CODEPAGE コンパイラー・オプションは、ASCII テープをサポートするための ASCII と EBCDIC 間での変換に使用されるコード・ページには影響を与えません。ASCII テープのサポートに使用される CCSID の選択方法、およびデフォルト CCSID については、z/OS DFSMS の資料を参照してください。

ASCII アルファベットの要求: SPECIAL-NAMES 段落で、ASCII を表す STANDARD-1 をコーディングします。

```
ALPHABET-NAME IS STANDARD-1
```

ファイルの FD 項目に、次のようにコーディングしてください。

```
CODE-SET IS ALPHABET-NAME
```

レコード・フォーマットの定義: QSAM ASCII テープ・ファイルは、以下のいずれかのレコード形式で処理してください。

- 固定長 (形式 F)
- 不定 (形式 U)
- 可変長 (形式 V)

可変長レコードを使用する場合、フォーマット D を明示的にコーディングすることはできません。代わりに、RECORDING MODE V。フォーマット情報は内部的に D モードに変換されます。D モード・レコードは、レコードごとに 4 バイトのレコード記述子を持ちます。

DD 名の定義: z/OS のもとでは、ASCII ファイルを処理するには、特別な JCL コーディングが必要です。DCB パラメーターの以下のサブパラメーターを、DD ステートメントにコーディングしてください。

BUFOFF=[L|n]

L

ブロック長 (ブロック接頭語を含む) が入る 4 バイトのブロック接頭語。

n

ブロック接頭語の長さ。

- 入力の場合は、0 から 99
- 出力の場合は、0 または 4

BLOCK CONTAINS 0 をコーディングした場合には、この値を使用してください。

BLKSIZE=n

n

ブロックのサイズ (ブロック接頭語の長さを含む)。

LABEL=[AL|AUL|NL]

AL

米国標準規格 (ANS) ラベル

AUL

ANS およびユーザー・ラベル

nl

ラベルなし

OPTCD=Q

Q

この値は、ASCII ファイルの場合には必須であり、ファイルが Enterprise COBOL を使用して作成された場合はデフォルトです。

関連参照

z/OS DFSMS: Using Data Sets (文字データ変換)

第 10 章 VSAM ファイルの処理

仮想記憶アクセス方式 (VSAM) は、直接アクセス・ストレージ・デバイス上のファイルに関するアクセス方式です。VSAM を使用して、ファイルのロード、ファイルからのレコードの取得、ファイルの更新、およびファイルのレコードの追加、置換、および削除を行うことができます。

このタスクについて

以下の VSAM ファイルの説明は、VSAMDB ファイルにもほぼ当てはまる。VSAMDB ファイルの詳細については、[229 ページの『第 11 章 VSAMDB ファイルの処理』](#)を参照のこと。

VSAM 処理には、QSAM と比較して以下の利点があります。

- 無許可アクセスに対するデータの保護。
- システム間の互換性。
- 装置からの独立性 (ブロック・サイズおよびその他の制御情報に留意する必要はありません)。
- より単純な JCL (システムに必要な情報は統合カタログに入れて提供されます)。
- 索引付きファイル編成または相対ファイル編成を使用する機能。

以下の表は、COBOL 用語や読者がよく知っている用語と VSAM 用語との違いを示しています。

表 22. VSAM、COBOL、および非 VSAM 用語の比較		
VSAM 用語	COBOL 用語	類似した非 VSAM 用語
データ・セット	ファイル	データ・セット
入力順データ・セット (ESDS)	順次ファイル	QSAM データ・セット
キー順データ・セット (KSDS)	索引付きファイル	ISAM データ・セット
相対レコード・データ・セット (RRDS)	相対ファイル	BDAM データ・セット
制御インターバル		ブロック
制御インターバル・サイズ (CISZ)		ブロック・サイズ
バッファ (BUFNI/BUFND)		BUFNO
アクセス方式制御ブロック (ACB)		データ制御ブロック (DCB)
クラスター (CL)		データ・セット
クラスター定義		データ・セット割り振り
JCL DD ステートメントの AMP パラメーター		JCL DD ステートメントの DCB パラメーター
レコード・サイズ		レコード長
JSON 文書	レコード	

この VSAM 資料では、ファイルという用語は、COBOL ファイルまたは VSAM データ・セットのいずれかを指します。

要件が複雑な場合、または VSAM を頻繁に使用する場合は、ご使用のオペレーティングシステムの VSAM 資料を参照してください。

関連概念

[202 ページの『VSAM ファイル』](#)

関連タスク

[203 ページの『VSAM ファイル編成およびレコードの定義』](#)

[209 ページの『VSAM ファイルの入出力ステートメントのコーディング』](#)

[217 ページの『VSAM ファイルでのエラー処理』](#)

[218 ページの『パスワードによる VSAM ファイルの保護』](#)

[219 ページの『z/OS および z/OS ユニックスのもとでの VSAM データ・セットの操作』](#)

[225 ページの『VSAM パフォーマンスの向上』](#)

関連参照

[z/OS DFSMS: Using Data Sets](#)

[データ・セットに対する z/OS DFSMS マクロ命令](#)

[z/OS DFSMS カタログのためのアクセス方式サービス・プログラム](#)

[225 ページの『VSAM ファイル用のレコード域の割り振り』](#)

[227 ページの『拡張アドレッシング機能サポート』](#)

VSAM ファイル

VSAM データ・セットの物理編成は、他のアクセス方式で使用されている編成とは大きく異なっています。

VSAM データ・セットは、制御インターバル (CI) および制御域 (CA) で保持されます。CI と CA のサイズは、通常、アクセス方式によって判別され、それらが使用される方法について意識することはありません。

VSAM では、次の 3 種類のファイル編成を使用できます。

VSAM 順次ファイル編成

(VSAM *ESDS* (入力順データ・セット) 編成とも呼ばれます。) VSAM 順次ファイル編成では、レコードは入力された順番に保管されます。

VSAM 入力順データ・セットは、QSAM 順次ファイルと同等です。レコードの順序は固定されます。

VSAM 索引付きファイル編成

(VSAM *KSDS* (キー順データ・セット) 編成とも呼ばれます。) VSAM 索引付きファイル (*KSDS*) では、レコードは、組み込み基本キー・フィールドの照合シーケンスの定義に従って配列されます。基本キーは、レコード内の 1 つ以上の連続する文字から構成されます。基本キーは、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。レコードの基本キーは、例えば、従業員番号や送り状番号にすることができます。

VSAM 相対ファイル編成

(VSAM 固定長または可変長 *RRDS* (相対レコード・データ・セット) 編成とも呼ばれます。) VSAM 相対レコード・セット (*RRDS*) には、相対キーによって順序付けされたレコードが入ります。相対キーとは、ファイルの始まりからのレコードの相対的な位置を表す相対レコード番号です。相対レコード番号は、固定長または可変長レコードを識別します。

VSAM 固定長 *RRDS* では、レコードはストレージ内の一連の固定長スロットに入れられます。各スロットは、相対レコード番号に関連付けられています。例えば、10 個のスロットが入っている固定長 *RRDS* では、最初のスロットは相対レコード番号 1 であり、10 番目のスロットは相対レコード番号 10 です。

VSAM 可変長 *RRDS* では、レコードはそれらの相対レコード番号に従って順序付けられます。レコードの保管および検索は、相対レコード番号の設定に従って行われます。

本書では、VSAM 相対レコード・データ・セット (または *RRDS*) という用語は、特に区別する必要がない限り、固定長レコードを持つ相対レコード・データ・セットと可変長レコードを持つ相対レコード・データ・セットの両方を指すために使用されています。

次の表は、種々のタイプの VSAM データ・セットの特性を比較したものです。

表 23. VSAM データ・セット・タイプの比較			
特性	入力順データ・セット (<i>ESDS</i>)	キー順データ・セット (<i>KSDS</i>)	相対レコード・データ・セッ ト (<i>RRDS</i>)
レコードの順序	書き込まれた順序	キー・フィールドによる照合 シーケンス	相対レコード番号の順序

表 23. VSAM データ・セット・タイプの比較 (続き)			
特性	入力順データ・セット (ESDS)	キー順データ・セット (KSDS)	相対レコード・データ・セット (RRDS)
アクセス	順次	索引を通してキー順	相対レコード番号順 (キーのよう処理される)
代替索引	COBOL ではサポートされませんが、1 つ以上の代替索引を持つことができます。	1 つ以上の代替索引を持つことができます。	代替索引を持つことはできません。
レコードの相対バイト・アドレス (RBA) と相対レコード番号 (RRN)	RBA を変更することはできません。	RBA を変更できます。	RRN を変更することはできません。
レコード追加用のスペース	データ・セットの終わりにあるスペースが使用されます。	レコードを挿入し、それらの長さを適切に変更するためには、分散フリー・スペースが使用されます。	固定長 RRDS の場合は、データ・セット内の空のロットが使用されます。 可変長 RRDS の場合は、分散フリー・スペースが使用され、追加されるレコードの長さは適切に変更されます。
レコード削除用のスペース	レコードは削除できませんが、そのスペースを同じ長さのレコードのために再利用することはできません。	削除または短縮されたレコードのスペースは、制御インターバルで自動的に再利用されます。	削除されたレコードのスペースを再利用することができます。
スパン・レコード	スパン・レコードを持つことができます。	スパン・レコードを持つことができます。	スパン・レコードを持つことはできません。
作業ファイルとしての再利用	代替索引を持っているか、キー範囲と関連付けられているか、またはボリューム当たりのエクステントが 123 個を超えている場合を除き、作業ファイルとして再利用できます。	代替索引を持っているか、キー範囲と関連付けられているか、またはボリューム当たりのエクステントが 123 個を超えている場合を除き、作業ファイルとして再利用できます。	再利用することができます。

関連タスク

[204 ページの『VSAM ファイルの順次編成の指定方法』](#)

[204 ページの『VSAM ファイルの索引編成の指定方法』](#)

[206 ページの『VSAM ファイルの相対編成の指定方法』](#)

[219 ページの『VSAM ファイルの定義』](#)

VSAM ファイル編成およびレコードの定義

ENVIRONMENT DIVISION の FILE-CONTROL 段落の項目を使用して、COBOL プログラムにおける VSAM ファイルのファイル編成およびアクセス・モードを定義します。

このタスクについて

の中に FILE SECTION の DATA DIVISION ファイルの説明をコード化する (FD) エントリを追加します。関連するレコード記述項目では、*record-name* およびレコード長を定義します。RECORD 節を使用して、レコードの論理サイズをコーディングしてください。

重要: Enterprise COBOL プログラムでは、アクセス方式サービス・プログラムを使用してそれらを定義してからでなければ、VSAM データ・セットを処理できません。

表 24. VSAM ファイル編成、アクセス・モード、およびレコード・フォーマット					
ファイル編成	順次アクセス	ランダム・アクセス	動的アクセス	固定長	可変長
VSAM 順次 (ESDS)	はい	いいえ	いいえ	はい	はい
VSAM 索引付き (KSDS)	はい	はい	はい	はい	はい
VSAM 相対 (RRDS)	はい	はい	はい	はい	はい

関連タスク

[204 ページの『VSAM ファイルの順次編成の指定方法』](#)

[204 ページの『VSAM ファイルの索引編成の指定方法』](#)

[206 ページの『VSAM ファイルの相対編成の指定方法』](#)

[207 ページの『VSAM ファイルのアクセス・モードの指定』](#)

[207 ページの『VSAM ファイルのレコード長の定義』](#)

[271 ページの『ファイル状況キーの使用』](#)

[273 ページの『VSAM 状況コードの使用 \(VSAM ファイルのみ\)』](#)

[219 ページの『VSAM ファイルの定義』](#)

VSAM ファイルの順次編成の指定方法

COBOL プログラム内の VSAM ESDS ファイルを識別します。ORGANIZATION IS SEQUENTIAL 句。順次ファイル内のレコードは、順次にだけしかアクセス (読み取りまたは書き込み) することができません。

このタスクについて

レコードをファイルに入れた後は、それを短くしたり、長くしたり、削除したり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。

次の例は、VSAM 順次ファイル (ESDS) の代表的な FILE-CONTROL 記入項目を示しています。

```
SELECT S-FILE
  ASSIGN TO SEQUENTIAL-AS-FILE
  ORGANIZATION IS SEQUENTIAL
  ACCESS IS SEQUENTIAL
  FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

関連概念

[202 ページの『VSAM ファイル』](#)

VSAM ファイルの索引編成の指定方法

ORGANIZATION IS INDEXED 節を使用して、COBOL プログラムの VSAM KSDS ファイルを示してください。RECORD KEY 節を使用して、レコードの基本キーをコーディングしてください。また、代替キーと代替索引を使用することができます。

このタスクについて

```
RECORD KEY IS data-name
```

上の例で、*data-name* は、DATA DIVISION のレコード記述項目で定義したときの基本キー・フィールドの名前です。主キーデータ項目のクラスは、アルファベット、英数字、DBCS、数字、国別、または UTF-8。もしそうなら USAGE NATIONAL 主キーは、国別カテゴリーにすることも、国別編集、数値編集、

国別 10 進数、または国別浮動小数点データ項目にすることもできます。レコード・キーの照合は、キーのクラスやカテゴリに関係なく、キーの 2 進値に基づいて行われます。

次の例は、動的にアクセスされる VSAM 索引付きファイル (KSDS) の場合のステートメントを示します。基本キー COMMUTER-NO のほかに、代替キー LOCATION-NO を指定します。

```
SELECT I-FILE
  ASSIGN TO INDEXED-FILE
  ORGANIZATION IS INDEXED
  ACCESS IS DYNAMIC
  RECORD KEY IS IFILE-RECORD-KEY
  ALTERNATE RECORD KEY IS IFILE-ALTREC-KEY
  FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

関連概念

[202 ページの『VSAM ファイル』](#)

関連タスク

[205 ページの『代替キーの使用』](#)

[205 ページの『代替索引の使用』](#)

関連参照

RECORD KEY 節 (*Enterprise COBOL for z/OS 言語解説書*)

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

代替キーの使用

基本キーのほかに、VSAM KSDS ファイル用の 1 つ以上の代替キーをコーディングすることができます。代替キーを使用すれば、索引付きファイルにアクセスして、基本キー順序以外の順序でレコードを読み取ることができます。

このタスクについて

代替キーは固有である必要はありません。重複してもよいように代替キーがコーディングされている場合、複数のレコードにアクセスできます。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。

COBOL プログラムで代替キーを定義するには、ALTERNATE RECORD KEY 句：

```
ALTERNATE RECORD KEY IS data-name
```

上の例で、*data-name* は、DATA DIVISION のレコード記述項目で定義したときの代替キー・フィールドの名前です。代替キーデータ項目は、主キーデータ項目と同様に、アルファベット、英数字、DBCS、数字、国別などのクラスにすることができます。または UTF-8。代替キーの照合は、キーのクラスやカテゴリに関係なく、キーの 2 進値に基づいて行われます。

代替索引の使用

VSAM KSDS ファイルで代替索引を使用するためには、アクセス方式サービスを使用して、代替索引 (AIX) と呼ばれるデータ・セットを定義する必要があります。

このタスクについて

AIX には、指定された代替キーのそれぞれの値について 1 つのレコードが入ります。レコードは、代替キー値により順次に配列されます。各レコードには、代替キー値が入っている関連索引付きファイル内のすべてのレコードの対応する基本キーが含まれます。

関連タスク

[220 ページの『代替索引の作成』](#)

VSAM ファイルの相対編成の指定方法

COBOL プログラムで VSAM RRDS ファイルを識別するには、ORGANIZATION IS RELATIVE 句。各論理レコードを相対レコード番号に関連付けるには、RELATIVE KEY IS 節を使用します。

このタスクについて

次の例は、相対キーに入れられた値によってランダムにアクセスされる相対レコード・データ・セット (RRDS) を示しています。

```
SELECT R-FILE
  ASSIGN TO RELATIVE-FILE
  ORGANIZATION IS RELATIVE
  ACCESS IS RANDOM
  RELATIVE KEY IS RFILE-RELATIVE-KEY
  FILE STATUS IS FSTAT-CODE VSAM-CODE.
```

ランダム化ルーチンを使用して、各レコード内のキー値をそのレコードの相対レコード番号と関連付けることができます。レコード・キーを相対レコード番号に変換する技法は数多くありますが、最もよく使用されるのは除算 / 剰余技法です。この技法では、キーをデータ・セット内のスロットの数で割って、商と剰余を出します。剰余に 1 を加算すると、結果は有効な相対レコード番号になります。

VSAM RRDS では、代替索引はサポートされません。

関連概念

[202 ページの『VSAM ファイル』](#)

[206 ページの『固定長および可変長 RRDS』](#)

関連タスク

[206 ページの『可変長 RRDS の使用』](#)

[219 ページの『VSAM ファイルの定義』](#)

固定長および可変長 RRDS

固定長レコードを含む RRDS では、各レコードがそれぞれ 1 つのスロットを占めます。レコードの保管と検索は、そのスロットの相対レコード番号に従って行われます。可変長 RRDS はスロットを持っていません。代わりに、定義されたフリー・スペースを使用して、より効率的なレコード挿入を行うことができます。

固定長レコードを含む RRDS をロードするときには、スロットをスキップし、それらを空にしておくこともできます。可変長レコードを含む RRDS をロードするときは、相対レコード番号をスキップすることができます。

可変長 RRDS の使用

可変長レコードを持つ相対レコードデータセット (RRDS) を使用するには、使用する必要があります VSAM 可変長 RRDS サポート。

このタスクについて

以下の手順を 実行します。

手順

1. ファイルを次のように定義します ORGANIZATION IS RELATIVE 句。
2. 可変長サイズでレコードを記述するには、FD 記入項目を使用します。
3. アクセス方式サービスを使用して、VSAM ファイルを RRDS として定義します。

タスクの結果

関連タスク

[219 ページの『VSAM ファイルの定義』](#)

関連参照

z/OS DFSMS カタログのためのアクセス方式サービス・プログラム

VSAM ファイルのアクセス・モードの指定

VSAM 順次ファイルのレコードは、順次でしかアクセスできません。VSAM 索引付きファイルおよび相対ファイルの中のレコードは、順次、ランダム、または動的の 3 つの方法でアクセスすることができます。

このタスクについて

順次アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS SEQUENTIAL をコーディングします。その場合、索引付きファイルのレコードは、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。

ランダムアクセスの場合、コード ACCESS IS RANDOM の中に FILE-CONTROL エントリ。その場合、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に従ってアクセスされます。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。

動的アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS DYNAMIC をコーディングします。動的アクセスは、同じプログラムの中での順次とランダムの混合アクセスです。動的アクセスを使用すると、順次処理とランダム処理の両方を実行する 1 つのプログラムを作成して、あるレコードは順次にアクセスし、別のレコードはキーによってアクセスすることができます。COBOL ランタイムを変更するかどうかを決定するには ACCESS IS DYNAMIC ロジックパスについては、「VSAM 動的アクセス オプション ロジックパス」の中にパフォーマンスチューニングガイド。

関連タスク

[214 ページの『VSAM ファイルからのレコードの読み取り』](#)

例: VSAM ファイルでの動的アクセスの使用

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。

プログラムが、VSAM ファイルの動的アクセスを使用して、時間給 \$15.00 から \$20.00 の従業員と \$25.00 以上の従業員を処理する場合、プログラムは以下のことを行います。

1. 1500 のキーに基づいて最初のレコードをランダムに検索します (ランダム検索 READ を使用して)。
2. 給与フィールドが 2000 を超えるまで、順次に読み取ります (READ NEXT を使用して)。
3. 2500 のキーに基づいて、次のレコードをランダム検索します。
4. ファイルの終わりになるまで、順次に読み取ります。

関連タスク

[214 ページの『VSAM ファイルからのレコードの読み取り』](#)

VSAM ファイルのレコード長の定義

VSAM レコードは、固定長または可変長に定義できます。COBOL は、RECORD 節と、ファイルの FD 記入項目に関連付けられたレコード記述から、レコード形式を判別します。

このタスクについて

VSAM ファイルについては、ブロック化の概念は意味を持たないため、BLOCK CONTAINS 節は省略して構いません。この節は構文検査されますが、プログラムの実行方法には影響を及ぼしません。

関連タスク

208 ページの『固定長レコードの定義』

208 ページの『可変長レコードの定義』

関連参照

13 ページの『FILE SECTION 記入項目』

Enterprise COBOL for z/OS 移行ガイド

固定長レコードの定義

VSAM レコードを固定長として定義するには、次のコーディング・オプションのいずれかを使用します。

このタスクについて

表 25. VSAM 可変長レコードの定義			
RECORD 節	節形式	レコード長	コメント
RECORD CONTAINS <i>integer</i> をコーディングする。	1	<i>integer-3</i> バイト の長さでサイズが固定。	ファイルに関連するレベル 01 レコード記述項目の長さは重要ではありません。
省略 RECORD 句ですが、コードはすべて level-01 ファイルに関連付けられているレコードは同じサイズとしてコード化され、OCCURS DEPENDING ON 句。		コーディングされた固定サイズ。	

関連参照

RECORD 節 (Enterprise COBOL for z/OS 言語解説書)

可変長レコードの定義

VSAM レコードを可変長として定義するには、次のコーディング・オプションのいずれかを使用します。

このタスクについて

表 26. VSAM 可変長レコードの定義			
RECORD 節	節形式	最大レコード長	コメント
コード RECORD IS VARYING FROM <i>integer-6</i> TO <i>integer-7</i> 。	3	<i>integer-7</i> バイト	ファイルに関連するレベル 01 レコード記述項目の長さは重要ではありません。
コード RECORD IS VARYING。	3	ファイルに関連する最大のレベル 01 レコード記述項目のサイズ。	コンパイラーが最大レコード長を決定します。
コード RECORD CONTAINS <i>integer-4</i> TO <i>integer-5</i> 。	2	<i>integer-5</i> バイト	最小レコード長は、 <i>integer-4</i> バイトです。
省略 RECORD 句ですが、複数のコード level-01 ファイルに関連付けられたレコードで、サイズが異なるか、OCCURS DEPENDING ON 句。		ファイルに関連する最大のレベル 01 レコード記述項目のサイズ。	コンパイラーが最大レコード長を決定します。

形式 V ファイルに `READ INTO` ステートメントを指定すると、そのファイルに関して読み取られたレコード・サイズが、コンパイラによって生成される `MOVE` ステートメントで使用されます。したがって、読み取られたレコードがレベル 01 レコード記述に対応していない場合には、予期したとおりの結果が得られないことがあります。`MOVE` ステートメントに関するその他の規則はすべて適用されます。例えば、`READ` ステートメントによって読み込まれた形式 V レコードに対して `MOVE` ステートメントを指定すると、レコードのサイズはそのレベル 01 レコード記述に相当します。

関連参照

RECORD 節 (*Enterprise COBOL for z/OS* 言語解説書)

VSAM ファイルの入出力ステートメントのコーディング

以下の COBOL ステートメントを使用して、VSAM ファイルを処理します。

このタスクについて

OPEN

VSAM データ・セットを処理のために COBOL プログラムに接続します。

WRITE

ファイルにレコードを追加するか、またはファイルをロードします。

START

`READ NEXT` ステートメントのためにクラスターにおける現在場所を設定します。

`START` はレコードを取り出しません。現行レコード・ポインターを設定するだけです。

READ および READ NEXT

ファイルからレコードを取り出します。

REWRITE

レコードを更新します。

DELETE

索引付きファイルおよび相対ファイルからのみ、レコードを論理的に除去します。

CLOSE

VSAM データ・セットをプログラムから切り離します。

以下のすべての要因によって、特定の VSAM データ・セットに使用できる入出力ステートメントが決まります。

- アクセス・モード (順次、ランダム、または動的)
- ファイル編成 (ESDS、KSDS、または RRDS)
- OPEN ステートメントのモード (INPUT、OUTPUT、I-O、または EXTEND)

次の表は、順次ファイル (ESDS) のステートメントとオープン・モードの可能な組み合わせを示しています。X は、列の上部に示されているオープン・モードでステートメントを使用できることを示します。

表 27. VSAM 順次ファイル用入出力ステートメント					
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

次の表は、索引付き (KSDS) ファイルおよび相対 (RRDS) ファイルに関して使用できるステートメントとオープン・モードの可能な組み合わせを示しています。X は、列の上部に示されているオープン・モードでステートメントを使用できることを示します。

表 28. VSAM 相対ファイルおよび索引付きファイル用入出力ステートメント					
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	
動的	OPEN	X	X	X	
	WRITE		X	X	
	START	X		X	
	READ	X		X	
	REWRITE			X	
	DELETE			X	
	CLOSE	X	X	X	

FILE STATUS 節でコーディングしたフィールドは VSAM によって、それぞれの入出力ステートメントの後で、操作の成功または失敗を示すために更新されます。

関連概念

[211 ページの『ファイル位置標識』](#)

関連タスク

[211 ページの『ファイルのオープン \(ESDS、KSDS、または RRDS\)』](#)

[214 ページの『VSAM ファイルからのレコードの読み取り』](#)

[215 ページの『VSAM ファイル内のレコードの更新』](#)

[216 ページの『VSAM ファイルへのレコードの追加』](#)

[216 ページの『VSAM ファイル内のレコードの置換』](#)

[217 ページの『VSAM ファイルからのレコードの削除』](#)

[217 ページの『VSAM ファイルのクローズ』](#)

関連参照

ファイル状況キー (*Enterprise COBOL for z/OS* 言語解説書)

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、および READ NEXT ステートメントによって設定されます。

その後の READ または READ NEXT 要求は、設定されたファイル位置標識の位置を使用し、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。

関連タスク

[214 ページの『VSAM ファイルからのレコードの読み取り』](#)

ファイルのオープン (ESDS、KSDS、または RRDS)

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。

このタスクについて

ファイルが使用可能であるのか、またはオプションであるのかによって、OPEN の処理、ファイルの作成、および結果のファイル状況キーに影響があります。例えば、EXTEND、I-O、または INPUT モードで、存在しない OPTIONAL 以外のファイルを開くと、結果は OPEN エラーになり、ファイル状況 35 が返されます。ただし、ファイルが OPTIONAL である場合、同じ OPEN ステートメントでファイル状況 05 が返され、オープン・モード EXTEND および I-O ではファイルが作成されます。

OPEN 操作が正しく機能するのは、ファイルの DD ステートメントまたはデータ・セット・ラベルで固定ファイル属性を指定し、しかも COBOL プログラムの SELECT 節および FD 記入項目でそのファイルについて一貫性のある属性を指定する場合だけです。以下の項目が一致していないと、ファイル状況キー 39 が戻され、OPEN ステートメントは失敗に終わります。

- ファイル編成の属性 (順次、相対、または索引付き)
- 基本レコード・キー
- 代替レコード・キー
- 最大レコード・サイズ
- レコード・タイプ (固定長または可変長)

VSAM ファイルについての OPEN ステートメントをコーディングする方法は、ファイルが空である (レコードが入ったことがない) か、ロード済みであるかによって異なります。ファイルのタイプがどちらであって

も、プログラムでは、それぞれの OPEN ステートメントの後でファイル状況キーを検査しなければなりません。

注: VSAMOPENFS オプションは、VSAM ファイルに対して正常に実行された OPEN ステートメントから報告されたファイル状況キーに作用します。VSAMOPENFS (COMPAT) オプションが有効になっている場合に、VSAM OPEN ステートメントが正常に検証されると、状況値 97 を受け取ります。VSAMOPENFS オプションについて詳しくは、[461 ページの『VSAMOPENFS』](#)を参照してください。

関連タスク

[212 ページの『空のファイルのオープン』](#)

[214 ページの『ロード済みファイル\(レコードが入っているファイル\)のオープン』](#)

関連参照

[213 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』](#)

[461 ページの『VSAMOPENFS』](#)

空のファイルのオープン

レコードをまったく含んでいないファイル(空ファイル)を開くには、OPEN ステートメントの形式を使用してください。

このタスクについて

開こうとするファイルのタイプに応じて、次のいずれかのステートメントを使用してください。

- ESDS ファイルの場合は、OPEN OUTPUT。
- KSDS および RRDS ファイルの場合は、OPEN OUTPUT または OPEN EXTEND。(どちらのコーディングでも効果は同じです。) ファイルをランダム・アクセスまたは動的アクセス用にコーディングし、それがオプション・ファイルであれば、OPEN I-O を使用することができます。

オプション・ファイルとは、プログラムが実行されるたびに、必ずしも使用可能である必要はないファイルです。INPUT、I-O、または OUTPUT モードで開かれるファイルは、FILE-CONTROL 段落の SELECT OPTIONAL 節で定義することにより、オプションとして定義できます。

順次的なファイルの初期ロード: ファイルの初期ロードとは、レコードを初めてファイルに書き込むことを意味します。これは、以前のすべてのレコードが削除されたファイルにレコードを書き込むことと同じではありません。VSAM ファイルを初期ロードするには、以下のようしてください。

手順

1. ファイルをオープンします。
2. 順次処理を行います (ACCESS IS SEQUENTIAL)。(順次処理は、ランダム処理や動的処理よりも速く行われます。)
3. WRITE を使用して、ファイルにレコードを加えます。

タスクの結果

OPEN OUTPUT を使用して VSAM ファイルをロードすると、プログラムのパフォーマンスが著しく向上します。使用 OPEN I-O または OPEN EXTEND プログラムのパフォーマンスに悪影響を及ぼします。

VSAM 索引付きファイルを順次にロードすると、順次処理ではユーザー定義のフリー・スペースが保持されるため、ロードとその後の処理の両方のパフォーマンスが最適化されます。将来の追加がより効率的になります。

と ACCESS IS SEQUENTIAL 記録は昇順で書く必要があります RECORD KEY 注文。

VSAM 相対ファイルを順次にロードすると、レコードは相対レコード番号の昇順にファイルに入れられます。

ランダムまたは動的なファイルの初期ロード: ランダム処理または動的処理でファイルをロードできますが、これらの処理は順次処理ほど効率的ではありません。VSAM はランダム処理や動的処理をサポート

していないため、COBOL は、VSAM を使用するためにいくつかの追加処理を実行する必要があります。
ACCESS IS RANDOM または ACCESS IS DYNAMIC と OPEN OUTPUT または OPEN I-O。これらのステップにより、ファイルの使用準備が整い、ファイルはロード済みファイル状況になります (少なくとも一度使用されたことがあるからです)。

ファイルが使用できるよう準備するための余分なオーバーヘッドに加えて、ランダム処理ではユーザー定義のフリー・スペースが考慮されません。その結果、将来の追加処理が非効率的になる恐れがあります。順序処理では、ユーザー定義のフリー・スペースが保持されます。

拡張フォーマットの VSAM データセットをロードする場合、ファイルステータス 30 は、OPEN もし z/OS DFSMS システム管理バッファリングは、バッファリングをローカル共有リソース (LSR) に設定します。この場合、VSAM データセットを正常にロードするには、次のように指定します。ACCBIAS=USER の中に DD AMP システム管理のバッファリングをバイパスするための VSAM データセットのパラメーター。

アクセス方式サービス・プログラムによる VSAM データ・セットのロード: アクセス方式サービス・プログラムの REPRO コマンドを使用して VSAM データ・セットをロードまたは更新することができます。可能なときはいつでも、REPRO を使用してください。

関連タスク

[214 ページの『ロード済みファイル \(レコードが入っているファイル\) のオープン』](#)

関連参照

[213 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』](#)
z/OS DFSMS カタログのためのアクセス方式サービス・プログラム (REPRO)

VSAM ファイルにレコードをロードするために使用されるステートメント

レコードを VSAM ファイルにロードするには、以下に示すステートメントを使用してください。

表 29. VSAM ファイルにレコードをロードするためのステートメント			
除算	ESDS	KSDS	RRDS
ENVIRONMENT DIVISION	SELECT ASSIGN FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS INDEXED RECORD KEY ALTERNATE RECORD KEY FILE STATUS PASSWORD ACCESS MODE	SELECT ASSIGN ORGANIZATION IS RELATIVE RELATIVE KEY FILE STATUS PASSWORD ACCESS MODE
DATA DIVISION	FD 記入項目	FD 記入項目	FD 記入項目
PROCEDURE DIVISION	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE	OPEN OUTPUT OPEN EXTEND WRITE CLOSE

関連タスク

[212 ページの『空のファイルのオープン』](#)
[215 ページの『VSAM ファイル内のレコードの更新』](#)

ロード済みファイル(レコードが入っているファイル)のオープン

すでにレコードが含まれているファイルを開くには、OPEN INPUT、OPEN I-O、または OPEN EXTEND。

このタスクについて

VSAM 入力順または相対レコード・ファイルを EXTEND としてオープンする場合には、追加されたレコードは、ファイル内の最後の既存のレコードの後に置かれます。

VSAM キー順ファイルを 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・キーを持っていないければなりません。

関連タスク

212 ページの『空のファイルのオープン』

219 ページの『z/OS および z/OS ユニックス のもとの VSAM データ・セットの操作』

関連参照

213 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』

z/OS DFSMS カタログのためのアクセス方式サービス・プログラム

VSAM ファイルからのレコードの読み取り

ファイルからレコードを検索 (READ) するには、READ ステートメントを使用します。レコードを読み取るためには、ファイルを INPUT または I-O としてオープンしていなければなりません。プログラムでは、それぞれの READ の後でファイル状況キーを検査しなければなりません。

このタスクについて

VSAM 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。

VSAM 索引付きファイルおよび相対レコード・ファイルの中のレコードは、次のように検索することができます。

順次

使用しているキーの昇順に応じて、RECORD KEY または ALTERNATE RECORD KEY インデックスファイルの場合はファイル位置インジケータの現在の位置から始まり、相対ファイルの場合は相対レコード位置の昇順に従って始まります。

ランダム

設定に応じて任意の順序で RECORD KEY または ALTERNATE RECORD KEY または RELATIVE KEY あなたの前に READ リクエスト

動的

順次とランダムの混合で

動的アクセスでは、特定のレコードを直接読み取るか、レコードを順番に読み取るかを切り替えることができます。READ NEXT 順次検索と READ ランダム検索 (キーによる) 用。

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 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。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になります。

```
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、VSAM 索引付きファイルに対して直接 READ が実行されると、その代替キーを持つデータ・セット (基本クラスター) の最初のレコードのみが検索されます。一連の READ

NEXT 同じ代替キーを持つ各データセットレコードを取得するためのステートメント。同じ代替キー値を持つ、読み取るべきレコードがほかにある場合には、02 のファイル状況コードが戻されます。そのキー値を持つ最後のレコードが読み取られると、00 のコードが戻されます。

関連概念

211 ページの『ファイル位置標識』

関連タスク

207 ページの『VSAM ファイルのアクセス・モードの指定』

VSAM ファイル内のレコードの更新

VSAM ファイルを更新するには、これらの PROCEDURE DIVISION ステートメントを使用します。

このタスクについて

表 30. VSAM ファイル内のレコードを更新するためのステートメント			
アクセス方式	ESDS	KSDS	RRDS
ACCESS IS SEQUENTIAL	OPEN EXTEND WRITE CLOSE または OPEN I-O READ REWRITE CLOSE	OPEN EXTEND WRITE CLOSE または OPEN I-O READ REWRITE DELETE CLOSE	OPEN EXTEND WRITE CLOSE または OPEN I-O READ REWRITE DELETE CLOSE
ACCESS IS RANDOM	該当なし	OPEN I-O READ WRITE REWRITE DELETE CLOSE	OPEN I-O READ WRITE REWRITE DELETE CLOSE
ACCESS IS DYNAMIC (順次処理)	該当なし	OPEN I-O READ NEXT WRITE REWRITE START DELETE CLOSE	OPEN I-O READ NEXT WRITE REWRITE START DELETE CLOSE

表 30. VSAM ファイル内のレコードを更新するためのステートメント (続き)

アクセス方式	ESDS	KSDS	RRDS
ACCESS IS DYNAMIC (ランダム処理)	該当なし	OPEN I-O READ WRITE REWRITE DELETE CLOSE	OPEN I-O READ WRITE REWRITE DELETE CLOSE

関連参照

213 ページの『VSAM ファイルにレコードをロードするために使用されるステートメント』

VSAM ファイルへのレコードの追加

COBOL WRITE ステートメントを使用すると、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することができます。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・サイズよりも大きいものであってはなりません。プログラムでは、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なければなりません。

このタスクについて

順次のレコード追加: OUTPUT または EXTEND としてオープンされた VSAM 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。

順次ファイルは、必ず順次に書き込まれます。

索引付きファイルの場合は、昇順キー配列で新規のレコードを書かなければなりません。ファイルが EXTEND としてオープンされている場合には、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。もし、RELATIVE KEY データ項目 SELECT 句では、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配置されます。

レコードをランダムまたは動的に追加する: インデックス付きデータセットにレコードを書き込むと、ACCESS IS RANDOM または ACCESS IS DYNAMIC、レコードを任意の順序で書き込むことができます。

VSAM ファイル内のレコードの置換

VSAM 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 として開かれていない場合には、レコードは再書き込みされず、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。

このタスクについて

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は、相対レコード番号を RELATIVE KEY データ項目を入力し、REWRITE。

VSAM 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。順次ファイルに対して DELETE を使用することはできません。

このタスクについて

ACCESS IS SEQUENTIAL を使用する場合、またはファイルにスパン・レコードが含まれている場合には、まず削除すべきレコードをプログラムで読み取らなければなりません。その後、読み取られたレコードを DELETE で除去します。DELETE の前の READ が成功しなかった場合には、削除は行われず、状況キー値が 92 に設定されます。

ACCESS IS RANDOM または ACCESS IS DYNAMIC を使用する場合は、削除すべきレコードを前もってプログラムで読み取る必要はありません。レコードを削除するには、削除するレコードのキーを RECORD KEY データ項目を入力し、DELETE。プログラムでは、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なければなりません。

VSAM ファイルのクローズ

プログラムを VSAM ファイルから切り離すには、CLOSE ステートメントを使用します。既にクローズされているファイルをクローズしようとする、論理エラーになります。それぞれの CLOSE 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。

このタスクについて

VSAM ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。

- 実行単位が正常に終了すると、その実行単位内の COBOL プログラムのいずれかで定義されている、オープン状態のすべてのファイルはクローズされます。
- 実行単位が異常終了した場合は、TRAP (ON) ランタイム・オプションが設定されていれば、その実行単位内の COBOL プログラムのいずれかで定義されているオープン状態のファイルはすべてクローズされます。
- 言語環境条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが再開すると、実行単位内の COBOL プログラムのうち、再度呼び出されて再入される可能性のあるプログラムに定義されているオープン・ファイルはクローズされます。

条件が処理された後でプログラムが実行を再開する位置を変更することができます。この変更を行うには、例えば、CEEMRCR 呼び出し可能サービスを使用して再開カーソルを移動するか、または C longjmp ステートメントのような言語構造体を使用してください。

- COBOL サブプログラムに CANCEL を出すと、そのプログラム内で定義されているオープン状態の非外部ファイルがすべてクローズされます。
- INITIAL 属性を持つ COBOL サブプログラムが制御権を戻すと、そのプログラム内で定義されているオープン状態の非外部ファイルがすべてクローズされます。
- マルチスレッド・アプリケーションのスレッドが終了すると、同じスレッド内部からオープンした外部ファイルと非外部ファイルはいずれもクローズされます。

このような暗黙 CLOSE 操作の実行時には DATA DIVISION 内のファイル状況キー・データ項目が設定されますが、EXCEPTION/ERROR 宣言は呼び出されません。

エラー: マルチスレッド・アプリケーションで VSAM ファイルをオープンした場合は、ファイルをオープンしたのと同じ実行スレッドからクローズする必要があります。異なるスレッドからファイルをクローズしようすると、ファイル状況コードの条件 90 でクローズが失敗します。

VSAM ファイルでのエラー処理

入出力ステートメントの操作が失敗しても、COBOL がユーザーに代わって訂正処置を行うことはありません。

このタスクについて

全て OPEN そして CLOSE VSAM ファイルでエラーが発生した場合、プログラム内の論理エラーか外部記憶媒体の入出力エラーかにかかわらず、何もコーディングしていなくても COBOL プログラムに制御を戻します。DECLARATIVE そして、いいえ FILE STATUS 句。

他の入出力ステートメントの操作の失敗については、重大レベルより軽度のエラーの後でプログラムに実行を継続させるかどうかを選択します。

COBOL は、特定の VSAM 入出力エラーを代行受信して処理するために、以下の方法を提供します。

- ファイル終了句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 節 (ファイル状況キーおよび VSAM 状況コード)
- INVALID KEY 句

プログラム内に定義するそれぞれの VSAM ファイルごとに状況キーを定義しなければなりません。それぞれの入力要求または出力要求 (特に、OPEN および CLOSE) の後で、状況キーの値を検査してください。

ファイルステータスキーまたは宣言をコーディングしないと、重大な VSAM 処理エラーが発生し、メッセージが発行され、言語環境シグナル状態になり、ランタイムオプションを指定すると異常終了が発生します。ABTERMENC (ABEND)。

関連タスク

267 ページの『入出力操作でのエラーの処理』

273 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』

関連参照

データ・セットに対する z/OS DFSMS マクロ命令 (VSAM マクロの戻りコードおよび理由コード)

パスワードによる VSAM ファイルの保護

z/OS システムで推奨するセキュリティー機構は RACF ですが、Enterprise COBOL では、無許可アクセスや無許可更新を防止するため、VSAM ファイルに明示パスワードを使用することもサポートします。

このタスクについて

明示パスワードを使用するには、FILE-CONTROL 段落で PASSWORD 節をコーディングしてください。この節は、ファイルのカタログ記入項目に読み取りまたは更新パスワードが入っている場合にのみ使用してください。

- カタログエントリに読み取りパスワードが含まれている場合、PASSWORD 条項の FILE-CONTROL 段落とそれを記述する DATA DIVISION。ファイルがオープンされるときには、参照された *data-name* に有効なパスワードが入っていないなければなりません。
- カタログ記入項目に更新パスワードが入っている場合には、ファイルはオープンおよびアクセスすることはできませんが、PASSWORD 節が FILE-CONTROL 段落で使用され、DATA DIVISION で記述されていない限り、更新できません。
- カタログ記入項目に読み取りパスワードと更新パスワードの両方が入っている場合には、プログラム内でファイルを読み取るときと更新するときの両方に更新パスワードを使用してください。

プログラムでレコードの検索だけを行い、更新を行わない場合には、読み取りパスワードしか必要ありません。プログラムでファイルをロードまたは更新する場合には、カタログされた更新パスワードを指定しなければなりません。

索引付きファイルの場合は、RECORD KEY の PASSWORD データ項目に有効なパスワードが含まれていなければ、ファイルを正常にオープンすることはできません。

VSAM 索引付きファイルをパスワード保護する場合、完全にパスワード保護するためには、すべての代替索引もパスワード保護しなければなりません。それぞれの代替索引はそれ自体のパスワードを持つため、

PASSWORD 節を置く場所が重要になります。PASSWORD 節は、それが適用されるキー節の直後になければなりません。

例: VSAM 索引付きファイルのパスワード保護

以下に、パスワード保護のある VSAM 索引付きファイルについて使用される COBOL コードの例を示します。

```
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
  SELECT LIBFILE  
  ASSIGN TO PAYMAST  
  ORGANIZATION IS INDEXED  
  RECORD KEY IS EMPL-NUM  
  PASSWORD IS BASE-PASS  
  ALTERNATE RECORD KEY IS EMPL-PHONE  
  PASSWORD IS PATH1-PASS  
  
WORKING-STORAGE SECTION.  
01 BASE-PASS          PIC X(8) VALUE "25BSREAD".  
01 PATH1-PASS         PIC X(8) VALUE "25ATREAD".
```

z/OS および z/OS ユニックスのもとでの VSAM データ・セットの操作

VSAM ファイルを z/OS および z/OS ユニックスのもとで使用する場合、アクセス方式サービス・プログラム (IDCAMS) コマンド、環境変数、および JCL (ジョブ制御言語) に関していくつかの特別なコーディング上の考慮事項に注意してください。

このタスクについて

VSAM ファイルが使用可能となるのは、以下のすべての条件が真である場合です。

- VSAM ファイルが、アクセス方式サービス・プログラムを使用して定義されている。
- DD ステートメント、環境変数、または ALLOCATE コマンドを指定することによって、プログラムで VSAM ファイルを定義している。
- 既にレコードが入っている。

VSAM ファイルは、定義されていても、レコードが入ったことがなければ、利用不能です。

VSAM 順次ファイルに対する OPEN ステートメントの完了時には、必ず戻りコード 0 が戻されます。

ファイルを空にするには、アクセス方式サービス・プログラム REPRO コマンドを使用してください。この方法でレコードを削除すると、ファイルの最も高い相対バイト・アドレス (RBA) が 0 にリセットされます。ファイルは事実上空になり、COBOL にとってはそれがレコードを含んだことがないかのように見えます。

関連タスク

[9 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定義する』](#)

[219 ページの『VSAM ファイルの定義』](#)

[220 ページの『代替索引の作成』](#)

[222 ページの『VSAM ファイルの割り振り』](#)

[223 ページの『RLS による VSAM ファイルの共用』](#)

VSAM ファイルの定義

Enterprise COBOL の VSAM 入力順、キー順、相対レコードの各データ・セットは、アクセス方式サービス・プログラム (IDCAMS) を使用してそれらを定義してからでなければ処理できません。

このタスクについて

VSAM クラスター とは、VSAM データ・セットの論理定義であり、1 つまたは 2 つのコンポーネントを持っています。

- VSAM クラスターのデータ・コンポーネントには、データ・レコードが入れられます。
- VSAM キー順クラスターの索引コンポーネントは、索引レコードから構成されます。

VSAM データ・セット (クラスター) を定義するには、`DEFINE CLUSTER` アクセス方式サービス・コマンドを使用してください。このプロセスには、データ転送を行わずに統合カタログ内に項目を作成することが含まれます。クラスターについては、以下の情報を定義してください。

- 記入項目の名前。
- この定義およびそのパスワードを入れるカタログの名前 (デフォルト名を使用できます)。
- 編成 (順次、索引付き、または相対)。
- データ・セットが占有する装置およびボリューム。
- データ・セットに必要なスペース。
- レコード・サイズおよび制御インターバル・サイズ (CISIZE)。
- 将来のアクセスに必要なパスワード (もしあれば)。

クラスター内のデータ・セットの種類によっては、クラスターごとに以下の情報も定義してください。

- VSAM 索引付きデータ・セット (KSDS) の場合は、レコード内の基本キーの長さおよび位置を指定してください。
- VSAM 固定長相対レコード・データ・セット (RRDS) の場合は、最大の COBOL レコード・サイズより大か等しいレコード・サイズを指定してください。

```
DEFINE CLUSTER NUMBERED
RECORDSIZE(n,n)
```

このようにデータ・セットを定義した場合、すべてのレコードは、固定スロット・サイズ *n* まで埋め込まれます。を使用する場合 `RECORD IS VARYING ON` データ名の形態 `RECORD` 条項、`WRITE` または `REWRITE` 指定された長さを使用する `DEPENDING ON` データ名 VSAM によって転送されるレコードの長さとして。このデータは、その後、固定スロット・サイズに埋め込まれます。 `READ` ステートメントは必ず、固定スロット・サイズを `DEPENDING ON data-name` で戻します。

- VSAM 可変長相対レコード・データ・セット (RRDS) の場合は、予期される平均サイズ COBOL レコードと予期される最大サイズ COBOL レコードを指定してください。

```
DEFINE CLUSTER NUMBERED
RECORDSIZE(avg,m)
```

予期される平均サイズ COBOL レコードは、予期される最大サイズ COBOL レコードより小さくなければなりません。

関連タスク

[220 ページの『代替索引の作成』](#)

[222 ページの『VSAM ファイルの割り振り』](#)

[206 ページの『VSAM ファイルの相対編成の指定方法』](#)

関連参照

[z/OS DFSMS カタログのためのアクセス方式サービス・プログラム](#)

代替索引の作成

代替索引は、複数のキーを使用するデータ・セット内のレコードへのアクセスを提供するものです。索引付きデータ・セット (KSDS) の基本索引キーと同じ方法でレコードにアクセスします。

このタスクについて

代替索引の使用を計画している場合には、以下のことを知っておかなければなりません。

- 索引が関連付けられるデータ・セット (基本クラスター) のタイプ。

- キーが固有であるか固有でないか。
- 索引をパスワード保護するかどうか。
- 代替索引使用のパフォーマンスの側面。

代替索引は、実際には、VSAM データ・セットのキーへのポインターを含む VSAM データ・セットであるため、代替索引および代替索引パス (代替索引と基本索引の間の関係を確立するエンティティ) を定義しなければなりません。代替索引を定義した後、代替索引とその基本クラスターの間の関係 (パス) を確立するカタログ記入項目を作成してください。このパスにより、代替キーを介して基本クラスターのレコードにアクセスすることが可能になります。

代替索引を使用するには、以下のステップに従ってください。

手順

1. 代替インデックスを定義するには、`DEFINE ALTERNATEINDEX` 指示。このコマンドの中では、以下の項目を指定します。

- 代替索引の名前。
- 関連する VSAM 索引付きデータ・セットの名前。
- 代替索引のレコードにおける位置と、代替索引が固有であるかどうか。
- データ・セットの変更時に代替索引を更新するかどうか。
- この定義およびそのパスワードを入れるカタログの名前 (デフォルト名を使用できます)。

COBOL プログラムでは、代替インデックスは `ALTERNATE RECORD KEY` 条項の `FILE-CONTROL` 段落。 `ALTERNATE RECORD KEY` 定義は、カタログ記入項目の定義と一致していなければなりません。カタログしたパスワード項目は、`ALTERNATE RECORD KEY` 句の後に直接コーディングしなければなりません。

2. 代替インデックスをベースクラスター (代替インデックスがアクセスを許可するデータセット) に関連付けるには、`DEFINE PATH` 指示。このコマンドの中では、以下の項目を指定します。

- パスの名前。
- パスが関連付けられている代替索引。
- 代替索引を含むカタログの名前。

基本クラスターと代替索引は、同じカタログ内の項目によって記述されます。

3. VSAM 索引付きデータ・セットをロードします。
4. `BLDINDEX` コマンドを使用して (一般に)、代替索引を作成します。入力ファイルを索引付きデータ・セット (基本クラスター) として識別し、出力ファイルを代替索引またはそのパスとして識別します。
`BLDINDEX` は、VSAM 索引付きデータ・セット (つまり、基本クラスター) のすべてのレコードを読み取り、代替索引を作成するのに必要なデータを取り出します。

あるいは、ランタイム・オプション `AIXBLD` を使用して、実行時に代替索引を作成することもできます。しかし、これはパフォーマンスに悪影響を及ぼすことがあります。

タスクの結果

221 ページの『[例: 代替索引の項目](#)』

関連タスク

205 ページの『[代替索引の使用](#)』

関連参照

言語環境プログラミング・リファレンス (AIXBLD (COBOL のみ))

例: 代替索引の項目

次の例では、COBOL `FILE-CONTROL` 記入項目と、2 つの代替索引を持つ VSAM 索引付きファイルに対する DD ステートメントまたは環境変数の関係を示しています。

JCL を使用する場合:

```
//MAINA      DD      DSN=clustername,DISP=OLD      (1)
//MAINA1     DD      DSN=path1,DISP=OLD            (2)
//MAINA2     DD      DSN=path2,DISP=OLD            (3)
```

環境変数を使用する場合:

```
export MAINA=DSN(clustername),OLD                  (1)
export MAINA=DSN(path1),OLD                         (2)
export MAINA=DSN(path2),OLD                         (3)
FILE-CONTROL.
  SELECT MAIN-FILE ASSIGN TO MAINA                  (4)
  RECORD KEY IS EM-NAME
  PASSWORD IS PW-BASE                               (5)
  ALTERNATE RECORD KEY IS EM-PHONE                  (6)
  PASSWORD IS PW-PATH1
  ALTERNATE RECORD KEY IS EM-CITY                    (7)
  PASSWORD IS PW-PATH2.
```

- (1) 基本クラスター名は *clustername* です。
- (2) 最初の代替索引パスの名前は *path1* です。
- (3) 2 番目の代替索引パスの名前は *path2* です。
- (4) 基本クラスターの DD 名または環境変数名は、ASSIGN 節で指定されます。
- (5) パスワードはその索引の直後に続きます。
- (6) キー EM-PHONE は、最初の代替索引に関連します。
- (7) キー EM-CITY は、2 番目の代替索引に関連します。

関連タスク

[220 ページの『代替索引の作成』](#)

VSAM ファイルの割り振り

VSAM データ・セットはすべて、アクセス方式サービス・プログラム DEFINE コマンドを介して事前定義し、カタログしなければなりません。VSAM データ・セットに関する情報の大部分は、カタログに入っているため、最小限の DD または環境変数情報のみを指定するだけで構いません。

このタスクについて

VSAM ファイル (索引付き、相対、および順次) の割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。

環境変数を使用して VSAM ファイルを割り当てる場合、変数名は大文字でなければなりません。通常、関係がある変数は、入力およびデータ・バッファだけです。以下のオプションを示された順番で指定する必要があります。他のオプションを指定してはなりません。

1. DSN(*dsname*)。ここで、*dsname* は基本クラスターの名前です。
2. OLD または SHR

VSAM ファイルおよび対応する export コマンドに必要な基本 DD ステートメントは、次のとおりです。

```
//ddname DD DSN=dsname,DISP=SHR,AMP=AMORG
export evname="DSN(dsname),SHR"
```

どちらの場合にも、*dsname* アクセス方式サービスで使用する名前と同じである必要があります DEFINE CLUSTER または DEFINE PATH 指示。データ・セットは既にカタログされているため、DISP は OLD または SHR でなければなりません。JCL を使用する際に MOD を指定すると、そのデータ・セットは OLD として扱われます。

AMP は、データ・セットについてプログラムによって提供される情報を補足する VSAM JCL パラメーターです。AMP が有効になるのは、プログラムが VSAM ファイルをオープンしたときです。AMP パラメーターを介して設定されたパラメーターは、カタログ内の情報またはプログラムによって提供される情報に優先します。AMP パラメーターは、次の場合にのみ必要です。

- ・ダミー VSAM データ・セットを使用する場合。例えば、

```
//ddname DD DUMMY,AMP=AMORG
```

- ・追加の索引またはデータ・バッファーを要求する場合。例えば、

```
//ddname DD DSN=VSAM.dsname,DISP=SHR,
// AMP=('BUFNI=4,BUFND=8')
```

環境変数を使用して VSAM データ・セットを割り振る場合は、AMP を指定することはできません。

VSAM 基本クラスターの場合は、SELECT 節の後の ASSIGN 節で指定した名前と同じシステム名 (DD 名または環境変数名) を指定してください。

COBOL プログラムで代替索引を使用する場合は、基本クラスターのシステム名 (DD ステートメントまたは環境変数) だけでなく、各代替索引パスごとのシステム名も指定する必要があります。プログラム内で代替索引パスのシステム名を明示的に宣言するための言語メカニズムはありません。このため、それぞれの代替索引パスのシステム名 (DD 名または環境変数名) を以下のガイドラインに従って作成する必要があります。

- ・基本クラスター名に整数を連結します。
- ・プログラム内のファイルに定義されている最初の代替レコードに関連付けられたパスの 1 から始まります (ALTERNATE RECORD KEY 条項の FILE-CONTROL 段落)。
- ・そのファイルのその後の代替レコード定義と関連付けられたパスごとに 1 ずつ増やします。

例えば、基本クラスターのシステム名が ABCD である場合、プログラム内のファイルに定義された最初の代替索引パスは ABCD1、2 番目の代替索引パスのシステム名は ABCD2 となり、以後同様に続きます。

基本クラスター・システム名とシーケンス番号を合わせた長さが 8 文字を超える場合は、システム名の基本クラスター部分の右側が切り捨てられ、連結結果が 8 文字になるよう切り詰められます。例えば、基本クラスターのシステム名が ABCDEFGH である場合、最初の代替索引パスのシステム名は ABCDEFG1、10 番目の代替索引パスのシステム名は ABCDEF10 となり、以後同様に続きます。

関連タスク

[175 ページの『ファイルの割り振り』](#)

関連参照

[MVS プログラム管理: ユーザーズ・ガイドおよび解説書](#)

RLS による VSAM ファイルの共用

VSAM JCL パラメーター RLS を使用することによって、VSAM とのレコード・レベル共用を指定することができます。RLS を指定することは、COBOL プログラム実行時に RLS モードを要求する唯一の方法です。

このタスクについて

整合性を保つ読み取りプロトコルが必要なときは RLS=CR を使用し、読み取り保全性プロトコルが必要でないときは RLS=NRI を使用してください。環境変数を使用して VSAM データ・セットを割り振る場合は、RLS を指定することはできません。

関連タスク

[224 ページの『RLS モードでの VSAM ファイルに関する更新問題の回避』](#)

[225 ページの『RLS モードでの VSAM ファイルのエラーの処理』](#)

関連参照

[224 ページの『RLS を使用する際の制約事項』](#)

RLS モードでの VSAM ファイルに関する更新問題の回避

VSAM データ・セットが I-O (更新) 用に RLS モードで開かれるとき、指定されている RLS の値 (RLS=CR または RLS=NRI) にかかわらず、最初の READ によりレコードの排他ロックが行われます。

このタスクについて

COBOL ファイルが ACCESS RANDOM と定義されている場合、VSAM は、WRITE ステートメントまたは REWRITE ステートメントの実行後、あるいは別のレコードに対する READ ステートメントの実行後に、レコードに対する排他ロックを解除します。WRITE または REWRITE の処理が実行されると、VSAM は即時にレコードを書き出します。

しかし、COBOL ファイルが ACCESS DYNAMIC として定義されている場合、VSAM は、WRITE や REWRITE ステートメントの後でも、READ ステートメントの後でもレコードの排他ロックを解放しません。ただし、I-O ステートメントにより VSAM が別の制御インターバル (CI) へ移動させられる場合には解放されます。この結果、WRITE または REWRITE が実行されても、処理が別の CI に移動してロックが解除されるまで、VSAM はレコードを書き込みません。ACCESS DYNAMIC を使用する際に、レコードの即時の書き出しや排他ロックの即時の解除 (あるいはその両方) を実現する 1 つの方法は、CI ごとに 1 レコードのみを許容する VSAM データ・セットを定義することです。

RLS=CR を指定すると、レコードはロックされ、別のレコードに対する別の READ が要求されるまで、そのレコードへの更新を妨げることができます。読み取り中のレコードに関するロックが有効な間、他のユーザーは同じレコードに対する READ を要求することはできませんが、読み取りロックが解除されるまで、そのレコードを更新することはできません。RLS=NRI を指定した場合、入力 of READ の実行時にはロックは有効になりません。別のユーザーがそのレコードを更新する可能性があります。

RLS=CR のロッキング規則により、アプリケーションはレコード・ロックが使用可能になるのを待つ可能性があります。この待機によって、入力 of READ がスローダウンことがあります。RLS=CR を使用する場合には、アプリケーションのロジックを修正することが必要な場合があります。アプリケーションが複数の更新環境で正しく機能することを確認するまでは、回復不能範囲を更新するバッチ・ジョブには、RLS パラメーターを使用しないでください。

VSAM データ・セットを RLS モードで INPUT または I-O 処理用にオープンするときには、OPEN または START を出すのは READ の直前が良いでしょう。OPEN または START と READ の間に遅延があると、OPEN または START の実行後に、アプリケーションが位置を定めたレコードの前の位置に他のユーザーがレコードを追加する可能性が高くなります。COBOL 実行時には、OPEN が要求された時点で VSAM データ・セットの開始位置が明示的に指し示されますが、READ が遅れると、他のユーザーによって追加されたレコードによって、VSAM データ・セットの実際の開始位置が変わる可能性があります。

RLS を使用する際の制約事項

RLS モードを使用すると、VSAM クラスター属性およびランタイム・オプションにいくつかの制約事項が適用されます。

以下の制約事項に留意してください。

- VSAM ファイルをオープンするとき、VSAM クラスター属性 KEYRANGE および IMBED はサポートされません。
- VSAM クラスター属性 REPLICATE は推奨されません。これは、システム規模のバッファ・プールと、ストレージ階層内の潜在的に大きな CF キャッシュ構造によって、利点が打ち消されるためです。
- VSAM ファイルをオープンするとき、VSAM が空のパスのオープンを認めないため、AIXBLD ランタイム・オプションはサポートされません。代替索引データ・セットを作成するために AIXBLD ランタイム・オプションが必要である場合には、VSAM データ・セットを非 RLS モードでオープンしなければなりません。
- 一時データ・セットは使用できません。

RLS モードでの VSAM ファイルのエラーの処理

アプリケーションが RLS モードの VSAM データ・セットにアクセスする場合には、それぞれの 要求の後でファイル状況および VSAM フィードバック・コードを検査するようにしてください。

このタスクについて

入出力処理時にアプリケーションが「SMSVSAM サーバー利用不能」に直面した場合には、VSAM ファイルの再オープンを試行する前にそれを明示的にクローズしてください。VSAM は、その障害を表す戻りコード 16 を生成します。フィードバック・コードはありません。COBOL プログラムでは、VSAM 戻りコード 16 に関して 2 番目のファイル状況域の最初の 2 バイトを検査することができます。COBOL 実行時には、OPEN 処理中にエラーが発生すると、IGZ0205W メッセージが生成され、ファイルは自動的にクローズされます。

その他のすべての RLS モード・エラーは、VSAM 戻りコード 4、8、または 12 を戻します。

関連タスク

273 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』

VSAM ファイル用のレコード域の割り振り

再入可能 COBOL プログラムの場合、VSAM ファイル用のレコード域はデフォルトで 16 MB 境界より上に割り振られます。

DATA (24) コンパイラ・オプションを指定すると、VSAM レコード域およびその他の動的ストレージ域は 16 MB より下のストレージに割り振られます。

VSAM ファイルレコードのデータを渡すプログラム CALL...USING パラメータ AMODE 24 サブプログラムが影響を受けます。このようなプログラムは、DATA (24) コンパイラ・オプションを使用して再コンパイルするか、Language Environment の HEAP ランタイム・オプションを使用すると、AMODE 24 プログラムによるレコードのアドレス指定が可能になります。

VSAM パフォーマンスの向上

多くの場合、COBOL および VSAM のパフォーマンスの調整を担当するのはシステム・プログラマーです。アプリケーション・プログラマーとしては、以下の表にリストされている VSAM の局面を制御することができます。

このタスクについて

表 31. VSAM パフォーマンスを改善する方法		
VSAM の局面	行うことができること	理由の説明とコメント
アクセス方式サービス・プログラムの呼び出し	IDCAMS を使用した、代替索引の事前作成	

表 31. VSAM パフォーマンスを改善する方法 (続き)

VSAM の局面	行うことができること	理由の説明とコメント
バッファリング	順次アクセスの場合は、もっと多くのデータ・バッファを要求し、ランダム・アクセスの場合は、もっと多くの索引バッファを要求します。両方指定 BUFND そして BUFNI もし ACCESS IS DYNAMIC。 アプリケーションが対話式に実行される場合を除いて、追加のバッファのコーディングを避けます。さらに、入出力遅延が原因であることが考えられる応答時間問題が生じた場合にのみ、バッファをコーディングするようにします。	デフォルトは、1つの索引バッファ (BUFNI) と 2つのデータ・バッファ (BUFND) です。
アクセス方式サービス・プログラムを使用したレコードのロード	以下の場合、アクセス方式サービス・プログラムの REPRO コマンドを使用します。 - ターゲット索引付きデータ・セットに既にレコードが入っている。 - 入力順次データ・セットに、更新されるかまたは索引付きデータ・セットに挿入されるレコードが入っている。 COBOL プログラムを使用してファイルをロードする場合は、OPEN OUTPUT および ACCESS SEQUENTIAL を使用します。	これらの条件のもとでは、REPRO コマンドは、COBOL プログラムと同じ速さまたはそれよりも速く、索引付きデータ・セットを更新することができます。
ファイル・アクセス・モード	最高のパフォーマンスを得るためには、順次にレコードにアクセスします。	動的アクセスは、順次アクセスよりは効率が低く、ランダム・アクセスよりは効率的です。ランダム・アクセスでは、VSAM がそれぞれの要求について索引にアクセスしなければならないため、EXCP が増加します。
キーの設計	レコード内のキーを、高位部分が比較的一定になり、低位部分が頻繁に変わるように設計します。	この方法を使用すると、キーが最適に圧縮されます。
複数の代替索引	複数の代替索引の使用を避けます。	更新は基本パスを介して適用されなければならないため、複数の代替パスを介して反映されることになるため、複数の代替索引を使用すると、パフォーマンスが低下する可能性があります。
相対ファイル編成	VSAM 可変長相対データ・セットではなく、VSAM 固定長相対データ・セットを使用します。	スペース効率は劣りますが、VSAM 固定長相対データセットはより実行時間 VSAM 可変長相対データセットよりも効率的。

表 31. VSAM パフォーマンスを改善する方法 (続き)

VSAM の局面	行うことができること	理由の説明とコメント
制御インターバル・サイズ (CISZ)	システム・プログラマーに、VSAM データ・セットのデータ・アクセスおよび将来の成長性に関する情報を提供してください。この情報から、システム・プログラマーは最適な制御インターバル・サイズ (CISZ) および FREESPACE サイズ (FSPC) を判別することができます。 制御域 (CA) の分割を最小限にするために、CISZ および FSPC に適切な値を選択します。クラスターに対して LISTCAT ALL コマンドを出して、現在の CA 分割の数を診断してから、すべての CA 分割を定期的に省略するためにクラスターを (EXPORT、IMPORT、または REPRO を使用して) 圧縮することができます。	VSAM は、直接アクセス記憶装置 (DASD) の使用アルゴリズムに最も適するように CISZ を計算しますが、これがアプリケーションにとっては効率的でないことがあります。 4K の平均 CISZ はほとんどのアプリケーションに適しています。CISZ をより小さくすることは、挿入を犠牲にして (つまり、CISZ 分割がより多くなり、結果としてデータ・セット内のスペースがより多くなる)、ランダム処理の検索をより速くすることを意味します。CISZ をより大きくすると、各 READ についてチャンネル間で転送されるデータがより多くなります。これは、大規模な OS BLKSIZE。 多くの制御域 (CA) 分割は、VSAM のパフォーマンスにとっては不利です。FREESPACE 値は、ファイルの使用法に応じて、CA 分割に影響を与える場合があります。

関連タスク

207 ページの『VSAM ファイルのアクセス・モードの指定』

z/OS DFSMS: Using Data Sets (リソース・プールの構築、フリー・スペースの最適パーセンテージの選択)

関連参照

z/OS DFSMS カタログのためのアクセス方式サービス・プログラム

拡張アドレッシング機能サポート

拡張アドレッシング機能属性を使用して定義された VSAM データ・セットにアクセスでき、これらの VSAM データ・セットを COBOL ソースを変更することなく COBOL プログラムで使用でき、旧バージョンの COBOL との互換性を維持することができます。

拡張アドレッシング機能サポートを使用すれば、COBOL の外部でより大容量の VSAM データ・セットを定義できます。相対バイト・アドレス (RBA) の 4 バイト・フィールドを使用して課せられる、データ・セット・サイズに関する 4 GB VSAM の構造的制限が除去されます。

拡張アドレッシング機能を使用するには、VSAM データ・セットがストレージ管理サブシステム (SMS) で管理され、拡張フォーマットとして定義されている必要があります。VSAM データ・セットのサイズ制限は、以下のいずれかの方法で決定されます。

- コントロール・インターバル (CI) サイズ x 4 GB
- ボリューム・サイズ x 59

例えば、4 KB の CI サイズからは 16 TB の最大データ・セット・サイズが得られ、32 KB の CI サイズからは 128 TB の最大データ・セット・サイズが得られます。パフォーマンス上の理由から、多くのアプリケーションでは 4 KB の CI サイズの方がよく使用されます。4 GB より大きくなる拡張フォーマット・データ・セットの場合、処理時間は増加しません。

拡張アドレッシング機能は、旧バージョン (RES を使用した VS COBOL II およびそれ以降のコンパイラー) でコンパイルされたプログラムでもサポートされます。

拡張アドレッシング機能と拡張フォーマットは、同じ概念ではありません。拡張フォーマットは拡張アドレッシング機能の前提条件です。拡張フォーマットは、3390/3380 論理トラックでのカウント・キー・データ (CKD) の保管方法に作用する技法です。拡張フォーマットにより、データ・ストライピングが実装さ

れ、入出力操作のパフォーマンスと信頼性が向上します。データ・セットが拡張フォーマット・データ・セットとして割り振られている場合は、各物理ブロックに 32 バイトが追加されます。

制約事項: 拡張アドレッシング機能は、DFSMS/MVS とは 1.3 で KSDS データ・セット用に導入されました。DFSMS/MVS とは 1.4 以降、拡張アドレッシング機能はレコード・レベル共用 (RLS) でサポートされます。DFSMS/MVS とは 1.5 では、拡張アドレッシング機能のサポートが他のすべての VSAM レコード編成に拡張されます。

関連タスク

z/OS DFSMS: Using Data Sets

第 11 章 VSAMDB ファイルの処理

VSAMDB ファイル(データセット)は、COBOL の VSAM を通して JSON ドキュメントを保存し、検索する機能を提供する。

VSAM ファイルの処理能力の詳細については、[VSAM ファイルの処理](#)を参照のこと。

VSAMDB ファイルは、VSAM レコードレベル共有(RLS)によってアクセスされる DATABASE(JSON)属性で定義された VSAM KSDS である。これらのファイル：

- JSON ドキュメントと呼ばれる、UTF-8 エンコードされた JavaScript Object Notation (JSON) テキストデータを格納する。
- 拡張性と可用性の高い z/OS 上の "NoSQL データベースを"ユーザーに提供する。

COBOL に加えて、VSAMDB ファイルにもアクセスできる：

- z/OS VSAM API
- EzNoSQL C API
- EzNoSQL Python API
- EzNoSQL Java API

次の表は、z/OS VSAM と COBOL の両方の文書で同等に使用されている用語を示している。

表 32. z/OS VSAM と COBOL 用語の比較	
z/OS VSAM 用語	COBOL 用語
主キー	プライム・キー
JSON 文書	VSAMDB レコード
キーレス・ベース	暗黙のキー

VSAMDB ファイルの定義

VSAMDB ファイルは、COBOL プログラムの ENVIRONMENT DIVISION の FILE-CONTROL 段落の項目を使用して定義する。

以下の属性は VSAMDB ファイルで使われる：

- FILE-CONTROL の段落：
 - assignment-name の前には "VSAMDB-" を付けなければならない。
 - ORGANIZATION は INDEXED でなければならない。
 - RECORD KEY 句は、暗黙的なプライム・キーにも明示的なプライム・キーにも必要である。
 - RECORD KEY データ項目は、関連するファイルレコード内で定義された項目を参照することはできない。データ項目は、FILE SECTION の外の項目を参照することができる。
 - RECORD KEY データ項目は、動的長エレメンタリー項目とすることができる。
 - RECORD KEY 句の LENGTH 句は、RECORD KEY データ項目が PICTURE U BYTE-LENGTH で定義されている場合に必要である。
 - ALTERNATE RECORD KEY データ・アイテムは、関連するファイル・レコード内で定義されたアイテムを参照することはできない。データ項目は、FILE SECTION の外の項目を参照することができる。
 - ALTERNATE RECORD KEY データ・アイテムは、ダイナミック・レンジス・エレメンタリー・アイテムとすることができる。
 - ALTERNATE RECORD KEY 句の LENGTH 句は、ALTERNATE RECORD KEY データ項目が PICTURE U BYTE-LENGTH で定義されている場合に必要である。

- LENGTH フレーズ項目は、4 バイトのバイナリ符号なし整数でなければならない。つまり、PICTURE 文字列は PICTURE 9(4)から 9(9)でなければならない。USAGE は BINARY、COMP、COMP-4、または COMP-5 でなければならない。
- DATA DIVISION の FILE SECTION にある：
 - FD は、RECORD 句のフォーマット 3 で定義されなければならない（可変長レコードの場合）。DEPENDING ON フレーズは、レコードに固定長の英数字または UTF-8 アイテムが含まれている場合に指定する必要があります。DEPENDING ON 句は、動的長エレメンタリ項目のレコードには指定できない。
 - ファイル・レコードは、英数字または UTF-8 初等的な固定長または動的長の項目でなければならない。固定長アイテムの場合は、ピクチャ文字列 PICTURE X(n) または PICTURE U BYTE-LENGTH n を持つことができる。動的な長さのアイテムの場合は、PICTURE X DYNAMIC または PICTURE U DYNAMIC という文字列と、LIMIT のオプション値を持つことができる。

例：動的長のキーとレコードを使った VSAMDB ファイルの定義

この例は、COBOL プログラムの ENVIRONMENT と DATA DIVISION が、動的長のキーとレコードを使用して VSAMDB ファイルを定義する方法を示している。

VSAMDB ファイルは DD 名に関連付けられます。"VDB1A1".

注：DD 名のプレフィックス "VSAMDB-は"、それが VSAMDB ファイルであることを示す。

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.        VSAMDBE1.
    *****
    ENVIRONMENT DIVISION.
    *****
    INPUT-OUTPUT SECTION.
    FILE-CONTROL.
    *****
    *      VSAMDB FILE
    *****
    SELECT F1 ASSIGN TO VSAMDB-VDB1A1
    ORGANIZATION INDEXED
    ACCESS MODE IS RANDOM
    RECORD KEY IS JSONKEY
    FILE STATUS IS FS1 VFS1.
    *****
    DATA DIVISION.
    *****
    FILE SECTION.
    *****
    FD  F1 RECORD IS VARYING IN SIZE.
    01  F1R PIC U DYNAMIC.
    *****
    WORKING-STORAGE SECTION.
    *****
    01  JSONKEY PIC U DYNAMIC.
    01  FS1-DATA.
    03  FS1                                PIC 99.
    03  VFS1.
    05  VFS1-R15                          PIC 9(4) BINARY.
    05  VFS1-FUNC                          PIC 9(4) BINARY.
    05  VFS1-REASON                        PIC 9(4) BINARY.
```

例：固定長のキーとレコードを使った VSAMDB ファイルの定義

この例は、COBOL プログラムの ENVIRONMENT と DATA DIVISION が、固定長のキーとレコードを使用して VSAMDB ファイルを定義する方法を示している。

必要であれば、動的長さと固定長さのキーとレコードを混ぜて使うこともできる。以下の例では、デモ用に VSAMDB ファイルを定義するために固定長のキーを考えている。

```
IDENTIFICATION DIVISION.
    PROGRAM-ID.        VSAMDBE2.
    *****
    ENVIRONMENT DIVISION.
    *****
    INPUT-OUTPUT SECTION.
```

```

FILE-CONTROL.
*****
*      VSAMDB FILE
*****
      SELECT F1 ASSIGN TO VSAMDB-VDB1A1
      ORGANIZATION INDEXED
      ACCESS MODE IS RANDOM
      RECORD KEY IS JSONKEY
      LENGTH IS JSONKEY-LENGTH BYTES
      FILE STATUS IS FS1 VFS1.
*****
      DATA DIVISION.
*****
      FILE SECTION.
*****
      FD F1 RECORD IS VARYING IN SIZE DEPENDING ON F1R-LENGTH.
      01 F1R PIC U BYTE-LENGTH 500.
*****
      WORKING-STORAGE SECTION.
*****
      01 JSONKEY PIC U BYTE-LENGTH 20.
      01 JSONKEY-LENGTH PIC 9(9) BINARY.
      01 F1R-LENGTH PIC 9(9) BINARY.
      01 FS1-DATA.
          03 FS1                      PIC 99.
          03 VFS1.
              05 VFS1-R15              PIC 9(4) BINARY.
              05 VFS1-FUNC             PIC 9(4) BINARY.
              05 VFS1-REASON           PIC 9(4) BINARY.

```

VSAMDB ファイルの使用

VSAMDB ファイルを処理するには、OPEN、WRITE、START、READ、REWRITE、DELETE、CLOSE ステートメントを使用する。

アクセスモードと OPEN 文のモードによって、VSAMDB データセットで利用できる入出力文が決まる。詳細については、Access モード、OPEN 文のモード、COBOL ファイル処理文の互換性の一覧表については、[VSAM ファイルの入出力文のコーディングを参照のこと](#)。

VSAMDB ファイルを処理するには、以下のステートメントを使用する：

- **OPEN**

VSAMDB ファイルを COBOL プログラムに接続する。

- **WRITE**

ファイルに新しいレコードを追加するか、ファイルをロードする。VSAMDB ファイルが暗黙のキーで定義されている場合、そのキーはプライム・キー項目に入力される。

- **START**

シーケンシャル処理のファイル位置インジケータを確立する。

- **READ と READ NEXT**

ファイルからレコードを取得し、ファイル位置インジケータを設定する。

- **REWRITE**

以前に READ 文で読み込んだファイルのレコードを更新する。

- **DELETE**

READ 文で読み込んだファイルからレコードを削除する。

- **CLOSE**

VSAMDB ファイルをプログラムから切断する。

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。インジケータは、START または（シーケンシャルアクセスまたはランダムアクセス）READ ステートメントの成功によって設定される。

シーケンシャル READ ステートメントは、確立されたファイル位置インジケータを使用し、それを更新する。

ランダムアクセス READ、ファイル位置インジケータを設定すると、その後の REWRITE、DELETE。

シーケンシャル・アクセス READ がファイル位置インジケータを設定した場合、その後の REWRITE または DELETE ステートメントには影響されない。

START ステートメントの PARTIAL フレーズを使用すると、キーの部分一致比較演算を使用することができません。これは、キーの接頭辞を使ってファイルカーソルを配置しようとするときに便利である。

VSAMDB ファイルからレコードを読み込む

VSAMDB ファイルからレコードを取得するには、READ 文を使用する。

レコードを読むには、INPUT または I-O としてファイルを開く必要がある。プログラムは、READ 毎にファイルステータスキーをチェックする必要がある。

VSAMDB ファイルはインデックス付きファイルであるため、ファイルに順次アクセスすると、ファイルは、ファイル位置インジケータの現在の位置から、キー (RECORD KEY または ALTERNATE RECORD KEY) の順序に従って検索される。ランダムにファイルにアクセスすると、READ 文の実行中に RECORD KEY または ALTERNATE RECORD KEY のキーの値に従ってレコードが検索される。

ランダムレコード検索用のキーの値は、JSON 値 (通常は JSON 文字列) を含む必要があるため、UTF-8。このため、キーには PIC U DYNAMIC または PIC U BYTE-LEGNTN を使用する。詳細は [JSON の紹介](#) を参照。例えば、JSON 文字列は常にダブルクオート文字 " で始まり、文字テキストを挟んで終わる。COBOL データ項目から直接 JSON 値を生成するには JSON GENERATE 文を使用し、必要な文字列を作成するには STRING 文を使用する。

例：JSON GENERATE を使って JSON 文字列を作成する

この例では、JSON GENRATE を使って JSON 文字列を作成する方法を示している。

次のコード・スニペットは、COBOL 項目から JSON 文字列を生成する方法を示している。この場合、デスティネーション項目の F-KEY-VALUE が RECORD KEY として直接使用される。

```
ENVIRONMENT DIVISION.  
FILE CONTROL.  
    SELECT FILE1 ...  
    RECORD KEY IS F-KEY-VALUE.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 MY-DATA PIC X(100).  
01 F-KEY-VALUE PIC U DYNAMIC.  
..  
PROCEDURE DIVISION.  
    MOVE "Alice" TO MY-DATA  
    JSON GENERATE F-KEY-VALUE FROM MY-DATA  
    NAME MY-DATA IS OMITTED
```

JSON GENERATE 文の実行に成功すると、F-KEY-VALUE には、UTF-8 でエンコードされた以下の JSON 文字列が格納される：

```
"Alice"
```

注：F-KEY-VALUE のような動的長さの項目については、長さは暗黙的に格納される。

例：JSON GENERATE を使って JSON 文字列を固定長アイテムにする

この例では、JSON GENERATE を使って JSON 文字列を固定長のアイテムにする方法を示している。

以下の例では、JSON GENERATE で、キーの現在のバイト長を示す LENGTH フレーズと、バイト長を取得する COUNT BYTES フレーズを使用している。

```
ENVIRONMENT DIVISION.  
FILE CONTROL.
```



```

SELECT FILE1 ...
RECORD KEY IS F-KEY-VALUE
LENGTH IS F-KEY-LEN BYTES.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 MY-DATA PIC X(100).
01 F-KEY-VALUE PIC U DYNAMIC.
01 F-KEY-LEN PIC 9(9) BINARY.
...
PROCEDURE DIVISION.
MOVE "Alice" TO MY-DATA
JSON GENERATE F-KEY-VALUE FROM MY-DATA
COUNT BYTES IN F-KEY-LEN
NAME MY-DATA IS OMITTED

```

注：VSAMDB ファイルは、文字長ではなく、キーとレコードのバイト長を使用して動作する。これが、文字長固定の PIC U 項目をキーやレコードとして指定できない理由である。固定バイト長および固定文字長の UTF-8 項目の詳細については、[COBOL での UTF-8 データ \(Unicode\) の使用](#)を参照のこと。

RECORD KEY LENGTH フレーズの使用

RECORD KEY 句または ALTERNATE RECORD KEY 句の LENGTH 句を使用して長さ項目を指定する場合は、READ 文を実行する前に、長さ項目にキーまたは代替キーの適切なバイト長を設定する必要があります。LENGTH フレーズを使用しない場合、キーアイテムの最大長が想定される。

JSON キーと UTF-8 データの長さは可変であるため、LENGTH フレーズを使用するか、動的長さのエレメンタリー項目（長さが暗黙的に格納される）を使用することを強く推奨します。

RECORD KEY 節または ALTERNATE RECORD 節で動的長さの基本項目を使用する場合、それぞれ LENGTH 句を指定することはできません。

注：VSAMDB のサポートとともにエンタープライズ COBOL 6.5、COBOL プログラムが生成された JSON のバイト長を取得するのに便利のように、JSON GENERATE 文に COUNT BYTES 句が追加された。さらに、ULENGTH 関数は、UTF-8 アイテムの文字長を計算するときに便利のように、開始バイト位置とバイト長を受け付けるように拡張されている。

動的長レコードでの READ 文の使用

JSON ドキュメントは可変長であるため、VSAMDB ファイルは、動的長エレメンタリー項目を含むレコードで定義する。

このアプローチには次のような利点がある：

- サテライト・アイテムのレコード・バイト長を記録する必要はない。
- 大きな固定長レコードを割り当てるときに発生する、無駄な記憶領域を心配する必要はない。
- 手でストレージのサイズを変更する必要はありません。必要であれば、READ ステートメントは、データに合わせて、ダイナミックレンジエレメンタリーアイテムに関連するストレージのサイズを自動的に変更することができる。

注：DYNAMIC LENGTH LIMIT の値が最大のレコードを含むのに十分であることを確認する。READ、READ NEXT 文の実行中に、COBOL ランタイムは動的長項目がレコードを READ するのに十分なスペースを持っているかどうかを判断する。もしそうなら、READ ステートメントが実行される。そうでない場合は、動的に割り当てられたストレージのサイズが適切に変更され、READ。

エンタープライズ COBOL 6.5 では、以下のステートメントを使用することで、動的長エレメンタリー項目に関連付けられた動的に割り当てられたストレージを明示的に解放することができます：

```
SET LENGTH OF item TO ZERO
```

この機能は、例えば、大きなレコードが読み込まれ処理され（その結果、大きなバッファが確保される）、（そのような大きなバッファが無駄になる可能性がある）プログラムの残りの部分を続行する前にバッファを解放したい場合に便利である。

VSAMDB ファイルへのレコードの追加

既存のレコードを置き換えることなく VSAMDB ファイルにレコードを追加するには、WRITE ステートメントを使用する。

このプログラムは、WRITE の各ステートメントの後に、ファイル・ステータス・キーをチェックする。WRITE 文を実行する前に、RECORD KEY または ALTERNATE RECORD KEY 項目を入力しないでください。

VSAMDB ファイルに暗黙のプライム・キー(キー・レス・ベース)がある場合は、RECORD KEY を少なくとも 122 バイトのストレージで定義しなければならない。これは、PIC U BYTE-LENGTH 122 または PIC U DYNAMIC (LIMIT が 122 と等しいかそれより大きい) で定義された項目を使用して行うことができる。

WRITE 文の実行中に、VSAM は、キー名 "znsq_id" の一意な主キーを自動的に生成し、レコードに挿入する。文の実行に成功すると、生成されたキーが RECORD KEY 項目で利用できるようになる。

VSAMDB ファイルのレコードの更新

READ ステートメントを実行して、VSAMDB ファイルの更新したいレコードを読み込む。

READ ステートメントを実行した後、更新されたデータをレコード領域に入力し、REWRITE ステートメントを使用する。

プログラムは、REWRITE の各ステートメントの後に、ファイル・ステータス・キーをチェックしなければならない。

RECORD KEY と ALTERNATE RECORD KEY の項目は、REWRITE ステートメントに設定する必要はありません。ランタイムシステムは、READ と REWRITE の間のレコードを暗黙的に追跡します。

注: READ を実行せずに REWRITE を実行しないでください。

VSAMDB ファイルのレコードの削除

READ 文を実行して、VSAMDB ファイルから削除したいレコードを読み込む。

READ、次に DELETE。

レコード領域と RECORD KEY および ALTERNATE RECORD KEY の項目は、DELETE ステートメントに設定する必要はありません。ランタイムシステムは、READ と DELETE の間のレコードを暗黙的に追跡します。

プログラムは、DELETE の各ステートメントの後に、ファイル・ステータス・キーをチェックしなければならない。

注: READ を実行せずに DELETE を実行しないでください。

VSAMDB ファイルの定義 z/OS

VSAMDB ファイル (VSAMDB ファイルは VSAM KSDS インデックスファイルであり、RLS によってアクセスされる) の定義方法は、VSAM ファイルの定義方法と同様である。

詳細については、[VSAM ファイルの定義](#)を参照のこと。

以下は、VSAMDB ファイルを定義する際の例外である：

- VSAMDB ファイルは、IDCAMS で DEFINE CLUSTER DATABASE (JSON) を使用して定義される。これは、ファイルが JSON データベースであることを示す。
- VSAMDB ファイルには、レコード内のオフセットと長さで定義されるプライム・キーがない。明示的なプライム・キーは、IDCAMS KEYNAME パラメータを使用して名前で識別される。暗黙の主キーが必要な場合は、KEYNAME パラメータを省略する。
- 代替インデックスは、オフセットと長さで定義されていません。ALTKEYS パラメータを使用して、キー名を特定する。

DFSMSStvs で VSAMDB を使用する

トランザクション VSAM (DFSMSStvs) サービスとのやり取り、例えばコミットの発行やバックアウトレポート処理などは、アプリケーションプログラマの責任である。

この相互作用は、Enterprise COBOL の言語コンストラクトでは直接サポートされていない。

VSAMDB システム要件

VSAMDB は z/OS 2.5 またはそれ以降を必要とする。

- VSAMDB 機能を有効にするには、以下の APAR を適用する必要がある：
 - OA66674
 - OA65272
- IBM パラレルシステムを設定し、有効にする必要があります。
- z/OS 2.5 またはそれ以降をプレックスモードで実行している論理パーティション (LPAR) が少なくとも 1 つ、必要な APAR とともにインストールされている。
- 少なくとも 1 つの内部または外部カップリング・ファシリティ (CF) が LPAR に接続されています。詳細は、『Processor Resource/Systems Manager Planning Guide』を参照。
- VSAMDB データベースにアクセスする各 LPAR で、SMSVSAM アドレス空間を有効にする。詳細は、[VSAM レコード・レベル共有の管理](#)を参照のこと。
- CACHESET を含む 1 つ以上の SMS 保存クラス (SORCLAS) を構成します。詳細は、[VSAM RLS のストレージ・クラスの定義](#)を参照のこと。CACHESET は、VSAMDB データベースが使用する 1 つ以上の CF キヤッシュ構造の名前を指定する。詳細については、[データ・クラスでの VSAM RLS 属性の定義](#)を参照してください。

VSAMDB 全プログラムのサンプル

Enterprise COBOL for z/OS のインストール・データセットに含まれている SIGYSAMP データセットをブラウズして、以下のサンプルを参照できます。

表 33. VSAMDB プログラムサンプル	
メンバー名	説明
IGYVDB1	非 VSAMDB ファイルにレコードを初期化し、レコードを読み込んで JSON に変換し、JSON を VSAMDB ファイルに書き込む。
IGYVDB2	JSON ドキュメント(レコード)で VSAMDB ファイルを初期化し、レコードを読み取り、レコードを COBOL グループに変換し、COBOL グループをレコードとして非 VSAMDB データセットに書き込む。
IGYVDB3	プライム・キーとオルタネート・キーを明示し、固定長のキーとレコードを持つ VSAMDB ファイルを使用する。
IGYVDB4	動的長のキーとレコードを持つ VSAMDB ファイルを使用する。
IGYVDB5	暗黙のプライムキーを持つ VSAMDB ファイルを使用する。

第 12 章 行順次ファイルの処理

行順次ファイルは、z/OS UNIX ファイルシステムに常駐し、印刷可能文字および制御文字をデータとして収容できます。各レコードは、EBCDIC 改行文字 (X'15') で終了します (この文字はレコードの長さには含まれません)。

このタスクについて

行順次ファイルは順次ファイルであるため、各レコードは入力順に 1 つずつ配置されます。プログラムでは、これらのファイルを順次にのみ処理することができ、レコードを、それらがファイルに入っているのと同じ順序で検索します (READ ステートメントを使用して)。新しいレコードは、前のレコードの後に入れます。

プログラムで行順次ファイル进行处理するには、以下のような COBOL 言語ステートメントをコーディングします。

- ファイルの識別と説明 ENVIRONMENT DIVISION そしてその DATA DIVISION
- PROCEDURE DIVISION 内でファイルのレコードを処理する

レコードの作成後は、ファイル内でレコードの長さや位置を変えることはできず、また削除することもできません。

関連タスク

[237 ページの『COBOL での行順次ファイルおよびレコードの定義』](#)

[238 ページの『行順次ファイルの割り振り』](#)

[239 ページの『行順次ファイル用の入出力ステートメントのコーディング』](#)

[241 ページの『行順次ファイルのエラーの処理』](#)

[UNIX システム・サービス ユーザーズ・ガイド](#)

COBOL での行順次ファイルおよびレコードの定義

使用 FILE-CONTROL 段落の ENVIRONMENT DIVISION COBOL プログラム内のファイルを行順次ファイルとして定義し、ファイルに対応する外部ファイル名 (DD 名または環境変数名) に関連付けます。

このタスクについて

外部ファイル名とは、ファイルがオペレーティング・システムに認識されるときに使用される名前です。次の例では、COMMUTER-FILE は、プログラムがファイルに使用する名前であり、COMMUTR は外部名です。

```
FILE-CONTROL.  
  SELECT COMMUTER-FILE  
  ASSIGN TO COMMUTR  
  ORGANIZATION IS LINE SEQUENTIAL  
  ACCESS MODE IS SEQUENTIAL  
  FILE STATUS IS ECODE.
```

外部名の前に、ASSIGN *assignment-name* 節に編成フィールド (S- または AS-) を含めてはなりません。の ACCESS フレーズと FILE STATUS 句はオプションです。

関連タスク

[238 ページの『行順次ファイルの構造の記述』](#)

[238 ページの『行順次ファイルの割り振り』](#)

[239 ページの『行順次ファイル用の入出力ステートメントのコーディング』](#)

関連参照

[238 ページの『行順次ファイルの制御文字』](#)

行順次ファイルの構造の記述

FILE SECTION で、そのファイルのファイル記述 (FD) 記入項目をコーディングします。関連するレコード記述項目では、*record-name* およびレコード長を定義します。

このタスクについて

RECORD 節を使用して、レコードの論理サイズ (バイト単位) をコーディングしてください。行順次ファイルはストリーム・ファイルです。行順次ファイルの本質は文字中心であるので、物理レコードは可変長です。

次の例は、行順次ファイルの FD 記入項目を示しています。

固定長レコードの場合:

```
FILE SECTION.  
FD  COMMUTER-FILE  
   RECORD CONTAINS 80 CHARACTERS.  
01  COMMUTER-RECORD.  
    05  COMMUTER-NUMBER          PIC  X(16).  
    05  COMMUTER-DESCRIPTION     PIC  X(64).
```

可変長レコードの場合:

```
FILE SECTION.  
FD  COMMUTER-FILE  
   RECORD VARYING FROM 16 TO 80 CHARACTERS.  
01  COMMUTER-RECORD.  
    05  COMMUTER-NUMBER          PIC  X(16).  
    05  COMMUTER-DESCRIPTION     PIC  X(64).
```

同じ固定サイズでコーディングし、OCCURS DEPENDING ON いくつかの条項についても level-01 ファイルに関連付けられたレコード記述エントリでは、その固定サイズが論理レコード長になります。しかし、レコードの終わりにあるブランクはファイルに書き込まれないので、物理レコードは可変長になります。

関連タスク

[238 ページの『行順次ファイルの割り振り』](#)

[239 ページの『行順次ファイル用の入出力ステートメントのコーディング』](#)

関連参照

データ部 -- ファイル記述項目

(Enterprise COBOL for z/OS 言語解説書)

行順次ファイルの制御文字

行順次ファイルには、制御文字を含めることができます。行順次ファイルに改行文字 (X'15') が含まれている場合、この改行文字はレコード区切り文字として機能することに注意してください。

改行以外の制御文字は、データとして扱われ、レコードの一部になります。

行順次ファイルの割り振り

行順次ファイルを割り当てることができます。z/OS UNIX ファイルシステムのいずれかを使用して DD ステートメントまたは環境変数。行順次ファイルの割り振りは、COBOL ファイルの割り振りに関する一般的な規則に従います。

このタスクについて

行順次ファイルを割り振るには、ASSIGN 節の外部名と一致する名前を持つ、DD 割り振りまたは環境変数をコーディングしてください。

- DD 割り振り:

- PATH='absolute-path-name' を指定する DD ステートメント
- PATH('absolute-path-name') を指定する TSO 割り振り

以下のオプションを指定することもできます。

- PATHOPTS
- PATHMODE
- PATHDISP
- PATH(*absolute-path-name*) の値を持つ環境変数。これ以外の値を指定することはできません。

例えば、*assignment-name* が COMMUTR である COBOL ファイル用に、プログラムで z/OS ユニックスファイル /u/myfiles/commuterfile を使用するには、次のコマンドを使用できます。

```
export COMMUTR="PATH(/u/myfiles/commuterfile)"
```

関連タスク

[175 ページの『ファイルの割り振り』](#)

[237 ページの『COBOL での行順次ファイルおよびレコードの定義』](#)

関連参照

[MVS プログラム管理: ユーザーズ・ガイドおよび解説書](#)

行順次ファイル用の入出力ステートメントのコーディング

行順次ファイル进行处理するには、下記の入出力ステートメントをコーディングします。

このタスクについて

OPEN

ファイルの処理を開始します。

行順次ファイルは、INPUT、OUTPUT、または EXTEND としてオープンすることができます。行順次ファイルを I-O としてオープンすることはできません。

READ

ファイルからレコードを読み取ります。

順次処理では、プログラムは、ファイルの作成時に入力されたのと同じ順序で、レコードを次々と読み取ります。

WRITE

ファイルにレコードを作成します。

プログラムは、ファイルの終わりに新しいレコードを書き込みます。

CLOSE

ファイルとプログラムの接続を解放します。

関連タスク

[237 ページの『COBOL での行順次ファイルおよびレコードの定義』](#)

[238 ページの『行順次ファイルの構造の記述』](#)

[240 ページの『行順次ファイルのオープン』](#)

[240 ページの『行順次ファイルからのレコードの読み取り』](#)

[241 ページの『行順次ファイルへのレコードの追加』](#)

[241 ページの『行順次ファイルのクローズ』](#)

[241 ページの『行順次ファイルのエラーの処理』](#)

関連参照

OPEN ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

READ ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

WRITE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

CLOSE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

行順次ファイルのオープン

プログラムは、READ または WRITE ステートメントを使用してファイルのレコードを処理する前に、OPEN ステートメントでファイルをオープンしなければなりません。OPEN ステートメントは、ファイルが使用可能であるか、または動的に割り振られていれば、機能します。

このタスクについて

プログラムの実行中にファイルが再度オープンされないように、CLOSE WITH LOCK をコーディングしてください。

関連タスク

[240 ページの『行順次ファイルからのレコードの読み取り』](#)

[241 ページの『行順次ファイルへのレコードの追加』](#)

[241 ページの『行順次ファイルのクローズ』](#)

[238 ページの『行順次ファイルの割り振り』](#)

関連参照

OPEN ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

CLOSE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

行順次ファイルからのレコードの読み取り

行順次ファイルから読み取りを行うには、ファイルをオープンして、READ ステートメントを使用します。プログラムは、ファイルの作成時に入力されたのと同じ順序でレコードを次々と読み取ります。

このタスクについて

ファイル・レコード内の文字は、以下のいずれかの条件が起こるまで、一度に1文字ずつレコード域に読み込まれます。

- レコード区切り文字 (EBCDIC 改行文字) が検出される。

区切り文字は廃棄され、レコード域の残りの部分はスペースで埋められます。(レコード域はファイル・レコードより長くなります。)

- レコード域全体が文字で満たされる。

次の未読文字がレコード区切り文字であると、その文字は廃棄されます。次の READ は、次のレコードの先頭文字から読み取りを行います。(レコード域はファイル・レコードと同じ長さになります。)

これ以外の場合、次の未読文字が、次の READ によって読み取られる先頭文字になります。(レコード域はファイル・レコードより短くなります。)

- ファイルの終わりが検出される。

レコード領域の残りにはスペースが埋められます。(レコード域はファイル・レコードより長くなります。)

関連タスク

[240 ページの『行順次ファイルのオープン』](#)

[241 ページの『行順次ファイルへのレコードの追加』](#)

[241 ページの『行順次ファイルのクローズ』](#)

[238 ページの『行順次ファイルの割り振り』](#)

関連参照

OPEN ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

WRITE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

行順次ファイルへのレコードの追加

行順次ファイルに追加を行うためには、ファイルを EXTEND としてオープンし、WRITE ステートメントを使用して、ファイルの最後のレコードの直後にレコードを追加します。

このタスクについて

レコード域の終わりにあるブランクは除去され、レコード区切り文字が追加されます。レコード域内の文字(先頭文字から、追加されたレコード区切り文字まで)は、1つのレコードとしてファイルに書き込まれます。

行順次ファイルに書き込まれるレコードは、USAGE DISPLAY および DISPLAY-1 項目だけを含んでいなければなりません。ゾーン分け 10 進データ項目は符号なしか、SEPARATE のフレーズ SIGN 署名された場合の条項。

関連タスク

[240 ページの『行順次ファイルのオープン』](#)

[240 ページの『行順次ファイルからのレコードの読み取り』](#)

[241 ページの『行順次ファイルのクローズ』](#)

[238 ページの『行順次ファイルの割り振り』](#)

関連参照

OPEN ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

WRITE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

行順次ファイルのクローズ

プログラムを行順次ファイルから切り離すには、CLOSE ステートメントを使用します。既にクローズされているファイルをクローズしようとする、論理エラーになります。

このタスクについて

行順次ファイルをクローズしなくても、以下の条件のもとでは、ファイルは自動的にクローズされます。

- 実行単位が正常に終了したとき。
- 実行単位が異常終了したときは、TRAP(ON) ランタイム・オプションが設定されている場合。
- 言語環境条件処理が完了し、その条件が起こったルーチン以外のルーチンでアプリケーションが再開すると、実行単位内の COBOL プログラムのうち、再度呼び出されて再入される可能性のあるプログラムに定義されているオープン・ファイルはクローズされます。

(条件が処理された後で) プログラムが再開する位置を変更することができます。これは、言語環境の CEEMRCR 呼び出し可能サービスで再開カーソルを移動するか、または C longjmp 呼び出しのような HLL 言語構造体を使用して行います。

これらの暗黙の CLOSE 操作が実行されると、ファイル状況コードが設定されますが、EXCEPTION/ERROR 宣言は呼び出されません。

関連タスク

[240 ページの『行順次ファイルのオープン』](#)

[240 ページの『行順次ファイルからのレコードの読み取り』](#)

[241 ページの『行順次ファイルへのレコードの追加』](#)

[238 ページの『行順次ファイルの割り振り』](#)

関連参照

CLOSE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

行順次ファイルのエラーの処理

入出力ステートメントが失敗しても、COBOL がユーザーに代わって訂正処置を取るわけではありません。入出力ステートメントが失敗した後で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。

このタスクについて

COBOL は、特定の行順次入出力エラーを代行受信して処理するために、以下の言語エレメントを提供します。

- ファイル終了句 (AT END)
- EXCEPTION/ERROR 宣言
- FILE STATUS 節

これらの技法を使用しなかった場合に、入出力処理でエラーが起こると、言語環境条件が発生します。

FILE STATUS 節を使用する場合は、必ずキーを調べ、キー値に基づいて適切なアクションを取るようにしてください。キーを調べなくても、プログラムが実行を継続できる可能性があります、予期したものではありません結果になることがあります。

関連タスク

[239 ページの『行順次ファイル用の入出力ステートメントのコーディング』](#)

[267 ページの『入出力操作でのエラーの処理』](#)

第 13 章 ファイルのソートおよびマージ

このタスクについて

SORT または MERGE ステートメントを使用して、レコードを特定の順序で配置することができます。同じ COBOL プログラム内で 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。

注：このトピックで説明する SORT ステートメント、ソート・プロセス、およびソートの制約事項は、フォーマット 1 SORT ステートメントにのみ関連しています。フォーマット 2 SORT ステートメントを使用した表のソートの詳細については、94 ページの『[テーブルのソート](#)』を参照してください。

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。

Enterprise COBOL では、ソートおよびマージ用の IBM ライセンス・プログラムは DFSORT または同等のプログラムでなければなりません。DFSORT に言及している場所では、任意の同等のソートまたはマージ製品を使用することができます。

SORT または MERGE ステートメントを含む COBOL プログラムは、16 MB 境界の上または下に常駐できます。

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。

手順

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に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。

タスクの結果

制約事項:

- z/OS ユニックスのもとでは、SORT ステートメントまたは MERGE ステートメントを含む COBOL プログラムを実行することはできません。この制約には、BPXBATCH も含まれます。
- THREAD オプションを指定してコンパイルされたプログラムでは、SORT または MERGE ステートメントを使用できません。これには、オブジェクト指向構文およびマルチスレッド・アプリケーションを使用するプログラムが含まれます。これらのプログラムは、両方とも THREAD オプションを必要とします。また、SORT または MERGE ステートメントを使用する COBOL プログラムは、z/OS Unix システム・サービスを必要とする他のアプリケーション、またはマルチスレッド化を使用するアプリケーションを直接的にも間接的にも呼び出すことができません。例えば、JVM では両方とも使用されます。

- DFSORT の従来の手法を使用することはできません。従来の技法は、次の場合に使用されます。
 - NOBLKSET オプションが指定されている。BLKSET は、DFSORT を呼び出すときのデフォルトです。
 - 中間作業用ストレージにテープ装置が使用されている。
 - L5 は、DFSORT OPTION 制御の RECORD ステートメントで使用されます。L5 は、平均レコード長を指定します。L5 を使用する代わりに、AVGRLLEN=n ステートメントを使用して同じものを指定できます。

レガシー COBOL プログラムのマイグレーションを支援するために、AMODE 31 で実行されるプログラムに対して、容認ランタイム APAR PH20569(z/OS 2.2) および PH21261(z/OS 2.3/2.4) が提供されています。AMODE 64 に関する耐障害性はありません。詳しくは、Using DFSORT option NOBLKSET (Enterprise COBOL 移行ガイド) を参照してください。

関連概念

[244 ページの『ソートおよびマージ・プロセス』](#)

関連タスク

[94 ページの『テーブルのソート』](#)

[245 ページの『ソートまたはマージ・ファイルの記述』](#)

[245 ページの『ソートまたはマージへの入力 of 記述』](#)

[247 ページの『ソートまたはマージからの出力 of 記述』](#)

[251 ページの『ソートまたはマージの要求』](#)

[254 ページの『ソートまたはマージの成否の判断』](#)

[255 ページの『ソートまたはマージ操作の途中停止』](#)

[255 ページの『FASTSORT によるソート・パフォーマンスの向上』](#)

[258 ページの『ソート動作の制御』](#)

[DFSORT アプリケーション・プログラミング・ガイド](#)

関連参照

[262 ページの『CICS SORT アプリケーションの制約事項』](#)

[SORT ステートメント \(Enterprise COBOL for z/OS 言語解説書\)](#)

[MERGE ステートメント \(Enterprise COBOL for z/OS 言語解説書\)](#)

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。

ファイルを並べ替えるには、フォーマット 1SORT 声明。

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。

使用 MERGE . . . USING 1 つの連続したファイルに結合するファイルに名前を付けます。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

[252 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 \(Enterprise COBOL for z/OS 言語解説書\)](#)

[MERGE ステートメント \(Enterprise COBOL for z/OS 言語解説書\)](#)

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。

このタスクについて

次のようにコーディングします。

手順

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。例:

```
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。例:

```
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).
```

タスクの結果

SD 項目で記述されたファイルは、ソート操作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。また、このファイルに DD 名定義を提供する必要はありません。

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。

このタスクについて

手順

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。例:

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT Input-File ASSIGN TO InFile.
```

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。例:

タスクの結果

251 ページの『ソートまたはマージの要求』

13 ページの『FILE SECTION 記入項目』

251 ページの『ソートまたはマージの要求』

入力プロシージャのコーディング

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、形式 1 SORT ステートメントの INPUT PROCEDURE 句を使用してください。

このタスクについて

入力プロシージャを使用して、以下のことを行うことができます。

- データ項目をソートファイルに解放する WORKING-STORAGE または LOCAL-STORAGE。
- 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.
```

次の表は、RELEASE そして RELEASE FROM 声明。

RELEASE	RELEASE FROM
MOVE SORT-EXT-RECORD TO SORT-RECORD PERFORM RELEASE-SORT-RECORD RELEASE-SORT-RECORD. RELEASE SORT-RECORD	PERFORM RELEASE-SORT-RECORD RELEASE-SORT-RECORD. RELEASE SORT-RECORD FROM SORT-EXT-RECORD

関連参照

249 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 (Enterprise COBOL for z/OS 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。

このタスクについて

手順

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。例:

```
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。例:

```
DATA DIVISION.  
FILE SECTION.  
FD Output-File  
    LABEL RECORDS ARE STANDARD  
    BLOCK CONTAINS 0 CHARACTERS  
    RECORDING MODE IS F  
    RECORD CONTAINS 100 CHARACTERS.  
01 Output-Record PIC X(100).
```

タスクの結果

関連タスク

[248 ページの『出力プロシージャのコーディング』](#)

[251 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』](#)

出力プロシージャのコーディング

ソート作業ファイルから別のファイルに書き込む前にソートされたレコードを選択、編集、または変更するには、OUTPUT PROCEDURE のフレーズフォーマット 1SORT 声明。

このタスクについて

それぞれの出力プロシージャは、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。

DFSORT コーディング: DFSORT を使用する場合そして RETURN ステートメントは AT ENDCOBOL プログラムの実行が終了する前の状態、フォーマット 1SORT ステートメントは DFSORT メッセージで異常終了する可能性があります IEC025A。このような状況を避けるには、RETURN 声明文 AT END フレーズ。加えて、AT END 条件が検出されるまで RETURN ステートメントが実行されるようにもしてください。AT END 条件が起こるのは、最後のレコードがソート作業ファイルからプログラムに戻され、後続の RETURN ステートメントが実行された後です。

249 ページの『例: DFSORT を使用する際の出力プロシージャのコーディング』

関連参照

249 ページの『入出力プロシージャに関する制約事項』

RETURN ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

例: DFSORT を使用する際の出力プロシージャのコーディング

次の例は、RETURN 声明は、AT END プログラムの実行が終了する前の状態。AT END 句をコーディングした RETURN ステートメントは、AT END 条件が起こるまで実行されます。

```
IDENTIFICATION DIVISION.
DATA DIVISION.
FILE SECTION.
SD OUR-FILE.
01 OUR-SORT-REC.
   03 SORT-KEY          PIC X(10).
   03 FILLER             PIC X(70).

WORKING-STORAGE SECTION.
01 WS-SORT-REC          PIC X(80).
01 END-OF-SORT-FILE-INDICATOR PIC X VALUE 'N'.
   88 NO-MORE-SORT-RECORDS      VALUE 'Y'.

PROCEDURE DIVISION.
A-CONTROL SECTION.
  SORT OUR-FILE ON ASCENDING KEY SORT-KEY
  INPUT PROCEDURE IS B-INPUT
  OUTPUT PROCEDURE IS C-OUTPUT.

B-INPUT SECTION.
  MOVE . . . . . TO WS-SORT-REC.
  RELEASE OUR-SORT-REC FROM WS-SORT-REC.

C-OUTPUT SECTION.
  DISPLAY 'STARTING READS OF SORTED RECORDS: '.
  RETURN OUR-FILE
  AT END
  SET NO-MORE-SORT-RECORDS TO TRUE.
  PERFORM WITH TEST BEFORE UNTIL NO-MORE-SORT-RECORDS
  IF SORT-RETURN = 0 THEN
    DISPLAY 'OUR-SORT-REC = ' OUR-SORT-REC
    RETURN OUR-FILE
  AT END
  SET NO-MORE-SORT-RECORDS TO TRUE
  END-IF
  END-PERFORM.
```

入出力プロシージャに関する制約事項

SORT によって呼び出されるそれぞれの入力または出力プロシージャ、および MERGE によって呼び出される各出力プロシージャには、いくつかの制約事項が適用されます。

以下の制約事項に従ってください。

- プロシージャに SORT または MERGE ステートメントを含めてはなりません。

制約事項: 入力プロシージャまたは出力プロシージャ内で、標準リンケージ規約に従うプログラムを呼び出すことができます。ただし、呼び出されたプログラムは、SORT、MERGE、RELEASE、または RETURN ステートメントを発行できません。

- プロシージャの中で 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の外側にあるプロシージャ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)。
- 入力プロシージャまたは出力プロシージャ内で、標準リンケージ規約に従う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してはなりません。
- 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。
- 言語環境の条件処理では、入力または出力プロシージャの中でユーザー作成条件処理ルーチンを設定することができません。

関連タスク

247 ページの『入力プロシージャのコーディング』

248 ページの『出力プロシージャのコーディング』

言語環境プログラミングガイド (リンク・エディットおよび実行の準備)

ソート・データ・セットおよびマージ・データ・セットの定義

z/OS のもとで DFSORT を使用するには、実行時 JCL に DD ステートメントをコーディングして、必要なデータ・セットを記述しなければなりません。

このタスクについて

ソートまたはマージ作業域

少なくとも 3 つのデータ・セットを定義します。SORTWK01、SORTWK02、SORTWK03、...、SORTWKnn (nn は 99 以下) などです。これらのデータ・セットは z/OS UNIX ファイルシステム にあってはなりません。

SYSOUT

データ・セット名を変更しない限り、ソート診断メッセージ用にこのデータ・セットを定義します。(名前は、SORT-CONTROL データ・セット内の OPTION 制御ステートメントの MSGDDN キーワードを使用して、または SORT-MESSAGE 特殊レジスターを使用して変更してください。)

SORTCKPT

ソートまたはマージでチェックポイントを設定する場合に、このデータ・セットを定義します。

入出力

必要に応じて、入出力データ・セットを定義します。

SORTLIB (DFSORT ライブラリー)

ソート・モジュールが入ったライブラリー (例えば、SYS1.SORTLIB) を定義します。

関連タスク

258 ページの『ソート動作の制御』

261 ページの『DFSORT によるチェックポイント・リスタートの使用』

可変長レコードのソート

このタスクについて

ソートへの入力ファイルに可変長レコードが入っていても、ソート作業ファイルは可変長として定義しないと可変長になりません。

コンパイラーがソート作業ファイルを可変長と見なすのは、そのファイルの SD 記入項目で以下のいずれかのエレメントをコーディングした場合です。

- RECORD IS VARYING 節
- 異なるサイズのレコードを定義する 2 つ以上のレコード記述、または OCCURS DEPENDING ON 句

SD 記入項目では RECORDING MODE 節は認められないため、ソート作業ファイルについて RECORDING MODE V を使用することはできません。

パフォーマンスの考慮: 可変長ファイルに対するソートのパフォーマンスを向上させるには、SMS= 制御カードまたは SORT-MODE-SIZE 特殊レジスターで、入力ファイルに最も頻繁に出現するレコード長 (モジュール長) を指定します。

関連タスク

[259 ページの『制御ステートメントによる DFSORT デフォルトの変更』](#)

[258 ページの『ソート動作の制御』](#)

ソートまたはマージの要求

入力ファイル (MERGE) を予備処理せずに使用する場合は、SORT . . . USING または MERGE . . . USING および、宣言した入力ファイル (ファイル) の名前 SELECT 句。

このタスクについて

ソートまたはマージされたレコードをソートまたはマージプログラムから他のファイルに転送して、それ以上の処理を行わない場合は、SORT . . . GIVING または MERGE . . . GIVING および、SELECT 句。例:

```
SORT Sort-Work-1
  ON ASCENDING KEY Sort-Key-1
  USING Input-File
  GIVING Output-File.
```

のために SORT . . . USING または MERGE . . . USING コンパイラは、ファイル (files) を開き、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 プログラムに解放し、ファイル (files) を閉じるための入力プロシージャを生成します。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。のために SORT . . . GIVING または MERGE . . . GIVING コンパイラは、ファイルを開き、レコードを返し、レコードを書き込んで、ファイルを閉じる出力プロシージャを生成します。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 または MERGE ステートメントの USING または GIVING ファイルは、z/OS UNIX ファイルシステムに常駐する順次ファイルにすることができます。

[246 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』](#)

FASTSORT コンパイラー・オプションを使用すれば、Enterprise COBOL コンパイラーの代わりに、IBM DFSORT または同等の製品でソート入出力を (Language Environment ランタイムを使用して) 実行することもできます。FASTSORT を使用すると、ほとんどのソート操作のパフォーマンスが向上します。詳しくは、[402 ページの『FASTSORT』](#) を参照してください。

ソートされる前のソート・レコードに対して入力プロシージャーを実行するには、SORT . . . INPUT PROCEDURE を使用します。ソートされたレコードに対して出力手順を実行する場合は、SORT . . . OUTPUT PROCEDURE。例:

```
SORT Sort-Work-1
  ON ASCENDING KEY Sort-Key-1
  INPUT PROCEDURE EditInputRecords
  OUTPUT PROCEDURE FormatData.
```

253 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、マージされたレコードに対して出力手順を実行する場合は、MERGE . . . OUTPUT PROCEDURE。例:

```
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。

関連タスク

250 ページの『ソート・データ・セットおよびマージ・データ・セットの定義』

関連参照

SORT ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

MERGE ステートメント (*Enterprise COBOL for z/OS 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。

このタスクについて

注: このトピックで説明されているソート基準設定のプロセスは、形式 1 SORT ステートメントにのみ関連します。形式 2 SORT ステートメントを使用したテーブルのソートについて詳しくは、[94 ページの『テーブルのソート』](#)を参照してください。

以下の手順を 実行します。

手順

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。

キーの最大数は規定されていませんが、キーはレコード記述の最初の 4092 バイト内になければなりません。キーの全長が 4092 バイトを超えてはなりません。ただし、EQUALS キーワードが DFSORT OPTION 制御ステートメントにコーディングされている場合は別で、その場合はキーの全長が 4088 バイトを超えてはなりません。

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。

タスクの結果

SORT そして MERGE キーはアルファベット、英数字、国別、または数値 (ただし、USAGE NATIONAL)。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。国別データ項目をキーとして指定すると、COLLATING SEQUENCE フレーズの SORT または MERGE ステートメントはそのキーには適用されません。

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

[94 ページの『テーブルのソート』](#)

関連参照

DFSORT アプリケーション・プログラミング・ガイド (SORT 制御ステートメント)

SORT ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

MERGE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

例: 入出力プロシージャーを使用したソート

以下は、形式 1 SORT ステートメントにおける入出力プロシージャーの使用例です。この例では、基本キー (SORT-GRID-LOCATION) と 2 次キー (SORT-SHIFT) を形式 1 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.
. . .
*****
* This SORT statement will do a 'table sort' using Format 1 SORT, and will *
* sort records in SORT-FILE in an ascending sequence based on the data *
* in sort keys SORT-GRID-LOCATION (primary) and SORT-SHIFT (secondary). *
* The source of the file records is data in table TAB-ENTRY *
* which is acquired in the input procedure 600-SORT3-INPUT and then *
* output back to the table in the output procedure 700-SORT3-OUTPUT. *
*****
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

[251 ページの『ソートまたはマージの要求』](#)

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、EBCDIC または ASCII 照合シーケンス、または別の照合シーケンスで行うことができます。デフォルトの照合順序は EBCDIC コーディングしない限り PROGRAM COLLATING SEQUENCE 条項の OBJECT-COMPUTER 段落。

このタスクについて

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。

ASCII ファイルをソートまたはマージするときには、ASCII 照合シーケンスを要求しなければなりません。そうするには、SORT または MERGE ステートメントの COLLATING SEQUENCE 句をコーディングし、*alphabet-name* を SPECIAL-NAMES 段落の STANDARD-1 として定義してください。

関連タスク

[7 ページの『照合シーケンスの指定』](#)

[252 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 (*Enterprise COBOL for z/OS 言語解説書*)

SORT ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

データのクラスおよびカテゴリー (*Enterprise COBOL for z/OS 言語解説書*)

同じキーを持つレコードのオリジナル・シーケンスの保持

このタスクについて

同じ照合レコードの順序を入力から出力まで保持することができます。

以下のいずれかの手法を使用します。

- EQUALS オプションをデフォルトとして指定して DFSORT をインストールします。
- 実行時に、IGZSRTCD データ・セットに EQUALS キーワードがある OPTION カードを提供します。
- SORT ステートメントで WITH DUPLICATES IN ORDER 句を使用します。これにより、IGZSRTCD データ・セットの OPTION カードに EQUALS キーワードが追加されます。

OPTION カードで NOEQUALS キーワードと DUPLICATES 句の両方は使用しないでください。そうしなければ、実行単位が終了します。

関連参照

DFSORT アプリケーション・プログラミング・ガイド (OPTION 制御ステートメント)

ソートまたはマージの成否の判断

の DFSORT プログラム各ソートまたはマージが終了すると、0 (正常完了) または 16 (異常完了) 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。

このタスクについて

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。
例:

```
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発行します。

デフォルトでは、DFSORT 診断メッセージは SYSOUT データ・セットに送られます。このデフォルトを変更したい場合は、DFSORT の OPTION 制御カードの MSGDDN パラメーターを使用するか、SORT-MESSAGE 特殊レジスターを使用してください。

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。

関連タスク

[258 ページの『NOFASTSRT によるソート・エラーの検査』](#)

[258 ページの『ソート動作の制御』](#)

関連参照

DFSORT アプリケーション・プログラミング・ガイド (DFSORT メッセージおよび戻りコード)

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。

このタスクについて

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、次の暗黙の RELEASE または RETURN (USING または GIVING ファイルに対して、あるレコードの読み取りまたは書き込みの後で起こる) が実行された直後に停止されます。

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。

FASTSRT によるソート・パフォーマンスの向上

FASTSRT コンパイラー・オプションを使用すると、ほとんどのソート操作のパフォーマンスが向上します。と FASTSRT、DFSORT 製品 (Enterprise COBOL) は、指定した入力ファイルと出力ファイルに対して I/O を実行します。SORT . . . USING そして SORT . . . GIVING 声明。

このタスクについて

コンパイラーは通知メッセージを出し、FASTSRT がパフォーマンスを向上させることができるステートメントを指摘します。

使用上の注意

- FASTSRT を使用する場合は、DFSORT オプション SORTIN または SORTOUT は使用できません。
FASTSRT コンパイラー・オプションは、USING または GIVING ファイルとして使用している行順次ファイルには適用されません。
- FASTSRT を使用する場合は、ファイル状況を指定しても、ソート操作中はファイル状況は無視されます。

関連参照

[402 ページの『FASTSRT』](#)

[256 ページの『JCL に関する FASTSRT の要件』](#)

[256 ページの『ソート入出力ファイルに関する FASTSRT の要件』](#)

JCL に関する FASTSRT の要件

実行時 JCL では、ソート作業ファイル (SORTWKnn) を、テープ・データ・セットではなく、直接アクセス装置に割り当てをしてください。

入出力ファイルの場合、DD ステートメントの DCB パラメーターは FD 記述と一致していなければなりません。

ソート入出力ファイルに関する FASTSRT の要件

FASTSRT を指定しても、FASTSRT の要件が満たされていないと、コンパイラーはメッセージを出し、代わりに COBOL ランタイムが入出力を実行します。その場合、プログラムはパフォーマンスの向上を望めなくなります。

注：このトピックで説明されている"入力ファイルおよび出力ファイルのソート"は、形式 1 SORT ステートメントにのみ関連します。

FASTSRT を使用するためには、ソートへの入力ファイルおよびソートからの出力ファイルを、次のように記述し、処理してください。

- USING 句では 1 つの入力ファイルのみ指定することができます。GIVING 句では 1 つの出力ファイルのみ指定することができます。
- 入力ファイルに対して入力プロシージャを使用したり、出力ファイルに対して出力プロシージャを使用したりすることはできません。

入力または出力プロシージャを使用する代わりに、次の DFSORT 制御ステートメントを使用できます。

- INREC
- OUTFILE
- OUTREC
- INCLUDE
- OMIT
- STOPAFT
- SKIPREC
- SUM

多くの DFSORT の機能では、入力または出力プロシージャでよく使用されるのと同様の操作を実行します。代わりに適切な DFSORT 制御ステートメントをコーディングし、それらを IGZSRTCD または SORTCNTL データ・セットに入れてください。

- 出力 FD 記入項目に LINAGE 節をコーディングしないでください。

- ソートで使用する FD に適用するために、INPUT 宣言 (入力ファイル用)、OUTPUT 宣言 (出力ファイル用)、またはファイル特定宣言 (入力ファイルまたは出力ファイル用) をコーディングしてはなりません。
- 可変長の相対ファイルを入力ファイルまたは出力ファイルとして使用してはなりません。
- 行順次ファイルを入力ファイルまたは出力ファイルとして使用してはなりません。
- 入力ファイルまたは出力ファイルのいずれかについて、SD と FD 記入項目のレコード記述では、両方とも同じ形式 (固定長または可変長) を定義しなければなりません。また、SD と FD の最大レコード・サイズでは、同じレコード長を定義しなければなりません。

出力ファイルに RELATIVE KEY 節をコーディングしても、これはソートでは設定されません。

パフォーマンスのヒント: 入出力レコードをブロック化すると、ソート処理のパフォーマンスが大幅に向上する可能性があります。

QSAM の要件

- QSAM ファイルのレコード形式は、固定長、可変長、またはスパンでなければなりません。
- QSAM 入力ファイルは空であっても構いません。
- 入力と出力の両方に同じ QSAM ファイルを使用する場合は、2 つの異なる DD ステートメントを使用してこのファイルを記述しなければなりません。例えば、FILE-CONTROL SECTION では、次のようにコーディングすることができます。

```
SELECT FILE-IN ASSIGN INPUTF.
SELECT FILE-OUT ASSIGN OUTPUTF.
```

DATA DIVISION で、FILE-IN と FILE-OUT の両方についての FD 記入項目を持つことになります。この場合、FILE-IN と FILE-OUT は、それらの名前を除いて同じです。

PROCEDURE DIVISION では、SORT ステートメントを次のように記述します。

```
SORT file-name
  ASCENDING KEY data-name-1
  USING FILE-IN GIVING FILE-OUT
```

そうすると JCL では、データ・セット INOUT がカタログされていると想定して、次のようにコーディングします。

```
//INPUTF DD DSN=INOUT,DISP=SHR
//OUTPUTF DD DSN=INOUT,DISP=SHR
```

これに反して、USING 句と GIVING 句に同じファイル名をコーディングするか、または入力ファイルと出力ファイルに同じ DD 名を割り当てた場合は、そのファイルは入力と出力のいずれかについて FASTSRT に受け入れられますが、両方については受け入れられません。ファイルが入力について FASTSRT に不適格となるような条件が特になければ、ファイルは入力について FASTSRT に受け入れられますが、出力については受け入れられません。ファイルが入力について FASTSRT に不適格である場合でも、出力について FASTSRT に適格である可能性があります。

FASTSRT に対して適格である QSAM ファイルは、形式 1 SORT ステートメントの実行中に COBOL プログラムからアクセスすることができます。例えば、ファイルが入力で FASTSRT に使用されている場合には、それを出力プロシージャーでアクセスすることができ、ファイルが出力で FASTSRT に使用されている場合には、それを入力プロシージャーでアクセスすることができます。

VSAM の要件

- VSAM 入力ファイルは空であってはなりません。
- VSAM ファイルをパスワード保護することはできません。
- USING 句と GIVING 句の両方で同じ VSAM ファイルを指定することはできません。

- FASTSORT に対して適格である VSAM ファイルは、形式 1 SORT ステートメントの処理が完了するまで、COBOL プログラムからアクセスすることはできません。例えば、ファイルが入力で FASTSORT に適格である場合には、それを出力プロシージャでアクセスしてはなりません。逆も同様です。（そのような処理を行うと、OPEN は失敗します。）

関連タスク

DFSORT アプリケーション・プログラミング・ガイド

NOFASTSORT によるソート・エラーの検査

NOFASTSORT オプションを使用してコンパイルすると、ソート・プロセスは、形式 1 SORT ステートメントの USING または GIVING 句で参照されるファイルに関して、オープン、クローズ、または入出力操作時にエラーの有無を検査しません。このため、SORT が正常に完了したかどうかを調べる必要がある場合があります。

このタスクについて

注：このトピックは形式 1 SORT ステートメントにのみ関連します。

必要なコードは、次の表に示されているように、USING および GIVING 句で参照されるファイルに FILE STATUS 節または ERROR 宣言がコーディングされているかどうかによって異なります。

表 34. NOFASTSORT によるソート・エラーの検査のメソッド		
FILE STATUS 節の有無	ERROR 宣言の有無	必要なコード
いいえ	いいえ	特別なコーディングは不要です。ソート・プロセス時に障害が起これば、プログラムは異常終了します。
はい	いいえ	SORT ステートメントの後で、形式 1 SORT-RETURN 特殊レジスターをテストし、ファイル状況キーをテストします。（ファイル状況コードは設定されますが、COBOL はこれを検査できないため、ファイル状況検査を完了したい場合にはお勧めしません。）
どちらでも可	はい	ERROR 宣言で、SORT-RETURN 特殊レジスターを 16 に設定して、ソート・プロセスを停止し、それが失敗したことを示します。SORT ステートメントの後で、形式 1 SORT-RETURN 特殊レジスターをテストします。

関連タスク

[254 ページの『ソートまたはマージの成否の判断』](#)

[271 ページの『ファイル状況キーの使用』](#)

[270 ページの『ERROR 宣言のコーディング』](#)

[255 ページの『ソートまたはマージ操作の途中停止』](#)

ソート動作の制御

ソートの前に特殊レジスターに値を挿入するか、またはコンパイラー・オプションを使用することにより、ソート動作の幾つかの局面を制御できます。また、制御ステートメントやキーワードの選択を行える場合もあります。

このタスクについて

ソートの後で特殊レジスターの内容を調べることによって、ソート動作を検査することができます。

以下の表は、特殊レジスターまたはコンパイラー・オプション、および同等のソート制御ステートメント・キーワード（使用可能な場合）を使用して影響を及ぼすことができるソート動作の局面をリストしています。

表 35. ソート動作を制御するためのメソッド		
設定またはテスト対象	特殊レジスタまたはコンパイラ・オプションを使用する場合	制御ステートメント (該当する場合は、およびキーワード) を使用する場合
予約される主記憶域の量	SORT-CORE-SIZE 特殊レジスタ	OPTION (キーワード RESINV)
使用される主記憶域の量	SORT-CORE-SIZE 特殊レジスタ	OPTION (キーワード MAINSIZE または MAINSIZE=MAX)
可変長レコードを持つファイルのレコードのモダル長	SORT-MODE-SIZE 特殊レジスタ	SMS=nnnnn
ソート制御ステートメント・データ・セットの名前 (デフォルト IGZSRTCD)	SORT-CONTROL 特殊レジスタ	なし
ソート・メッセージ・ファイルの名前 (デフォルト SYSOUT)	SORT-MESSAGE 特殊レジスタ	OPTION (キーワード MSGDDN)
ソート・レコードの数	SORT-FILE-SIZE 特殊レジスタ	OPTION (キーワード FILSZ)
ソート完了コード	SORT-RETURN 特殊レジスタ	なし

ソート特殊レジスタ: SORT-CONTROL は、ソート制御ステートメント・ファイルの DD 名が入られる、8 文字の COBOL 特殊レジスタです。IGZSRTCD というデフォルトの DD 名を使用したくない場合は、ソート制御ステートメントが含まれているデータ・セットの DD 名を SORT-CONTROL に割り当ててください。

SORT-CORE-SIZE、SORT-FILE-SIZE、SORT-MESSAGE、および SORT-MODE-SIZE 特殊レジスタは、それらにデフォルト以外の値を割り当てた場合には、SORT インターフェースで使用されます。しかし、実行時には、ソート制御ステートメント・データ・セット内の制御ステートメントのパラメータは、特殊レジスタ内の対応する設定をオーバーライドし、その影響に対するメッセージが出されます。

SORT-RETURN 特殊レジスタを使用すると、ソートまたはマージが正常に実行されたかどうかを判断したり、ソートまたはマージ操作を途中で停止することができます。

プログラム内に設定したソート特殊レジスタごとに、コンパイラ警告メッセージ (W レベル) が出されます。

関連タスク

[254 ページの『ソートまたはマージの成否の判断』](#)

[255 ページの『ソートまたはマージ操作の途中停止』](#)

[259 ページの『制御ステートメントによる DFSORT デフォルトの変更』](#)

[261 ページの『ソート・ファイル用のスペースの割り振り』](#)

[DFSORT アプリケーション・プログラミング・ガイド \(DFSORT プログラム制御ステートメントの使用\)](#)

関連参照

[260 ページの『IGZSRTCD データ・セットのデフォルトの特性』](#)

制御ステートメントによる DFSORT デフォルトの変更

ソート・パフォーマンスを向上させるために DFSORT のシステム・デフォルトを変更したい場合には、実行時データ・セット IGZSRTCD に含めた制御ステートメントを介して DFSORT に情報を渡してください。

このタスクについて

(リストされた順序で) IGZSRTCD に含めることができる制御ステートメントは、次のとおりです。

手順

1. SMS=nnnnn。nnnnn は、最も出現頻度の高いレコード・サイズの長さ (バイト単位) です。 (SD ファイルが可変長である場合にのみ使用します。)
2. OPTION (キーワード SORTIN または SORTOUT を除く)。
3. その他の DFSORT 制御ステートメント (SORT、MERGE、RECORD、END を除く)。

タスクの結果

制御ステートメントは、2 桁目から 71 桁目の間にコーディングしてください。行をコンマで終わらせ、次の行を新しいキーワードで開始すれば、制御ステートメント・レコードを継続することができます。レコードではラベルもコメントも使用することはできず、レコード自体を DFSORT コメント・ステートメントにすることはできません。

関連タスク

[258 ページの『ソート動作の制御』](#)

DFSORT アプリケーション・プログラミング・ガイド (DFSORT プログラム
制御ステートメントの使用)

関連参照

[260 ページの『IGZSRCTD データ・セットのデフォルトの特性』](#)

IGZSRCTD データ・セットのデフォルトの特性

IGZSRCTD データ・セットは任意指定です。デフォルトは、LRECL=80、BLKSIZE=400、および DD 名 IGZSRCTD です。

SORT-CONTROL 特殊レジスターにコーディングすることによって、別の DD 名を使用することができます。SORT-CONTROL データ・セットの DD 名を定義したときに、メッセージ IGZ0027W を受け取った場合は、OPEN 障害が起こっており、問題を調べる必要があります。

関連タスク

[258 ページの『ソート動作の制御』](#)

ソートまたはマージ操作のためのストレージの割り振り

このタスクについて

DFSORT のインストール時に設定される特定のパラメーターによって、DFSORT が使用するストレージの量が決まります。一般的に、DFSORT が使用できるストレージが多いほど、ソートまたはマージ操作の実行速度が速くなります。

しかし、DFSORT インストールで、領域内のすべてのフリー・スペースを COBOL 操作用に割り振ってはいません。プログラムの実行時には、以下のために使用できるストレージが必要です。

- 入力または出力プロシージャーから動的に呼び出される COBOL プログラム
- 言語環境のランタイム・ライブラリー・モジュール
- 入力または出力プロシージャーで使用するために領域にロードされる可能性があるデータ管理モジュール
- これらのモジュールによって獲得されるストレージ

ソートまたはマージのある特定の実行のために、インストール時に設定された DFSORT 保管値をオーバーライドすることができます。これを行うには、ソート制御ステートメント・データ・セット内の OPTION 制御ステートメントに MAINSIZE および RESINV キーワードをコーディングするか、または SORT-CORE-SIZE 特殊レジスターを使用してください。

領域内のすべてのフリー・スペースが完全に COBOL プログラムのソート操作に使用されるかぎり、ストレージ割り振りをオーバーライドすることがないように注意してください。

関連タスク

[258 ページの『ソート動作の制御』](#)
[DFSORT 導入およびカスタマイズ](#)

関連参照

[DFSORT アプリケーション・プログラミング・ガイド \(OPTION 制御ステートメント\)](#)

ソート・ファイル用のスペースの割り振り

NOFASTSRT または入力プロシージャを使用する場合は、ソートするファイルのサイズを DFSORT は認識しません。このため、大きいファイルをソートするときにスペース不足状態に陥ったり、小さいファイルをソートするときにリソースの過剰割り振りが発生する可能性があります。

このタスクについて

このような状況が発生した場合は、SORT-FILE-SIZE 特殊レジスターを使用して、ソートに必要なリソースの量 (例えば、ワークスペースまたは ハイパースペース) を DFSORT が判別できるようにします。SORT-FILE-SIZE を、妥当な入力レコード見積数に設定します。この値は、FILSZ=En の値として DFSORT に渡されます。

関連タスク

[258 ページの『ソート動作の制御』](#)
[247 ページの『入力プロシージャのコーディング』](#)
[DFSORT アプリケーション・プログラミング・ガイド](#)

DFSORT によるチェックポイント・リスタートの使用

z/OS のもとで DFSORT の実行中に取られたチェックポイントは、DFSORT によって取られたものでない限り、再始動には使用できません。

このタスクについて

SORT または MERGE ステートメントの実行中に COBOL プログラムによって取られたチェックポイントは無効です。すなわち、そのような再始動は検出されますが、取り消されます。

ソートまたはマージ操作中にチェックポイントを取るには、以下のステップを行ってください。

手順

1. SORTCKPT についての DD ステートメントを JCL に追加します。
2. I-O-CONTROL 段落で RERUN 節をコーディングします。

```
RERUN ON assignment-name
```

3. ソート制御ステートメント・データ・セット (デフォルトの DD 名 IGZSRTCD) 内の OPTION 制御ステートメントで CKPT (または CHKPT) キーワードをコーディングします。

タスクの結果

関連概念

[823 ページの『第 48 章 割り込みおよびチェックポイント・リスタート』](#)

関連タスク

[259 ページの『制御ステートメントによる DFSORT デフォルトの変更』](#)
[823 ページの『チェックポイントの設定』](#)

DFSORT での COBOL ユーザー出口ルーチンの使用

COBOL で DFSORT の E15 および E35 のユーザー出口ルーチンを作成することができます。COBOL のユーザー出口ルーチン・インターフェースについて詳しくは、[z/OS DFSORT アプリケーションプログラミングガイド](#) の [ユーザー独自のユーザー出口ルーチンの使用](#) を参照してください。

このタスクについて

以下の規則が適用されることに注意してください。

- COBOL ルーチンには SORT ステートメントも MERGE ステートメントも含めないでください。
- STOP RUN を使用して COBOL ルーチンを戻さないでください。
- DFSORT に返す戻りコードは RETURN-CODE 特殊レジスターを使用して設定する必要があります。
- DFSORT は汎用レジスターの上位半分を使用する可能性があります。COBOL 出口ルーチンが Enterprise COBOL 5 以降のバージョンでコンパイルされている場合は、HGPR(PRESERVE) コンパイラー・オプションを使用する必要があります。これは HGPR オプションのデフォルト設定です。

タスクの結果

関連参照

405 ページの『HGPR』

RETURN-CODE (*Enterprise COBOL for z/OS* 言語解説書)

CICS のもとでのソート

CICS のもとでサポートされる IBM ソート・プロダクトはありません。ただし、フォーマット 1SORT ステートメントは、以下で実行されるソートプログラムで記述します。CICS 少量のデータを並べ替える。フォーマット 2 を使用することもできます SORT 声明文 CICS テーブルを並べ替えます。

このタスクについて

形式 1 SORT ステートメントには、入力プロシージャと出力プロシージャの両方を指定しなければなりません。入力プロシージャでは、RELEASE ステートメントを使用して、ソートが実行される前にレコードを COBOL プログラムからソート・プログラムに転送します。出力プロシージャでは、RETURN ステートメントを使用して、ソートが実行された後でレコードをソート・プログラムから COBOL プログラムに転送します。

フォーマット 2SORT 声明は以下でサポートされています CICS 特別なことを書く必要はありません SORT プログラム。

関連タスク

247 ページの『入力プロシージャのコーディング』

248 ページの『出力プロシージャのコーディング』

545 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

262 ページの『CICS SORT アプリケーションの制約事項』

553 ページの『CICS 予約語テーブル』

CICS SORT アプリケーションの制約事項

COBOL アプリケーションにはいくつかの制限が適用される。CICS そして、フォーマット 1SORT 声明。

制約事項は以下のとおりです。

- USING または GIVING 句を含む形式 1 SORT ステートメントはサポートされません。
- ソート制御データ・セットはサポートされません。SORT-CONTROL 特殊レジスターのデータは無視されます。

- 入力または出力プロシージャで以下の CICS コマンドを使用すると、予測できない結果をもたらす可能性があります。

- CICS LINK
- CICS XCTL
- CICS RETURN
- CICS HANDLE
- CICS IGNORE
- CICS PUSH
- CICS POP

上記以外の CICS コマンドは、NOHANDLE または RESP オプションを使用する場合には、使用できます。NOHANDLE または RESP を使用しないと、予測できない結果が生じる恐れがあります。

関連参照

553 ページの『[CICS 予約語テーブル](#)』

第 14 章 エラーの処理

このタスクについて

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。

Enterprise COBOL 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。

- ユーザー要求ダンプ
- 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
- ON EXCEPTION または ON OVERFLOW で CALL 声明
- エラー処理用のユーザー作成ルーチン

関連タスク

[266 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』](#)

[266 ページの『算術演算でのエラーの処理』](#)

[267 ページの『入出力操作でのエラーの処理』](#)

[275 ページの『プログラム呼び出し時のエラーの処理』](#)

[276 ページの『エラー処理用のルーチンの作成』](#)

ダンプの要求

このタスクについて

言語環境呼び出し可能サービス CEE3DMP への呼び出しをコーディングすることによって、プログラム内の事前に指定した任意のポイントで、言語環境 ランタイム環境およびメンバー言語ライブラリーの定様式ダンプを取ることができます。

```
77 Title-1          Pic x(80)   Display.  
77 Options          Pic x(255)  Display.  
01 Feedback-code    Pic x(12)   Display.  
    .  
    .  
    . Call "CEE3DMP" Using Title-1, Options, Feedback-code
```

定様式ダンプに記号変数を組み込むためには、TEST コンパイラー・オプションを使用してコンパイルし、CEE3DMP の VARIABLES サブパラメーターを使用します。さらに、ランタイム・オプションを介して、選択したエラー状態に合ったダンプを作成するように要求することができます。

プログラム内の事前に指定した任意のポイントでシステム・ダンプを取ることができます。ゼロのクリーンアップ値を指定した言語環境サービス CEE3ABD を呼び出して、クリーンアップなしの異常終了を要求します。この呼び出し可能サービスは実行単位を即刻終了させ、異常終了が出されると、システム・ダンプが要求されます。

関連参照

448 ページの『TEST』

言語環境デバッグガイド

言語環境プログラミング・リファレンス (CEE3DMP 生成ダンプ)

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。

このタスクについて

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。コーディングしない場合は ON OVERFLOW のフレーズ STRING または UNSTRING ステートメントでは、不完全な操作は通知されません。

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)	bbbbbbbb ¹	bbbbbbbb 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 節でその状況进行处理することができます。

このタスクについて

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 が負数乗された
- 負数の分数乗

浮動小数点指数オーバーフローは、浮動小数点算術計算の値を System z の浮動小数点オペランド形式で表すことができない場合に起こります。このタイプのオーバーフローは SIZE ERROR を起こしません。代わりに、異常終了が起こります。異常終了を代行受信し、独自のエラー・リカバリー論理を提供するために、ユーザー作成条件処理ルーチンをコーディングすることができます。

例: 0 による除算の検査

次の例は、ON SIZE ERROR ゼロ除算が発生した場合にプログラムが情報メッセージを発行するように命令文を作成します。

```
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。

このタスクについて

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。

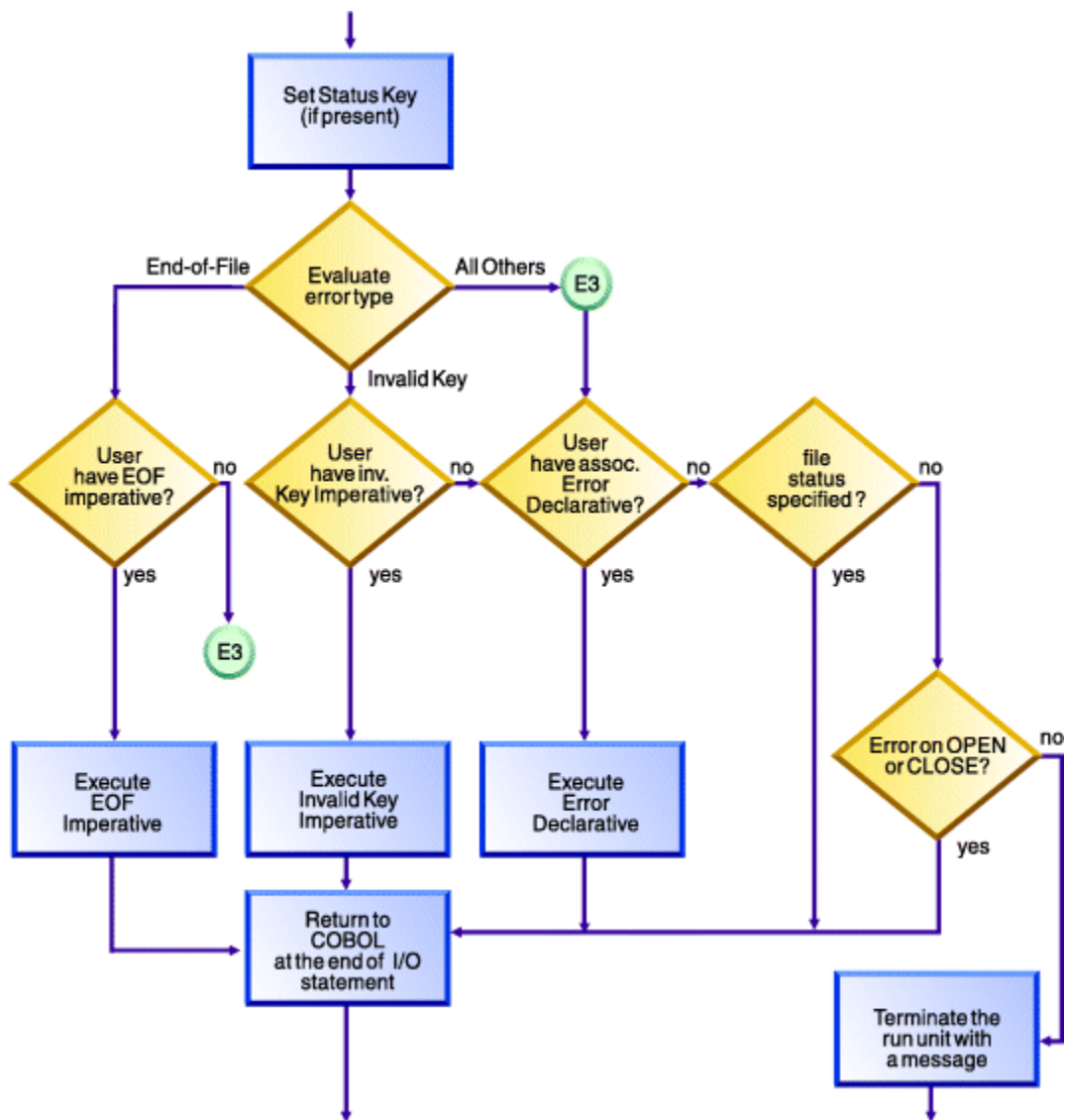
- ファイル終わり条件 (AT END)
- ERROR 宣言
- FILE STATUS 節およびファイル状況キー
- ファイル・システム状況コード
- READ または WRITE ステートメント内の命令ステートメント句

VSAM ファイルについては、FILE STATUS 節を指定した場合は、VSAM 状況コードをテストして、エラー処理論理に対してプログラムを指示することもできます。

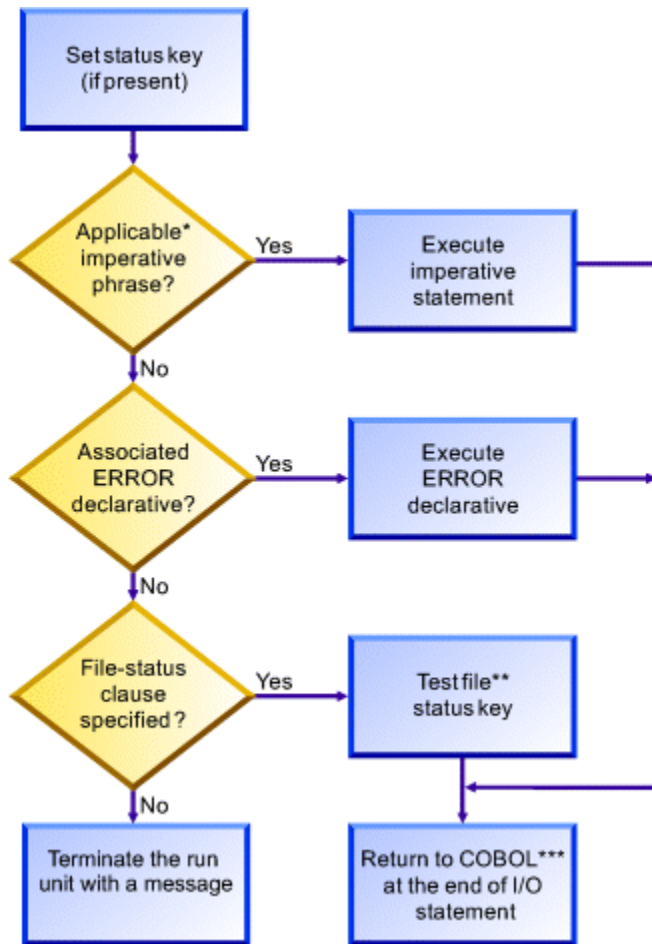
- INVALID KEY 句

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、重大度 3 の言語環境条件がシグナル通知され、この条件が処理されない場合には、実行単位が終了します。

次の図は、VSAM 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。



次の図は、QSAM または行順次ファイルの入力または出力エラー後のロジック・フローを示しています。このエラーは、REEL/UNIT 節が指定された READ ステートメント、WRITE ステートメント、または CLOSE ステートメントから生じた可能性があります (QSAM のみ)。



*QSAM の可能なフレーズは AT END、AT END-OF-PAGE、そして INVALID KEY; 行順次の場合、AT END。

**ファイル状況キーをテストするコードを書く必要があります。

*** COBOL プログラムの実行は、エラーを引き起こした入出力ステートメントの後で継続されます。

関連タスク

[269 ページの『ファイルの終わり条件 \(AT END\) の使用』](#)

[270 ページの『ERROR 宣言のコーディング』](#)

[271 ページの『ファイル状況キーの使用』](#)

[190 ページの『QSAM ファイルのエラーの処理』](#)

[273 ページの『VSAM 状況コードの使用 \(VSAM ファイルのみ\)』](#)

[241 ページの『行順次ファイルのエラーの処理』](#)

[275 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 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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 条件が期待されています。例えば、メイン・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进行处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。もし、READ ファイルの終わり以外の条件により操作が失敗した場合、AT END また、NOT AT END フレーズが演奏されます。その代わりに、関連する宣言型プロシージャーを実行した後で、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ーのいずれもコーディングせずに、代わり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します。

関連参照

AT END 句 (*Enterprise COBOL for z/OS 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ーを 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ーをコーディングしないと、入力または出力エラーの発生後に、ジョブが取り消されるか、異常終了します。

このタスクについて

このようなプロシージャーをそれぞれ 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ー
- 各ファイル・オープン・モードのプロシージャー (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ー

ERROR 宣言プロシージャーでは、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ーとファイル状況キーを組み合わせ使用することができます。

マルチスレッド化: マルチスレッドのアプリケーションで入出力宣言をコーディングするときは、デッドロックを避けます。入出力操作の結果として制御が入出力宣言に移動した場合、その宣言内のステートメントが実行されている間は、そのファイルと関連付けられた自動逐次化ロックが保持されます。宣言内に入出力操作をコーディングすると、ロジックは以下のサンプルに示すデッドロックに陥ることがあります。

```
Declaratives.  
D1 section.  
  Use after standard error procedure on F1  
  Read F2.  
.  
D2 section.  
  Use after standard error procedure on F2
```

```
Read F1.  
.....  
End declaratives.  
.....  
Rewrite R1.  
Rewrite R2.
```

このプログラムを2つのスレッドで実行している場合、次の一連のイベントが発生するおそれがあります。

手順

1. スレッド 1: 再書き込み R1 が F1 に対するロックを獲得し、入出力エラーが発生する。
2. スレッド 1: F1 に対するロックを保持したまま、宣言 D1 に入る。
3. スレッド 2: 再書き込み R2 が F2 に対するロックを獲得し、入出力エラーが発生する。
4. スレッド 2: 宣言 D2 に入る。
5. スレッド 1: 宣言 D1 から F2 を読み取り、スレッド 2 によって保持されている F2 のロックに対し待機する。
6. スレッド 2: 宣言 D2 から F1 を読み取り、スレッド 1 によって保持されている F1 ロックに対し待機する。
7. デッドロック。

タスクの結果

関連参照

EXCEPTION/ERROR 宣言 (*Enterprise COBOL for z/OS* 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。

このタスクについて

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
```

- データ定義 DATA DIVISION (WORKING-STORAGE、LOCAL-STORAGE、または LINKAGE SECTION)、例えば:

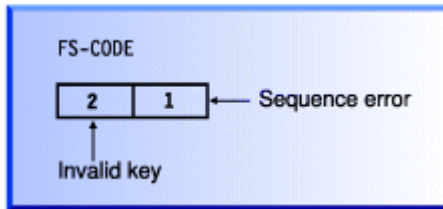
```
WORKING-STORAGE SECTION.  
01 data-name-1 PIC 9(2) USAGE NATIONAL.
```

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。

- 各入出力要求後に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。

VSAM ファイルの場合、さらに 2 番目の ID を FILE STATUS 節にコーディングして、VSAM 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。

ファイルステータスキーは単独でも、INVALID KEY フレーズ、または補足として EXCEPTION または ERROR 宣言的。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。

[272 ページの『例: ファイル状況キー』](#)

関連タスク

[273 ページの『VSAM 状況コードの使用 \(VSAM ファイルのみ\)』](#)

[275 ページの『INVALID KEY 句のコーディング』](#)

[487 ページの『入出力エラーの検出および処理』](#)

関連参照

FILE STATUS 節 (*Enterprise COBOL for z/OS 言語解説書*)

ファイル状況キー (*Enterprise COBOL for z/OS 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 MAINFILE ASSIGN TO AS-MAINA
    FILE STATUS IS MAIN-CHECK-KEY

DATA DIVISION.

WORKING-STORAGE SECTION.
01 MAIN-CHECK-KEY PIC X(2).

PROCEDURE DIVISION.
    OPEN INPUT MAINFILE
    IF MAIN-CHECK-KEY NOT = "00"
        DISPLAY "Nonzero file status returned from OPEN " MAIN-CHECK-KEY
    . . .
```

VSAM 状況コードの使用 (VSAM ファイルのみ)

要求の処理を正確に特定する上で、COBOL 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 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、VSAM 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。

このタスクについて

```
FILE STATUS IS data-name-1 data-name-8
```

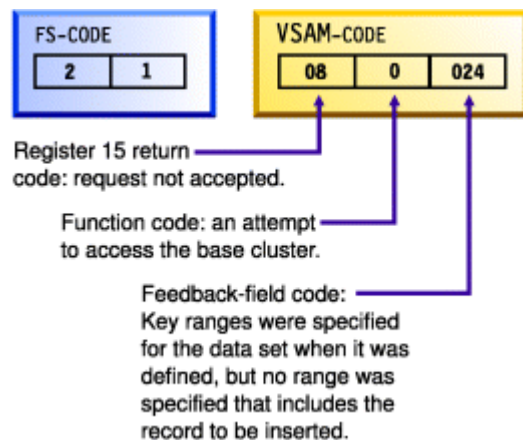
上記のデータ項目 *data-name-1* は COBOL ファイル状況キーを指定します。これは、2 文字の英数字または国別データ項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目として定義されます。

データ項目 *data-name-8* は VSAM 状況コードを指定します。これは、3 個の従属 2 バイト 2 進数フィールドを持つ、6 バイトの英数字グループ・データ項目として定義されます。VSAM 状況コードは、COBOL ファイル状況キーが 0 でないときに、意味のある値を含みます。

data-name-8 は、以下の VSAM-CODE の場合のように WORKING-STORAGE SECTION 内で定義できます。

```
01 RETURN-STATUS.  
  05 FS-CODE          PIC X(2).  
  05 VSAM-CODE.  
    10 VSAM-R15-RETURN PIC S9(4) Usage Comp-5.  
    10 VSAM-FUNCTION   PIC S9(4) Usage Comp-5.  
    10 VSAM-FEEDBACK   PIC S9(4) Usage Comp-5.
```

Enterprise COBOL は、*data-name-8* を使用して、VSAM から提供される情報を渡します。次の例では、FS-CODE が *data-name-1* に相当し、VSAM-CODE が *data-name-8* に相当します。



273 ページの『例: VSAM 状況コードの検査』

関連参照

FILE STATUS 節 (Enterprise COBOL for z/OS 言語解説書)

ファイル状況キー (Enterprise COBOL for z/OS 言語解説書)

データ・セットに対する z/OS DFSMS マクロ命令 (VSAM マクロの戻りコードおよび理由コード)

例: VSAM 状況コードの検査

次の例は、索引付きファイルを読み取り (5 番目のレコードで開始する)、入力または出力要求があるたびにその後でファイル状況キーを検査し、ファイル状況キーがゼロ以外であれば VSAM 状況コードを表示するものです。

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 VSAMFILE ASSIGN TO VSAMFILE
    ORGANIZATION IS INDEXED
    ACCESS DYNAMIC
    RECORD KEY IS VSAMFILE-KEY
    FILE STATUS IS FS-CODE VSAM-CODE.
DATA DIVISION.
FILE SECTION.
FD VSAMFILE
   RECORD 30.
01 VSAMFILE-REC.
   10 VSAMFILE-KEY          PIC X(6).
   10 FILLER                 PIC X(24).
WORKING-STORAGE SECTION.
01 RETURN-STATUS.
   05 FS-CODE                PIC XX.
   05 VSAM-CODE.
       10 VSAM-RETURN-CODE   PIC S9(2) Usage Binary.
       10 VSAM-COMPONENT-CODE PIC S9(1) Usage Binary.
       10 VSAM-REASON-CODE   PIC S9(3) Usage Binary.
PROCEDURE DIVISION.
    OPEN INPUT VSAMFILE.
    DISPLAY "OPEN INPUT VSAMFILE FS-CODE: " FS-CODE.

    IF FS-CODE NOT = "00"
        PERFORM VSAM-CODE-DISPLAY
        STOP RUN
    END-IF.

    MOVE "000005" TO VSAMFILE-KEY.
    START VSAMFILE KEY IS EQUAL TO VSAMFILE-KEY.
    DISPLAY "START VSAMFILE KEY=" VSAMFILE-KEY
           " FS-CODE: " FS-CODE.
    IF FS-CODE NOT = "00"
        PERFORM VSAM-CODE-DISPLAY
    END-IF.

    IF FS-CODE = "00"
        PERFORM READ-NEXT UNTIL FS-CODE NOT = "00"
    END-IF.

    CLOSE VSAMFILE.
    STOP RUN.

READ-NEXT.
    READ VSAMFILE NEXT.
    DISPLAY "READ NEXT VSAMFILE FS-CODE: " FS-CODE.
    IF FS-CODE NOT = "00"
        PERFORM VSAM-CODE-DISPLAY
    ELSE
        DISPLAY VSAMFILE-REC
    END-IF.

VSAM-CODE-DISPLAY.
    DISPLAY "VSAM-CODE ==>"
           " RETURN: " VSAM-RETURN-CODE,
           " COMPONENT: " VSAM-COMPONENT-CODE,
           " REASON: " VSAM-REASON-CODE.
```

以下に、VSAM 状況コード情報を検査するプログラム例からの出力例を示しています。

```
OPEN INPUT VSAMFILE FS-CODE: 00
START VSAMFILE KEY=000005 FS-CODE: 00
READ NEXT VSAMFILE FS-CODE: 00
000005 THIS IS RECORD NUMBER 5
READ NEXT VSAMFILE FS-CODE: 00
000006 THIS IS RECORD NUMBER 6
READ NEXT VSAMFILE FS-CODE: 10
VSAM-CODE ==> RETURN: 08 COMPONENT: 2 REASON: 004
```


INVALID KEY 句のコーディング

INVALID KEY 句は、VSAM 索引付きファイルおよび相対ファイルの、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。

このタスクについて

INVALID KEY 句は、QSAM ファイルに対する WRITE 要求に組み込むこともできますが、QSAM ファイルの場合、句の意味が限定されます。この句が使用されるのは、いっぱいになっているディスクに書き込もうとした場合に限られます。

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。
- 入出力ステートメントで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。

275 ページの『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。

メイン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、メイン・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、メイン・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。

顧客レコードのメイン・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 MAIN-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 MAIN-CUSTOMER-NUMBER
READ MAIN-CUSTOMER-FILE INTO WS-CUSTOMER-RECORD
INVALID KEY
    DISPLAY "MAIN 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、プログラム・オブジェクトを

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。

このタスクについて

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
```

の ON EXCEPTION このフレーズは、呼び出されたプログラムの可用性にのみ適用されます。初期ロード時に。呼び出されたプログラムがロードされたが、他の理由（初期化など）で失敗した場合、の ON EXCEPTION フレーズは実行されません。

関連参照

Enterprise COBOL for z/OS 移行ガイド

エラー処理用のルーチンの作成

ON EXCEPTION 句、ON SIZE ERROR 句、またはその他の言語構成要素を使用すると、プログラムの実行中に発生する可能性のあるエラー条件の大部分を処理することができます。しかし、マシン・チェックなどの異常条件が発生すると、通常、アプリケーションは異常終了します。

このタスクについて

しかし、Enterprise COBOL および 言語環境では、そのような条件が発生したときにユーザー作成プログラムが制御を獲得できる方法を提供しています。言語環境条件処理を使用して、独自のエラー処理ルーチンを COBOL で書くことができます。それらは、プログラムの報告、分析、あるいは修正でさえも行うことができ、プログラムが実行を再開できるようにします。

COBOL ランタイムからのほとんどのメッセージは、対応する条件を通知することによって生成されます。これは、エラー ハンドラが制御を取得することを意味します。I (情報) および W (警告) レベルの条件の場合、条件をパーコレートするだけで、ランタイムがメッセージを表示して実行を再開します。特に、これは、さまざまなコンパイラ生成ランタイム診断に必要であり、MSG オプション（例：SSRANGE(,MSG)）が意図したとおりに動作するようになります。

アプリケーション用に独自のエラー処理ルーチンを書く際には、COBOL プログラムを適切なコンパイラー・オプションを使用してコンパイルする必要があります。詳しくは、[429 ページの『OPTIMIZE』](#)を参照してください。

言語環境がユーザー作成のエラー・プログラムに制御を渡すようにさせるには、まずその入り口点を言語環境に対して示し、登録する必要があります。PROCEDURE-POINTER データ項目を使用して、プロシージャ入り口点の入り口アドレスを言語環境・サービスに渡すことができます。

関連タスク

[617 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

[429 ページの『OPTIMIZE』](#)

第 2 部 プログラムのコンパイルおよびデバッグ

第 15 章 z/OS のもとでのコンパイル

Enterprise COBOL プログラムは、z/OS のもとでは、ジョブ制御言語 (JCL)、TSO コマンド、CLIST、または ISPF パネルを使用してコンパイルすることができます。

このタスクについて

JCL を使ったコンパイルの場合、IBM は一連のカタログ式プロシージャーを用意しています。これを使用すれば、書かなければならない JCL コーディングの量を減らすことができます。カタログ式プロシージャーが要件に合わない場合は、独自の JCL を書くことができます。JCL を使用して、単一のプログラムをコンパイルすること、または複数のプログラムをバッチ・ジョブの中でコンパイルすることができます。

TSO のもとでコンパイルするときは、TSO コマンド、CLIST、または ISPF パネルを使用することができます。

また、cob2 コマンドを使用すると、z/OS ユニックス シェルでもコンパイルできます。

Enterprise COBOL コンパイラーを呼び出すツールまたはインターフェースを開発している場合には、Enterprise COBOL コンパイラーをアセンブラー・プログラムから開始してください。

コンパイル・ステップの一部として、コンパイルに必要なデータ・セットを定義し、プログラムに必要なコンパイラー・オプションおよび必要な出力を指定する必要があります。

コンパイラーは、COBOL プログラムを、コンピューターが処理できる言語 (オブジェクト・コード) に変換します。さらに、コンパイラーは、ソース・ステートメント内のエラーをリストして、プログラムのデバッグおよび調整に役立つ補足情報を提供します。コンパイルを制御するには、コンパイラー指示ステートメントおよびコンパイラー・オプションを使用してください。

プログラムをコンパイルした後、コンパイルの結果を検討して、コンパイラー検出エラーがあれば、それを訂正する必要があります。

関連タスク

[279 ページの『JCL を使用したコンパイル』](#)

[287 ページの『TSO のもとでのコンパイル』](#)

[311 ページの『第 16 章 z/OS ユニックス のもとでのコンパイル』](#)

[289 ページの『アセンブラー・プログラムからコンパイラーを開始する』](#)

[291 ページの『コンパイラー入出力の定義』](#)

[299 ページの『z/OS のもとでのコンパイラー・オプションの指定』](#)

[302 ページの『複数プログラムのコンパイル \(バッチ・コンパイル\)』](#)

[306 ページの『ソース・プログラムのエラーの訂正』](#)

関連参照

[481 ページの『第 21 章 コンパイラー指示ステートメント』](#)

[291 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』](#)

[301 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』](#)

JCL を使用したコンパイル

コンパイルで JCL の情報 (ジョブ記述、コンパイラーを呼び出すステートメント、および必要なデータ・セットの定義 (z/OS ユニックスファイルのディレクトリー・パス (存在する場合) など)) を組み込みます。

このタスクについて

z/OS のもとでプログラムをコンパイルする最も簡単な方法は、カタログ式プロシージャーを使用する JCL をコーディングすることです。カタログ式プロシージャーとは、プロシージャー・ライブラリー (SYS1.PROCLIB) と呼ばれる区分データ・セットにある一連のジョブ制御ステートメントです。

次の JCL は、カタログ式プロシージャを使用するための一般形式を示します。

```
//jobname JOB parameters
//stepname EXEC [PROC=]procname[, {PARM=|PARM.stepname=} 'options']
//SYSIN DD data-set parameters
           (source program to be compiled)
/*
//
```

カタログ式プロシージャを使用してオブジェクト指向のプログラムをコンパイルするときには、追加の考慮事項があります。

642 ページの『[例: プロシージャ型 DLL アプリケーションのサンプル JCL](#)』

関連タスク

280 ページの『[カタログ式プロシージャの使用](#)』

285 ページの『[プログラムをコンパイルするための JCL の作成](#)』

299 ページの『[z/OS のもとでのコンパイラ・オプションの指定](#)』

304 ページの『[バッチ・コンパイルでのコンパイラ・オプションの指定](#)』

640 ページの『[DLL を作成するためのプログラムのコンパイル](#)』

関連参照

291 ページの『[z/OS のもとでコンパイラによって使用されるデータ・セット](#)』

カタログ式プロシージャの使用

カタログ式プロシージャは、JCL の EXEC ステートメントに指定してください。

このタスクについて

例えば、次の JCL は、Enterprise COBOL プログラムをコンパイルし、必要なデータ・セットを定義するために、IBM 提供のカタログ式プロシージャ IGYWC を呼び出します。

```
//JOB1 JOB1
//STEPA EXEC PROC=IGYWC
//COBOL.SYSIN DD *
000100 IDENTIFICATION DIVISION
      * (the source code)
/*
```

ソース・コードの後の /* は省略できます。ソースコードがデータセットに保存されている場合は、SYSIN DD * データセットを記述する適切なパラメータを使用します。

これらのプロシージャは、z/OS の一部であるすべてのジョブ・スケジューラーで使用することができます。スケジューラーは、不要なパラメータを検出すると、それらを見捨てるか、または代替パラメータで置き換えます。

コンパイラ・オプションがプロシージャで明示的に指定されない場合、インストール時に設定されたデフォルト・オプションが適用されます。これらのデフォルト・オプションは、必須オプションを含む EXEC ステートメントを使用してオーバーライドすることができます。

対応する DD ステートメントをオーバーライドすることにより、z/OS UNIX ファイルシステム内のデータ・セットを指定できます。ただし、コンパイラユーティリティファイル (SYSUT バツ) およびライブラリのコピー (SYSLIB) は、MVs データセット。

カタログ式プロシージャの呼び出し、EXEC ステートメントのオーバーライドと追加、および DD ステートメントのオーバーライドと追加の詳細は、言語環境情報の中で記載されています。

関連タスク

言語環境プログラミングガイド

関連参照

[281 ページの『コンパイル・プロシージャ \(IGYWC\)』](#)

[282 ページの『コンパイルおよびリンク・エディット用プロシージャ \(IGYWCL\)』](#)

[283 ページの『コンパイル、リンク・エディット、実行用のプロシージャ \(IGYWCLG\)』](#)

[MVS プログラム管理: ユーザーズ・ガイドおよび解説書](#)

コンパイル・プロシージャ (IGYWC)

IGYWC は、プログラムをコンパイルする単一ステップのカタログ式プロシージャです。オブジェクト・モジュールを作成します。コンパイラーを呼び出す他のどのカタログ式プロシージャにおいても、コンパイル・ステップは類似しています。

次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD *          (or appropriate parameters)
```

コンパイルするプログラムでコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。例:

```
//COBOL.SYSLIB DD DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWC PROC LNGPRFX='IGY.V6R5M0',
//          LIBPREFIX='CEE'
//*
//*  COMPILE A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE  USAGE
//*  LNGPRFX   IGY.V6R5M0     PREFIX FOR LANGUAGE DATA SET NAMES
//*  LIBPRFX   CEE            PREFIX FOR LIBRARY DATA SET NAMES
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//*  CALLER MUST ALSO SUPPLY //COBOL.SYSLIB DD . . . for COPY statements
//*
//COBOL EXEC PGM=IGYCRCTL,REGION=0M
//STEPLIB DD DSN=DSNAME=&LNGPRFX..SIGYCOMP,DISP=SHR          (1)
//          DD DSN=DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//          DD DSN=DSNAME=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSN=DSNAME=&&LOADSET,UNIT=SYSALLDA,
//          DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT8 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT9 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT10 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT11 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT12 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT13 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT14 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT15 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSMDECK DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
```

(1)

STEPLIB は、インストール先によって異なります。

[282 ページの『例: z/OS UNIX ファイルシステムでのコンパイル用の JCL』](#)

例: z/OS UNIX ファイルシステムでのコンパイル用の JCL

以下のジョブは、プロシージャー IGYWC を使用して、z/OS UNIX ファイルシステム 内にある COBOL プログラム demo.cbl をコンパイルします。このジョブは、生成されたコンパイラー・リスト demo.lst、オブジェクト・ファイル demo.o、および SYSADATA ファイル demo.adt を z/OS UNIX ファイルシステム 内に書き込みます。

```
//UNIXDEMO JOB ,
// TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,REGION=50M,
// NOTIFY=&SYSUID,USER=&SYSUID
//COMPILE EXEC IGYWC,
// PARM.COBOL='LIST,MAP,RENT,FLAG(I,I),XREF,ADATA'
//SYSPRINT DD PATH='/u/userid/cobol/demo.lst',      (1)
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),              (2)
// PATHMODE=(SIRUSR,SIWUSR),                      (3)
// FILEDATA=TEXT                                  (4)
//SYSLIN DD PATH='/u/userid/cobol/demo.o',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR)
//SYSADATA DD PATH='/u/userid/cobol/demo.adt',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR)
//SYSIN DD PATH='/u/userid/cobol/demo.cbl',
// PATHOPTS=ORDONLY,
// FILEDATA=TEXT,
// RECFM=F
```

(1)

PATH ファイルのパス名を指定します z/OS UNIX ファイルシステム。

(2)

PATHOPTS は、ファイルのアクセス (読み取り、読み取り / 書き込みなど) を示し、ファイルの状況 (付加、作成、切り捨てなど) を設定します。

(3)

PATHMODE は、ファイルの作成時に設定される許可、すなわちファイル・アクセス属性を示します。

(4)

FILEDATA は、データをテキストまたはバイナリーのどちらとして扱うかを指定します。

複数のファイルを混在させて使用できます z/OS UNIX ファイルシステム (PATH='unix ディレクトリパス') と伝統的な MVS データセット (DSN= mvs データセット名) の編集 DD ステートメント (この例ではオーバーライドとして表示)。ただし、コンパイラー・ユーティリティー・ファイル (DD ステートメント SYSUTx) および COPY ライブラリー (DD ステートメント SYSLIB) は、MVS データ・セットでなければなりません。

関連参照

[291 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』](#)

[UNIX システム・サービス・コマンド解説書](#)

[MVS JCL リファレンス](#)

コンパイルおよびリンク・エディット用プロシージャー (IGYWCL)

IGYWCL プロシージャーは、プログラムのコンパイルとリンク・エディットを行う 2 ステップのカタログ式プロシージャーです。

COBOL ジョブ・ステップは、バインダー (リンケージ・エディター) への入力となるオブジェクト・モジュールを生成します。他のオブジェクト・モジュールを追加することもできます。次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD *          (or appropriate parameters)
```

```
//COBOL.SYSLIB DD DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```

//IGYWCL PROC      LNGPRFX='IGY.V6R5M0',
//                  LIBPRFX='CEE',
//                  PGMLIB='&&GOSET',GOPGM=GO
//**
//**
//** COMPILE AND LINK EDIT A COBOL PROGRAM
//**
//** PARAMETER      DEFAULT VALUE      USAGE
//**  LNGPRFX      IGY.V6R5M0          PREFIX FOR LANGUAGE DATA SET NAMES
//**  SYSLBLK      3200                BLOCK SIZE FOR OBJECT DATA SET
//**  LIBPRFX      CEE                 PREFIX FOR LIBRARY DATA SET NAMES
//**  PGMLIB       &&GOSET              DATA SET NAME FOR LOAD MODULE
//**  GOPGM        GO                  MEMBER NAME FOR LOAD MODULE
//**
//** CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//**
//** CALLER MUST ALSO SUPPLY //COBOL.SYSLIB DD . . . for COPY statements
//**
//COBOL EXEC PGM=IGYCRCTL,REGION=0M
//STEPLIB DD      DSNAME=&LNGPRFX..SIGYCOMP,DISP=SHR      (1)
//          DD      DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//          DD      DSNAME=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSPRINT DD      SYSOUT=*
//SYSLIN DD      DSNAME=&&LOADSET,UNIT=SYSALLDA,
//              DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
//SYSUT1 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT2 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT3 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT4 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT5 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT6 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT7 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT8 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT9 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT10 DD     UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT11 DD     UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT12 DD     UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT13 DD     UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT14 DD     UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT15 DD     UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSMDECK DD     UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//LKED EXEC PGM=IEWBLINK,COND=(8,LT,COBOL),REGION=0M
//SYSLIB DD      DSNAME=&LIBPRFX..SCEELKEX,DISP=SHR      (2)
//          DD      DSNAME=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT DD      SYSOUT=*
//SYSLIN DD      DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//          DD      DDNAME=SYSIN
//SYSLMOD DD      DSNAME=&PGMLIB(&GOPGM),
//              SPACE=(CYL,(3,1,1)),
//              UNIT=SYSALLDA,DISP=(MOD,PASS),DSNTYPE=LIBRARY
//SYSUT1 DD      UNIT=SYSALLDA,SPACE=(CYL,(1,1))

```

(2) SYSLIB は、インストール先によって異なります。

COBOL ジョブ・ステップは、バインダー（リンケージ・エディター）への入力となるオブジェクト・モジュールを生成します。他のオブジェクト・モジュールを追加することもできます。COBOL プログラムでデータ・セットを参照する場合は、これらのデータ・セットを定義する DD ステートメントも指定する必要があります。

ります。次の DD ステートメントを入力ストリームで指定し、ソース・プログラムの位置を示す必要があります。

```
//COBOL.SYSIN DD *          (or appropriate parameters)
```

プログラムがコピーブックを使用する場合は、COPY ステートメントで指定されている SYSLIB または他のライブラリーに対して DD ステートメントも指定する必要があります。例:

```
//COBOL.SYSLIB DD DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWCLG PROC LNGPRFX='IGY.V6R5M0',
//          LIBPRFX='CEE',GOPGM=GO
//*
//*  COMPILE, LINK EDIT AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE  USAGE
//*  LNGPRFX    IGY.V6R5M0      PREFIX FOR LANGUAGE DATA SET NAMES
//*  LIBPRFX    CEE             PREFIX FOR LIBRARY DATA SET NAMES
//*  GOPGM      GO              MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//*  CALLER MUST ALSO SUPPLY //COBOL.SYSLIB DD . . . for COPY statements
//*
//COBOL EXEC PGM=IGYCRCTL,REGION=0M
//STEPLIB DD DSN=DSNAME=&LNGPRFX..SIGYCOMP,DISP=SHR          (1)
//          DD DSN=DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//          DD DSN=DSNAME=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSN=DSNAME=&&LOADSET,UNIT=SYSALLDA,
//          DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
//          UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT8 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT9 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT10 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT11 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT12 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT13 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT14 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT15 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSDUMP DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//LKED EXEC PGM=IEWBLINK,COND=(8,LT,COBOL),REGION=0M
//SYSLIB DD DSN=DSNAME=&LIBPRFX..SCEELKEX,DISP=SHR          (2)
//          DD DSN=DSNAME=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSN=DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//          DD DDNAME=SYSIN
//SYSLMOD DD DSN=DSNAME=&&GOSET(&GOPGM),SPACE=(CYL,(1,1,1)),
//          UNIT=SYSALLDA,DISP=(MOD,PASS),DSNTYPPE=LIBRARY
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//GO EXEC PGM=*.LKED.SYSLMOD,COND=(8,LT,COBOL),(4,LT,LKED)),
//          REGION=0M
//STEPLIB DD DSN=DSNAME=&LIBPRFX..SCEERUN,DISP=SHR          (1)
//          DD DSN=DSNAME=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
```

(1)
STEPLIB は、インストール先によって異なります。

(2)
SYSLIB は、インストール先によって異なります。

プログラム (GO ステートメント) を実行するためのプロシージャでは、有効な *DDName* の最大長は 8 文字です。COBOL プログラムの ENVIRONMENT DIVISION には FILE CONTROL 段落があり、その *assignment-name* は *DDName* と一致していなければなりません。例:

```
//GO.DDName DD DSN=data-set-name
```

プログラムをコンパイルするための JCL の作成

カタログ式プロシージャではより複雑なプログラムに必要な柔軟性が得られない場合は、独自のジョブ制御ステートメントを作成してください。次の例では、プログラムのコンパイルに使用される JCL の一般形式を示します。

このタスクについて

```
//jobname JOB acctno,name,MSGCLASS=1 (1)
//stepname EXEC PGM=IGYCRCTL,PARM=(options) (2)
//STEPLIB DD DSN=IGY.V6R5M0.SIGYCOMP,DISP=SHR (3)
// DD DSN=SYS1.SCEERUN,DISP=SHR
// DD DSN=SYS1.SCEERUN2,DISP=SHR
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(subparms) (4)
//SYSUT2 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT3 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT4 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT5 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT6 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT7 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT8 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT9 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT10 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT11 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT12 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT13 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT14 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSUT15 DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSMDECK DD UNIT=SYSALLDA,SPACE=(subparms)
//SYSPRINT DD SYSOUT=A (5)
//SYSLIN DD DSN=MYPROG,UNIT=SYSALLDA, (6)
// DISP=(MOD,PASS),SPACE=(subparms)
//SYSIN DD DSN=dsname,UNIT=device, (7)
// VOLUME=(subparms),DISP=SHR
```

- (1) JOB ステートメントは、ジョブの開始を示します。
- (2) EXEC ステートメントは、Enterprise COBOL コンパイラ (IGYCRCTL) を呼び出すことを指定します。
- (3) この DD ステートメントは、Enterprise COBOL コンパイラが常駐するデータ・セットを定義します。言語環境データ・セットが LNKLIST で使用可能な場合を除き、言語環境 SCEERUN および SCEERUN2 データ・セットを (コンパイラ SIGYCOMP データ・セットとともに) 連結に組み込む必要があります。
- (4) SYSUT DD ステートメントは、コンパイラがソース・プログラムの処理に使用するユーティリティー・データ・セットを定義します。すべての SYSUT ファイルは、直接アクセス・ストレージ・デバイス上になければなりません。
- (5) SYSPRINT DD ステートメントは、LIST や MAP などのコンパイラ・オプションから出力を受け取るデータ・セットを定義します。SYSOUT=A は、宛先がシステム出力装置であるデータ・セットの標準指定です。
- (6) SYSLIN DD ステートメントは、OBJECT コンパイラ・オプションからの出力を受け取るデータ・セット (オブジェクト・モジュール) を定義します。

(7)

SYSIN DD ステートメントは、ジョブ・ステップへの入力として使用されるデータ・セット (ソース・コード) を定義します。

以下のデータ・セットのコンパイル DD ステートメントでは、z/OS UNIX ファイルシステム (PATH='unix ディレクトリー・パス') と従来の MVS とはデータ・セット (DSN=mvs データ・セット名) のファイルを組み合わせ使用できます。

- ソース・ファイル
- オブジェクト・ファイル
- リスト
- ADATA ファイル
- デバッグ・ファイル
- 実行可能モジュール

ただし、コンパイラ・ユーティリティー・ファイル (DD ステートメント SYSUTX) および COPY ライブラリー (DD ステートメント SYSLIB) は、MVS とはデータ・セットでなければなりません。

JCL 内のカタログ式プロシージャに関する例をさらに確認するには、[280 ページの『カタログ式プロシージャの使用』](#)とその後続のトピックを参照してください。

[286 ページの『例: コンパイル用のユーザー作成 JCL』](#)

[642 ページの『例: プロシージャ型 DLL アプリケーションのサンプル JCL』](#)

関連参照

[MVS JCL リファレンス](#)

[MVS プログラム管理: ユーザーズ・ガイドおよび解説書](#)

例: コンパイル用のユーザー作成 JCL

次の例は、基本 JCL を適応させることができる可能性をいくつか示します。

```
//JOB1      JOB                                (1)
//STEP1     EXEC PGM=IGYCRCTL,PARM='OBJECT'    (2)
//STEPLIB   DD DSN=IGY.V6R5M0.SIGYCOMP,DISP=SHR
//          DD DSN=SYS1.SCEERUN,DISP=SHR
//          DD DSN=SYS1.SCEERUN2,DISP=SHR
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT3    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT8    DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT9    DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT10   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT11   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT12   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT13   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT14   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT15   DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSMDECK  DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSPRINT  DD SYSOUT=A
//SYSLIN    DD DSN=MYPROG,UNIT=SYSDA,
//          DISP=(MOD,PASS),SPACE=(TRK,(3,3))
//SYSIN     DD *                                (3)
000100 IDENTIFICATION DIVISION.
. . .
/*                                              (4)
```

(1)

JOB1 は、ジョブの名前です。

(2)

STEP1 は、ジョブ内で唯一のジョブ・ステップの名前です。この EXEC ステートメントは、生成されたオブジェクト・コードを (リンク・ステップへの入力として使用する) ディスクまたはテープに入れることも指定しています。

(3)

アスタリスクは、入力ストリームの中で入力データ・セットが続くことを示します。

(4)

区切りステートメント /* は、入力ストリームの中で、データとその後の制御ステートメントとを区切ります。

TSO のもとでのコンパイル

TSO のもとでは、TSO コマンド、コマンド・リスト (CLIST)、REXX EXEC、または ISPF を使用すると、従来の MVS データ・セットを使用するプログラムをコンパイルすることができます。z/OS ユニックスファイルを使用するプログラムの場合は、TSO コマンドまたは REXX EXEC を使用してコンパイルすることができます。

このタスクについて

いずれの方式を使用する場合も、以下のように、データ・セットを割り振り、コンパイルを要求しなければなりません。

手順

1. ALLOCATE コマンドを使用してデータ・セットを割り振ります。

どのコンパイルでも、作業データ・セット (SYSUTn) と、SYSIN および SYSPRINT データ・セットを割り振らなければなりません。

特定のコンパイラー・オプションを指定する場合は、他のデータ・セットも割り振る必要があります。例えば、TERMINAL コンパイラー・オプションを指定した場合には、SYSTEM データ・セットを割り振って、端末でコンパイラー・メッセージを受け取る必要があります。

データ・セットはどんな順序で割り振っても構いません。ただし、コンパイルを開始する前に、必要なすべてのデータ・セットを割り振っておく必要があります。

2. READY プロンプトで CALL コマンドを使用してコンパイルを要求します。

```
CALL 'IGY.V6R5M0.SIGYCOMP(IGYCRCTL)'
```

タスクの結果

指定できるのは ALLOCATE そして CALLTSO コマンドラインでコマンドを実行するか、使用していない場合は z/OS ユニックスファイル s の場合は、それらを CLIST に含めることができます。

SYSUTx ユーティリティ・データ・セットおよび SYSLIB ライブラリーを除いて、どのコンパイラー・データ・セットにも z/OS ユニックスファイルを割り振ることができます。ALLOCATE ステートメントの形式は、次のとおりです。

```
Allocate File(SYSIN) Path('/u/myu/myap/std/prog2.cbl')
Pathopts(ORDONLY) Filedata(TEXT)
```

[288 ページの『例: TSO のもとでコンパイルするための ALLOCATE および CALL』](#)

[288 ページの『例: TSO のもとでコンパイルするための CLIST』](#)

関連参照

[291 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』](#)

例: TSO のもとでコンパイルするための ALLOCATE および CALL

次の例は、TSO のもとでコンパイルするときの ALLOCATE および CALL コマンドの指定方法を示します。

```
[READY]
ALLOCATE FILE(SYSUT1) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT2) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT3) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT4) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT5) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT6) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT7) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT8) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT9) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT10) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT11) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT12) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT13) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT14) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSUT15) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSMDECK) CYLINDERS SPACE(1 1)
[READY]
ALLOCATE FILE(SYSPRINT) SYSOUT
[READY]
ALLOCATE FILE(SYSTEM) DATASET(*)
[READY]
ALLOCATE FILE(SYSLIN) DATASET(PROG2.OBJ) NEW TRACKS SPACE(3,3)
[READY]
ALLOCATE FILE(SYSIN) DATASET(PROG2.COBOL) SHR
[READY]
CALL 'IGY.V6R5M0.SIGYCOMP(IGYCRCTL)' 'LIST,NOCOMPILE(S),OBJECT,FLAG(E,E),TERMINAL'
      (COBOL listings and messages)
[READY]
FREE
FILE(SYSUT1,SYSUT2,SYSUT3,SYSUT4,SYSUT5,SYSUT6,SYSUT7,SYSUT8,SYSUT9,SYSUT10,SYSUT11,SYSUT12,
SYSUT13,SYSUT14,SYSUT15,SYSMDECK,SYSPRINT,SYSTEM,+
SYSIN,SYSLIN)
[READY]
```

例: TSO のもとでコンパイルするための CLIST

次の例は、TSO のもとでコンパイルするための CLIST を示しています。FREE コマンドは必要ありません。しかし、上手なプログラミングを行うには、ファイルを割り振る前に、それらを解放しておく必要があります。

```
PROC 1 MEM
CONTROL LIST
FREE F(SYSUT1)
FREE F(SYSUT2)
FREE F(SYSUT3)
FREE F(SYSUT4)
FREE F(SYSUT5)
FREE F(SYSUT6)
FREE F(SYSUT7)
FREE F(SYSUT8)
FREE F(SYSUT9)
FREE F(SYSUT10)
FREE F(SYSUT11)
```



```

FREE F(SYSUT12)
FREE F(SYSUT13)
FREE F(SYSUT14)
FREE F(SYSUT15)
FREE F(SYSMDECK)
FREE F(SYSPRINT)
FREE F(SYSIN)
FREE F(SYSLIN)
ALLOC F(SYSPRINT) SYSOUT
ALLOC F(SYSIN) DA(COBOL.SOURCE(&MEM)) SHR REUSE
ALLOC F(SYSLIN) DA(COBOL.OBJECT(&MEM)) OLD REUSE
ALLOC F(SYSUT1) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT2) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT3) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT4) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT5) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT6) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT7) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT8) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT9) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT10) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT11) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT12) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT13) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT14) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSUT15) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
ALLOC F(SYSMDECK) NEW SPACE(1,1) CYL UNIT(SYSALLDA)
CALL 'IGY.V6R5M0.SIGYCOMP(IGYCRCTL)'

```

関連参照

TSO/E コマンド解説書

アセンブラー・プログラムからコンパイラーを開始する

動的起動で ATTACH または LINK マクロを使用して、アセンブラー・プログラム内から Enterprise COBOL コンパイラーを開始することができます。コンパイラー・オプションおよび処理時に使用するデータ・セットの DD 名を指定する必要があります。

このタスクについて

例:

```
symbol {LINK|ATTACH} EP=IGYCRCTL,PARAM=(optionlist[,ddnamelist]),VL=1
```

EP

コンパイラーのシンボル名を指定します。制御プログラム (ライブラリー・ディレクトリー記入項目からの) が、プログラムが実行を開始すべき入り口点を決定します。

PARAM

アセンブラー・プログラムからコンパイラーに渡されるアドレス・パラメーターをサブリストとして指定します。

アドレス・パラメーター・リストの最初のフルワードには、COBOLoptionlist のアドレスが入っています。2 番目のフルワードには、ddnamelist のアドレスが入っています。

optionlist

コンパイル用に指定された COBOL オプションが入っている可変長リストのアドレスを指定します。リストを提示しない場合でも、このアドレスは指定しなければなりません。

optionlist はハーフワード境界から始めなければなりません。高位の 2 バイトには、このリストの残りの部分にあるバイト数のカウントが含まれます。オプションが指定されない場合、このカウントは 0 でなければなりません。optionlist は、各フィールドが次のフィールドとコンマで区切られて、自由形式です。ブランクまたはゼロは使用できません。コンパイラーは最初の 100 文字のみを認識します。

ddnamelist

コンパイラー処理中に使用されるデータ・セットの代替の DD 名を含んでいる可変長リストのアドレスを指定します。標準 DD 名を使用する場合は、ddnamelist を省略することができます。

ddnamelist はハーフワード境界から始めなければなりません。高位の 2 バイトには、このリストの残りの部分にあるバイト数のカウントが含まれます。8 バイト未満の名前は、左寄せにしてブランクで埋め込む必要があります。代替 DD 名がリストから省略されている場合、標準名が想定されます。名前が省略されている場合、8 バイトの記入項目には 2 進数 0 が入っていなければなりません。リストを短縮して、終わりから名前を省略することができます。

指定された SYSUTn データ・セットはすべて直接アクセス記憶装置上になければならず、物理順次編成をもっていなければなりません。これらのデータ・セットが z/OS UNIX ファイルシステムに常駐してはなりません。

次の表に、*ddnamelist* 内の 8 バイトの記入項目のシーケンスを示します。

代替 DD 名 8 バイト項目	代替 DD 名が置き換えられる名前
1	SYSLIN
2	該当なし
3	該当なし
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSUT4
12	SYSTEM
※13	SYSUT5
14	SYSUT6
15	SYSUT7
16	SYSADATA
17	SYSJAVA
18	SYSDEBUG
19	SYSMDECK
20	DBRMLIB
21	SYSOPTF
22	SYSUT8
23	SYSUT9
24	SYSUT10
25 GB	SYSUT11
26	SYSUT12
27	SYSUT13
28	SYSUT14
29	SYSUT15

VL

アドレス・パラメーター・リストの最後のフルワードの符号ビットを 1 に設定するように指定します。
コンパイラーは、処理を完了すると、レジスター 15 に戻りコードを書き込みます。

関連タスク

[291 ページの『コンパイラー入出力の定義』](#)

関連参照

[291 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』](#)

[301 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』](#)

コンパイラー入出力の定義

コンパイラーが作業を行うために使用する数種類のデータ・セットを定義しなければなりません。コンパイラーは、入力データ・セットおよびライブラリーを使用して、さまざまなタイプの出力 (オブジェクト・コード、リスト、およびメッセージを含む) を作成します。また、コンパイラーは、コンパイル時にユーティリティー・データ・セットも使用します。

このタスクについて

関連タスク

[294 ページの『ソース・コード・データ・セットの定義 \(SYSIN\)』](#)

[295 ページの『コンパイラー・オプション・データ・セットの定義 \(SYSOPTF\)』](#)

[296 ページの『ソース・ライブラリーの指定 \(SYSLIB\)』](#)

[296 ページの『出力データ・セットの定義 \(SYSPRINT\)』](#)

[296 ページの『コンパイラー・メッセージの端末への送信 \(SYSTEM\)』](#)

[297 ページの『オブジェクト・コードの作成 \(SYSLIN または SYSPUNCH\)』](#)

[297 ページの『関連データ・ファイル \(SYSADATA\) の定義』](#)

[297 ページの『Java ソース出力ファイル \(SYSJAVA\) の定義』](#)

[298 ページの『ライブラリー処理出力ファイル \(SYSMDECK\) の定義』](#)

関連参照

[291 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』](#)

[301 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』](#)

z/OS のもとでコンパイラーによって使用されるデータ・セット

次の表は、コンパイラーが使用する各データ・セットの機能、装置要件、および許可される装置クラスをリストしたものです。

表 36. コンパイラー・データ・セット						
タイプ	DD 名 (ddname)	関数	必須かどうか	装置要件	許可される 装置クラス	z/OS UNIX ファ イルシス テムで使 用できる か?
入力	SYSIN ¹	ソース・プログラムの読み取り	はい	カード読取装置; 中間記憶装置	任意の	はい
	SYSOPTF	コンパイラー・オプションの読み取り	OPTFILE が有効な場合	カード・リーダー、中間ストレージ、直接アクセス	Any	はい
	SYSLIB または他のコピー・ライブラリー ¹	ユーザー・ソース・ライブラリーの読み取り (PDSs または PDSEs)	プログラムが COPY または BASIS 声明	直接アクセス	SYSDA	いいえ
ユーティリティー ²	SYSUT1、SYSUT2、SYSUT3、SYSUT4、SYSUT6	コンパイル時にコンパイラーによって使用される作業データ・セット	はい	直接アクセス	SYSALLDA	いいえ
	SYSUT5	コンパイル時にコンパイラーによって使用される作業データ・セット	プログラムに COPY、REPLACE、または BASIS ステートメントが含まれている場合	直接アクセス	SYSALLDA	いいえ
	SYSUT7	リストを作成するためにコンパイラーによって使用される作業データ・セット	はい	直接アクセス	SYSALLDA	いいえ
	SYSUT8、SYSUT9、SYSUT10、SYSUT11、SYSUT12、SYSUT13、SYSUT14、SYSUT15	コンパイル時にコンパイラーによって使用される作業データ・セット	はい	直接アクセス	SYSALLDA	いいえ

表 36. コンパイラー・データ・セット (続き)

タイプ	DD 名 (ddname)	関数	必須かどうか	装置要件	許可される 装置クラス	z/OS UNIX ファ イルシス テムで使 用できる か?
出力	SYSPRINT ¹	ストレージ・マップ、 リスト、およびメッ セージの書き込み	はい	プリンター; 中間記憶装 置	SYSSQ、 SYSDA、標準 出力クラス A	はい
	SYSTEM	進行メッセージおよ び診断メッセージの 書き込み	TERM が有効な場合	出力装置; TSO 端末		はい
	SYSPUNCH	オブジェクト・コー ドの作成	DECK が有効な場合	カード穿孔 装置; 直接ア クセス	SYSSQ、 SYSDA	はい
	SYSLIN	コンパイラーからの 出力およびバインダ ー (リンケージ・エデ ィター) への入力と してのオブジェク ト・モジュール・デ ータ・セットの作成	OBJECT が有効な場合	直接アクセ ス	SYSSQ、 SYSDA	はい
	SYSADATA ¹	関連データ・ファイ ル・レコードの書き 込み	ADATA が有効な場合	出力装置		はい
	SYSJAVA	クラス定義用の生成 済み Java ソース・フ ァイルの作成	クラス定義をコンパイ ルする場合	(z/OS ユニッ クスファイ ルでなければ ならない)		はい
	SYSUDUMP、 SYSABEND、 または SYSMDUMP	ダンプの書き込み	DUMP が有効な場合 (ま れにしか使用されない)	直接アクセ ス	SYSDA	はい
	SYSDEBUG	オブジェクト・モジ ュールとは別のデー タ・セットへの記号 デバッグ情報テーブ ルの書き込み	TEST(...,SEP,...) が有効な場合	直接アクセ ス	SYSDA	はい
	SYSMDECK	処理 MDECK オプシ ョン、または作業デ ータセット NOMDECK 指定され ている	はい	直接アクセ ス	SYSALLDA	はい

1. EXIT オプションを使用して、これらのデータ・セットからユーザー出口を提供することができます。
2. ユーティリティ・データ・セットは単一ボリュームでなければなりません。ユーティリティ・データ・セットには DSNTYPE=LARGE (SYSUT1 - SYSUT15) を指定できません。

関連参照

[294 ページの『論理レコード長とブロック・サイズ』](#)

[399 ページの『出口』](#)

論理レコード長とブロック・サイズ

作業データ・セット (SYSUT n) および z/OS ユニックスファイル 以外のコンパイラ・データ・セットの場合は、DCB パラメータの BLKSIZE サブパラメータを使用してブロック・サイズを設定することができます。この値は、データ・セットが常駐する装置に許されているものでなければなりません。設定する値は、データ・セットが固定長か可変長かによって異なります。

固定長レコード (RECFM=F または RECFM=FB) の場合、LRECL は論理レコード長で、BLKSIZE は LRECL $\times$ n に等しくなります (n はブロック化因数)。

次の表は、固定長データ・セットに定義されている値を示しています。一般には、これらの値を変更すべきではありませんが、以下のデータ・セットの場合は値を変更できます。

- SYSDEBUG: リストされた範囲内で、任意の LRECL を指定することができます。推奨値は 1024 です。
- SYSPRINT、SYSDEBUG: BLKSIZE=0 を指定でき、その結果として、システム決定のブロック・サイズになります。

表 37. 固定長コンパイラ・データ・セットのブロック・サイズ			
データ・セット	RECFM	LRECL (バイト)	BLKSIZE ¹
SYSDEBUG ²	F または FB	80 から 1024 ³	LRECL $\times$ n
SYSIN	F または FB	80	80 $\times$ n
SYSLIB または他のコピー・ライブラリー	F または FB	80	80 $\times$ n
SYSLIN	F または FB	80	80 $\times$ n
SYSMDECK	F または FB	80	80 $\times$ n
SYSOPTF	F または FB	80	80 $\times$ n
SYSPRINT ²	F または FB	133	133 $\times$ n
SYSPUNCH	F または FB	80	80 $\times$ n
SYSTEM	F または FB	80	80 $\times$ n
1. n = ブロック化因数 2. BLKSIZE=0 を指定すると、システムがブロック・サイズを決定します。 3. SYSDEBUG のデフォルトの LRECL は 1024 です。			

可変長レコード (RECFM=V) の場合、LRECL は論理レコード長で、BLKSIZE は LRECL + 4 に等しくなります。

表 38. 可変長コンパイラ・データ・セットのブロック・サイズ			
データ・セット	RECFM	LRECL (バイト)	BLKSIZE (バイト) 最小許容値
SYSADATA	VB	1020	1024

ソース・コード・データ・セットの定義 (SYSIN)

以下に示す SYSIN DD ステートメントを使用して、ソース・コードが入ったデータ・セットを定義します。

このタスクについて

```
//SYSIN DD DSNAME=dsname,UNIT=SYSSQ,VOLUME=(subparms),DISP=SHR
```

ソース・コードまたは BASIS ステートメントを入力ストリームに直接入れることができます。そうするには、次の SYSIN DD ステートメントを使用してください。

```
//SYSIN DD *
```

ソースコードまたは BASIS 声明文は DD * 声明。コンパイルの後に別のジョブ・ステップが続いている場合は、そのステップの EXEC ステートメントが /* ステートメントまたは最後のソース・ステートメントの後に続く必要があります。

コンパイラ・オプション・データ・セットの定義 (SYSOPTF)

以下に示す SYSOPTF DD ステートメントをコーディングすることによって、COBOL プログラムのコンパイラ・オプションが入ったデータ・セットを定義します。

このタスクについて

```
//SYSOPTF DD DSNAME=dsname,UNIT=SYSDA,VOLUME=(subparms),DISP=SHR
```

コンパイラ・オプション・データ・セットを使用するには、OPTFILE を、コンパイラ呼び出しオプションとして、またはソース・プログラムの PROCESS または CBL ステートメントで、指定します。

SYSOPTF データ・セット内で以下を行います。

- 呼び出しオプション、または PROCESS または CBL ステートメントのコンパイラ・オプションで使用するものと同じ構文を使用して、桁 2 から桁 72 に、フリー・フォームで、コンパイラ・オプションを指定します。
- 行がコメントとして扱われるようにするには、桁 1 にアスタリスク (*) をコーディングします。
- オプションとして、桁 73 から 80 に、シーケンス番号をコーディングします。これらの桁は無視されます。

また、OPTFILE オプションを使用してコンパイルする場合は、SYSOPTF DD ステートメントの後ろの入力ストリームに直接、コンパイラ・オプションを配置することができます。

```
//COB      EXEC PGM=IGYCRCTL,PARM='OPTFILE'
//SYSOPTF DD DATA,DLM=@@
SSRANGE   ARITH(COMPAT)
OPTIMIZE
. . .
@@
//SYSIN    DD . . .
```

複数のコンパイラ・オプション・データ・セットがある場合には、以下のように、複数の SYSOPTF DD ステートメントを連結することができます。

```
//SYSOPTF DD DSNAME=dsname1, . . .
//        DD DSNAME=dsname2, . . .
```

連結の後ろのデータ・セットにあるコンパイラ・オプションが、連結のそれより前にあるデータ・セットのオプションよりも優先されます。

関連参照

[294 ページの『論理レコード長とブロック・サイズ』](#)

[428 ページの『OPTFILE』](#)

ソース・ライブラリーの指定 (SYSLIB)

プログラムに COPY または BASIS ステートメントが含まれている場合には、SYSLIB DD ステートメントを使用します。これらの DD ステートメントは、ソース・コード内の COPY ステートメントまたは入力ストリーム内の BASIS ステートメントによって要求されるデータが入っているライブラリー (区分データ・セット) を定義します。

このタスクについて

```
//SYSLIB DD DSNAME=copylibname,DISP=SHR
```

複数のコピー・ライブラリーまたは基本ライブラリーがある場合には、次のように複数の DD ステートメントを連結します。

```
//SYSLIB DD DSNAME=PROJECT.USERLIB,DISP=SHR  
// DD DSNAME=SYSTEM.COPYX,DISP=SHR
```

ライブラリーは直接アクセス記憶装置に置かれます。JCL を使用して、または TSO のもとで、コンパイルする場合、ライブラリーが z/OS UNIX ファイルシステムに入っていないはなりません。

出力データ・セットの定義 (SYSPRINT)

DD 名 SYSPRINT を使用して、リストを生成します。このリストには、PARM パラメーターのデフォルト・オプションまたは要求されたオプションの結果 (すなわち、診断メッセージ、オブジェクト・コード・リスト) が入れられます。

このタスクについて

出力は、SYSOUT データ・セット、プリンター、直接アクセス・ストレージ・デバイス、または磁気テープ装置に送ることができます。例:

```
//SYSPRINT DD SYSOUT=A
```

の SYSPRINT データセットは、シーケンシャルデータセット、PDS または PDSE メンバー、または z/OS ユニックスファイル。SYSPRINT データ・セットのレコード・フォーマット、レコード長、およびブロック・サイズの指定方法の詳細については、下の関連参照を参照してください。

関連参照

[294 ページの『論理レコード長とブロック・サイズ』](#)

コンパイラー・メッセージの端末への送信 (SYSTEM)

TSO でコンパイルしている場合には、SYSTEM データ・セットを定義して、コンパイラー・メッセージを端末に送信することができます。

このタスクについて

```
ALLOC F(SYSTEM) DA(*)
```

SYSTEM は、他のさまざまな方法で定義することができます (SYSOUT データ・セット、ディスク上のデータ・セット、z/OS UNIX ファイルシステム内のファイル、別の印刷クラスなどに)。

オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)

使用する場合 OBJECT コンパイラオプションを使用すると、オブジェクトコードを従来の MVs データセットまたは z/OS ユニックスファイル、またはテープに録音します。コンパイラは、SYSLIN または SYSPUNCH DD 声明。

このタスクについて

```
//SYSLIN DD DSNAME=dsname,UNIT=SYSDA,  
// SPACE=(subparms),DISP=(MOD,PASS)
```

使用 DISP パラメータの SYSLIN DD オブジェクト コード データ セットが次のいずれであるかを示すステートメント:

- バインダー (リンケージ・エディター) に渡される
- カタログされる
- 保存される
- カタログされた既存のライブラリーに追加される

上記の例では、データが作成され、別のジョブ・ステップであるバインダー (リンケージ・エディター) ジョブ・ステップに渡されます。

インストール時に、DECK オプションと SYSPUNCH DD 声明。B は、穿孔データ・セットの標準出力クラスです。

```
//SYSPUNCH DD SYSOUT=B
```

NOOBJECT オプションが有効な場合、SYSLIN DD ステートメントは必要ではありません。NODECK オプションが有効な場合は、SYSPUNCH DD ステートメントは必要ではありません。

関連参照

[427 ページの『OBJECT』](#)

[393 ページの『DECK』](#)

関連データ・ファイル (SYSADATA) の定義

ADATA コンパイラ・オプションを使用する場合には、SYSADATA ファイルを定義します。

このタスクについて

```
//SYSADATA DD DSNAME=dsname,UNIT=SYSDA
```

SYSADATA ファイルは、コンパイル中に収集されたプログラムに関する情報が入っている特定のレコード・タイプを含む順次ファイルです。このファイルは、従来の MVS データ・セットまたは z/OS ユニックスファイルにすることができます。

関連参照

[378 ページの『ADATA』](#)

Java ソース出力ファイル (SYSJAVA) の定義

オブジェクト指向プログラムをコンパイルする場合は、SYSJAVA DD ステートメントを追加します。生成された Java ソース・ファイルは、SYSJAVA DD 名に書き込まれます。

このタスクについて

```
//SYSJAVA DD PATH='/u/userid/java/Classname.java',  
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),  
// PATHMODE=SIRWXU,  
// FILEDATA=TEXT
```

の SYSJAVA ファイルは z/OS UNIX ファイルシステム。

関連タスク

[326 ページの『JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル』](#)

デバッグ・データ・セットの定義 (SYSDEBUG)

JCL または TSO からコンパイルし、TEST(...,SEP,...) コンパイラオプションを使用すると、シンボリックデバッグ情報テーブルは、SYSDEBUG DD 声明。

このタスクについて

```
//SYSDEBUG DD DSN=dsname,UNIT=SYSDA
```

SYSDEBUG データ・セットは、順次データ・セット、PDS または PDSE メンバー、あるいは HFS ファイルにすることができます。SYSDEBUG データ・セットのレコード・フォーマット、レコード長、およびブロック・サイズの指定方法の詳細については、論理レコード長およびブロック・サイズに関する下記の関連参照を参照してください。

言語環境用途 SYSDEBUG ダンプサービスのため。TEST|NOTEST(...,SEPARATE(DSNNAME),...) コンパイラ・オプションが有効な場合、SYSDEBUG データ・セット名はオブジェクト・プログラムに格納され、実行時にデフォルトとして使用されます。実行時にデータセットの名前を変更するには、SYSDEBUG COBOL デバッグファイルユーザー出口、IGZIUXB。あなたは監督することができます IBM z/OS デバッグ名前を変更したデータセットに SET DEFAULT LISTINGS コマンド、ユーザー終了 EQAUEDAT、または EQADEBUG DD 声明。

指定するデータセット名は DDNAME SYSDEBUG 複数の IBM 製品、含む言語環境、IBM z/OS デバッグ、Fault Analyzer、そしてアプリケーション・パフォーマンス・アナライザー。詳細については、これらの個別製品の資料を参照してください。

関連タスク

言語環境のカスタマイズ (COBOL デバッグ・ファイル名の変更)

IBM z/OS デバッグ・ユーザー・ガイド (デバッグ・ツールが COBOL および PL/I 別個のデバッグ・ファイルを見つける方法)

関連参照

[294 ページの『論理レコード長とブロック・サイズ』](#)

[448 ページの『TEST』](#)

ライブラリー処理出力ファイル (SYSMDECK) の定義

すべてのコンパイルに SYSMDECK DD 割り振が必要です。指定する場合は MDECK コンパイラオプション、SYSMDECK DD 割り当ては永続的なものを指定する必要があります MVs データセットまたは z/OS ユニックスファイル。ただし、NOMDECK オプションを使用する場合は、代わりに SYSMDECK をユーティリティー (一時) データ・セットとして指定できます。

このタスクについて

次の例は、永続データ・セット名を指定します。

```
//SYSMDECK DD DSNAME=dsname,UNIT=SYSDA
```

次の例は、z/OS UNIX ファイルを指定します。

```
//SYSMDECK DD PATH='/path/filename.dek',  
//FILEDATA=TEXT,  
//PATHOPTS=(ORDWR,OCREAT,OTRUNC),  
//PATHMODE=(SIRUSR,SIWUSR)
```

次の例は、ユーティリティ (一時) データ・セットを指定します。

```
//SYSMDECK DD UNIT=SYSALLDA
```

の SYSMDECK ファイルには更新された入力ソースのコピーライブラリ処理、つまり結果の COPY、BASIS、REPLACE、EXEC SQL INCLUDE、そして EXEC SQLIMS INCLUDE 声明。

関連参照

[419 ページの『MDECK』](#)

z/OS のもとでのコンパイラ・オプションの指定

コンパイラは、デフォルトのコンパイラ・オプションを使用してインストールされます。コンパイラのインストール時に、システム・プログラマーは、例えば、パフォーマンスの向上やある程度の標準の維持を確実に行うために、コンパイラ・オプションの設定を固定することができます。固定されたコンパイラ・オプションはオーバーライドできません。

このタスクについて

固定されていないオプションについては、以下のいずれかの方法でコンパイラ・オプションを指定して、デフォルト設定をオーバーライドすることができます。

- COBOL ソースの PROCESS または CBL ステートメントにコーディングする。
- コンパイラの開始時に組み込む (JCL の EXEC ステートメントの PARM パラメーター、または TSO のコマンド行のいずれか)。
- SYSOPTF データ・セットに組み込んで、上記のいずれかの方法で、OPTFILE コンパイラ・オプションを指定する。

コンパイラは、以下の優先順位で高位から下位へ、オプションを認識します。

1. 設置場所で固定されているインストール先デフォルト
2. バッチ内の最初のプログラムに対して有効になっているコンパイラ・オプション BUFSIZE、OUTDD、SQL、および SQLIMS の値
3. IDENTIFICATION DIVISION の前に置かれた PROCESS (または CBL) ステートメント
4. コンパイラ呼び出しで指定されたオプション (JCL PARM パラメーターまたは TSO CALL コマンド)
5. 固定されていないインストール先デフォルト

この優先順位の順序は、対立するオプションまたは互いに排他的なオプションが指定されたときにどのオプションが有効かも決定します。

SYSOPTF データ・セットのオプションの優先順位は、OPTFILE コンパイラ・オプションを指定した場所に依存します。例えば、PROCESS ステートメントで OPTFILE を指定した場合、SYSOPTF オプションによってコンパイラの呼び出し時に指定したオプションが置き換えられます。詳細については、OPTFILE オプションに関する下の関連参照を参照してください。

大部分のオプションは対になっているため、いずれか一方を選択します。例えば、相互参照リストのオプションペアは XREF|NOXREF 相互参照リストが必要な場合は、XREF ; そうでない場合は指定してください NOXREF。

サブパラメーターがあるオプションもあります。例えば、リストで1 ページ当たり 44 行が必要な場合には、LINECOUNT(44) と指定します。

[301 ページの『例: JCL によるコンパイラー・オプションの指定』](#)

[301 ページの『例: TSO のもとでのコンパイラー・オプションの指定』](#)

関連タスク

[295 ページの『コンパイラー・オプション・データ・セットの定義 \(SYSOPTF\)』](#)

[300 ページの『PROCESS \(CBL\) ステートメントでのコンパイラー・オプションの指定』](#)

[304 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』](#)

関連参照

[301 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』](#)

[371 ページの『COBOL コンパイラー・オプション』](#)

[376 ページの『矛盾するコンパイラー・オプション』](#)

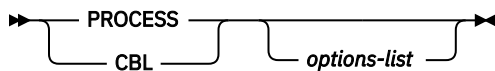
[428 ページの『OPTFILE』](#)

PROCESS (CBL) ステートメントでのコンパイラー・オプションの指定

COBOL プログラム内では、ほとんどのコンパイラー・オプションを PROCESS (CBL) ステートメント内にコーディングできます。このステートメントは、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。

このタスクについて

PROCESS(CBL) ステートメントの構文



シーケンス・フィールドを使用しない場合、PROCESS ステートメントは 1 桁目以降から開始できます。シーケンス・フィールドを使用する場合、シーケンス番号は 1 桁目で始まり、6 文字が含まれる必要があります。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 桁目以降で開始できます。

CBL を PROCESS の同義語として使用することができます。CBL は、シーケンス・フィールドを使用しない場合、同様に 1 桁目以降で開始できます。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 桁目以降で開始できます。

PROCESS および CBL ステートメントは、72 桁目以前で終了させる必要があります。

PROCESS または CBL ステートメントと *options-list* の最初のオプションとは、1 つ以上のブランク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ブランクで区切ります。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 または CBL ステートメントをコーディング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、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 または CBL 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 (CBL) 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 または CBL ステートメントが COBOL プログラムで見つかった場合、COBOL コンパイラーによりエラー診断が生成されます。

関連参照

参照形式 (*Enterprise COBOL for z/OS 言語解説書*)

CBL (PROCESS) ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

例: JCL によるコンパイラー・オプションの指定

次の例は、z/OS のもとで JCL を使用してコンパイラー・オプションを指定する方法を示します。

```
//STEP1      EXEC PGM=IGYCRCTL,  
//           PARM='LIST,NOCOMPILE(S),OBJECT,FLAG(E,E)'
```

例: TSO のもとでのコンパイラー・オプションの指定

次の例は、TSO のもとでコンパイラー・オプションを指定する方法を示します。

```
[READY]  
CALL 'SYS1.LINKLIB(IGYCRCTL)' 'LIST,NOCOMPILE(S),OBJECT,FLAG(E,E)'
```

z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力

コンパイラーがソース・プログラムの処理を完了すると、有効であったコンパイラー・オプションに応じて、1 つ以上の出力が作成されています。

表 39. コンパイラ出力の種類 z/OS		
コンパイラー・オプション	コンパイラー出力	出力のタイプ
ADATA	コンパイル中のプログラムに関する情報	関連データ・ファイル
DLL	DLL サポートで使用可能なオブジェクト・モジュール	オブジェクト
DUMP	コンパイルが異常終了した場合は、システム・ダンプ (SYSUDUMP、SYSABEND、または SYSMDUMP DD ステートメントが必要)。まれにしか使用されません。	リスト
EXPORTALL	DLL のエクスポートされた記号	オブジェクト
FLAG	コンパイラーがプログラムで検出したエラーのリスト	リスト
LIST	マシン言語およびアセンブラ言語でのオブジェクト・コードのリスト	リスト
MAP (HEX) または MAP (DEC)	プログラムのデータ項目のマップ	リスト
MDECK	プログラムのライブラリー処理ステートメントの拡張	ライブラリー処理サイド・ファイル
NUMBER	ユーザー提供の行番号がリストに示される	リスト
COMPILE を指定した OBJECT または DECK	オブジェクト・コード	オブジェクト
OFFSET	オブジェクト・コードの相対アドレスのマップ	リスト
OPTIMIZE (1) または OPTIMIZE (2)	最適化されたオブジェクトコード	オブジェクト
RENT	再入可能オブジェクトコード	オブジェクト
SOURCE	ソース・プログラムのリスト	リスト
SQL	Db2 バインド処理用の SQL ステートメントおよびホスト変数情報	データベース要求モジュール (DBRM)
TERMINAL	端末に送信される進行メッセージと診断メッセージ	端末

表 39. コンパイラ出力の種類 z/OS (続き)		
コンパイラー・オプション	コンパイラー出力	出力のタイプ
TEST (DWARF)	対話式デバッグを可能にするための、オブジェクト・モジュール内の DWARF 形式のデバッグ情報	オブジェクト
TEST (NOSEP)	IBM z/OS デバッガおよび定様式ダンプのための情報テーブル	オブジェクト
TEST (SEP)	IBM z/OS デバッガおよび定様式ダンプのための情報テーブル	独立したデバッグ・ファイル
NOTEST (DWARF)	アプリケーション障害分析ツールを使用可能にするための、基本 DWARF 形式の診断情報	オブジェクト
VBREF	ソース・プログラム内のステートメントの相互参照リスト	リスト
XREF	プロシーチャー、プログラム、およびデータの名前に関するソート済み相互参照リスト	リスト

コンパイルからのリスト出力は、SYSPRINT で定義されたデータ・セットに入れられ、オブジェクト出力は SYSLIN または SYSPUNCH に入れられます。進行メッセージと診断メッセージは、SYSTEMR データ・セットに送ることや、SYSPRINT データ・セットに入れることができます。データベース要求モジュール (DBRM) は、DBRMLIB 内に定義されるデータ・セットです。別個のデバッグファイルは、SYSDEBUG。

コンパイル時に作成されたリストを保管してください。デバッグまたは調整が必要になったときに、作業のテスト段階でこのリストを使用することができます。このリストは、アプリケーションが実動に入ったのちに、診断およびデバッグのために使用することもできます。

コンパイル後に行うのは、コンパイラーがプログラムで見つけたエラーを修正することです。エラーが検出されなかった場合は、プロセスの次のステップである、プログラムのバインド (リンク・エディット) に進むことができます。(コンパイラー・オプションを使用してオブジェクト・コードの生成を抑止した場合、オブジェクト・コードを入手するには再コンパイルする必要があります。)

関連タスク

言語環境プログラミングガイド (リンク・エディットおよび実行の準備)

関連参照

[307 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』](#)

[371 ページの『COBOL コンパイラー・オプション』](#)

複数プログラムのコンパイル (バッチ・コンパイル)

コンパイラーを 1 回呼び出すだけで、一連の別個の COBOL プログラムをコンパイルすることができます。このコンパイルから作成されたオブジェクト・プログラムをリンクして、1 つのプログラム・オブジェクトまたは複数の別個のプログラム・オブジェクトにすることができます (これは NAME コンパイラー・オプションで制御されます)。

このタスクについて

バッチ・ジョブの一部として複数のプログラムをコンパイルするときは、以下を行う必要があります。

- 1 つのプログラム・オブジェクトを作成するのか、複数のプログラム・オブジェクトを作成するのかを決定する。
- シーケンスの中の各プログラムを終了させる。
- コンパイラー・オプションを指定する (バッチ・ジョブ内のプログラムで指定されたコンパイラー・オプションの影響を認識する)。

別個のプログラム・オブジェクトを作成するには、それぞれのオブジェクト・セットの前に **NAME** コンパイラー・オプションを付けなければなりません。コンパイラーが **NAME** コンパイラー・オプションを検出すると、次の **NAME** コンパイラー・オプションが見つかるまで、そのシーケンス内の最初のプログラムおよび後続のすべてのプログラムが1つのプログラム・オブジェクトにリンク・エディットされます。それから、**NAME** オプションを使用してコンパイルされる一連のプログラムは、それぞれ1つの分離したプログラム・オブジェクトに入れられます。

シーケンス内の各プログラムは、**END PROGRAM** マーカーで終了させる必要があります(ただし、バッチの最後のプログラムは例外であり、**END PROGRAM** マーカーは省略可能です)。あるいは、シーケンスの中の各プログラムの前に **CBL** または **PROCESS** ステートメントを付けることができます。

プログラム(プログラム・シーケンスの最後のプログラムを除く)で **END PROGRAM** マーカーを省略すると、そのシーケンス内の次のプログラムが直前のプログラムにネストされます。以下のいずれかの状況では、エラーとなる可能性があります。

- この結果ネストされるプログラムの中に **PROCESS** ステートメントがある。
- **CBL** ステートメントがシーケンス番号域(桁1から6)に完全にコーディングされていない。

CBL ステートメントがシーケンス番号域(桁1から6)に完全にコーディングされている場合は、それはソース・ステートメント行のラベルと見なされるため、その **CBL** ステートメントに関するエラー・メッセージは出されません。

303 ページの『例: バッチ・コンパイル』

関連タスク

304 ページの『バッチ・コンパイルでのコンパイラー・オプションの指定』

関連参照

421 ページの『名前』

例: バッチ・コンパイル

次の例は、IGYWCL カタログ式プロシーチャーを1回呼び出して、3つのプログラム(**PROG1**、**PROG2**、および **PROG3**)のバッチ・コンパイルおよび2つのプログラム・オブジェクトの作成を行う方法を示しています。

次のステップが発生します。

- **PROG1** と **PROG2** は一緒にリンク・エディットされて、**PROG2** という名前の1つのプログラム・オブジェクトが形成されます。このプログラム・オブジェクトの入り口点は、デフォルトによって、プログラム・オブジェクトの最初のプログラムである **PROG1** が使用されます。
- **PROG3** それ自体がリンク編集されてプログラムオブジェクトその名前を持つ **PROG3**。この場合は、プログラム・オブジェクト内の唯一のプログラムであるので、入り口点も **PROG3** です。

```
//jobname JOB acctno,name,MSGLEVEL=1
//stepname EXEC IGYWCL
//COBOL.SYSIN DD *
010100 IDENTIFICATION DIVISION.
010200 PROGRAM-ID PROG1.

      .
019000 END PROGRAM PROG1.
020100 IDENTIFICATION DIVISION.
020200 PROGRAM-ID PROG2.

      .
029000 END PROGRAM PROG2.
      CBL NAME
030100 IDENTIFICATION DIVISION.
030200 PROGRAM-ID PROG3.

      .
039000 END PROGRAM PROG3.
/*
//LKED.SYSLMOD DD DSN=&&GOSSET          (1)
/*
//P2          EXEC PGM=PROG2
//STEPLIB     DD   DSN=&&GOSSET,DISP=(SHR,PASS) (2)
```

```

      . . .
      /*
      //P3      EXEC PGM=PROG3
      //STEPLIB DD      DSN=&&G0SET,DISP=(SHR,PASS)
      . . .
      /*
      //

```

(1)

データセット名は LKED ステップ SYSLMOD 仮名称に変更&&G0SET メンバー名なし。

(2)

一時データセット&&G0SET として使用されます STEPLIB ステップ用 P2 そして P3 コンパイルされたプログラムを実行します。言語環境のライブラリーが共用ストレージに常駐していない場合は、さらに、ライブラリー・データ・セットを STEPLIB の DD ステートメントとして追加する必要があります。

(3)

PROG1 および PROG2 の実行に必要なその他の DD ステートメントと入力を追加しなければなりません。

(4)

PROG3 の実行に必要なその他の DD ステートメントと入力を追加しなければなりません。

関連参照

言語環境プログラミングガイド (IBM 提供のカタログ式プロシージャー)

バッチ・コンパイルでのコンパイラー・オプションの指定

バッチ・シーケンス内の各プログラムにコンパイラー・オプションを指定するには、プログラムの前に CBL または PROCESS ステートメントを指定するか、コンパイラーの呼び出し時に指定します。

このタスクについて

現行プログラムで CBL または PROCESS ステートメントが指定されている場合、コンパイラーは、CBL または PROCESS ステートメントを、最初のプログラムの前に有効なオプションとともに解決します。現行プログラムに CBL または PROCESS ステートメントが含まれていない場合、コンパイラーは、前のプログラムに有効なオプションの設定値を使用します。

特定のコンパイラー・オプションは、バッチ・シーケンスの中の各プログラムのコンパイラー・オプション設定の優先順位に影響を及ぼすことに注意してください。コンパイラー・オプションは、以下の優先順位 (高位から下位) で、認識されます。

1. 設置場所で固定されているインストール先デフォルト。
2. バッチ内の最初のプログラムに有効な BUFSIZE、DEFINE、OUTDD、SQL、SQLIMS、および TEST コンパイラー・オプションの値
3. 現行プログラムの CBL または PROCESS ステートメントのオプション (ある場合)
4. コンパイラー呼び出しで指定されたオプション (JCL PARM または TSO CALL)
5. 固定されていないインストール先デフォルト

バッチ・シーケンス内のいずれかのプログラムが、BUFSIZE、DEFINE、OUTDD、SQL、SQLIMS、または TEST オプションを必要とする場合、そのオプションは、バッチ・シーケンス内の最初のプログラムに対して有効でなければなりません。(BASIS、COPY、または REPLACE ステートメントを処理するとき、コンパイラーは、バッチ処理のすべてのプログラムを 1 つの入力ファイルとして処理します。)

バッチに対して オプションを指定した場合、バッチ・コンパイル中に NUMBER および SEQUENCE オプションを変更することはできません。コンパイラーは、オプションのもとで NUMBER および SEQUENCE を処理するとき、バッチ内のすべてのプログラムを 1 つの入力ファイルとして扱います。このため、入力ファイル全体のシーケンス番号が昇順になっていなければなりません。

コンパイラーが CBL または PROCESS ステートメントの LANGUAGE オプションをエラーと診断すると、言語選択は、コンパイラーが最初の CBL または PROCESS ステートメントを検出する前に有効であった内容

に戻ります。バッチ・コンパイル時に有効な言語は、その環境での CBL または PROCESS ステートメントの処理規則に準拠しています。

[305 ページの『例: バッチ・コンパイルでのオプションの優先順位』](#)

[305 ページの『例: バッチ・コンパイルでの LANGUAGE オプション』](#)

関連参照

[393 ページの『DEFINE』](#)

例: バッチ・コンパイルでのオプションの優先順位

次のサンプル・リストは、バッチ・コンパイルでのコンパイラ・オプションの優先順位を示しています。

```
PP 5655-EC6 IBM Enterprise COBOL for z/OS 6.5.0 PXXXXXXX Date 05/31/2024. . .
Invocation parameters:
NOTERM
PROCESS(CBL) statements:
CBL CURRENCY,FLAG(I,I)
Options in effect: All options are installation defaults unless otherwise noted:
  NOADATA
  ADV
  QUOTE
  ARITH(COMPAT)
  NOAWO
  NOBLOCK0
  BUFSIZE(4096)
  . . .
  CURRENCY      Process option PROGRAM 1
  . . .
  FLAG(I,I)     Process option PROGRAM 1
  . . .
  NOTERM        INVOCATION option
  . . .
End of compilation for program 1
. . .

PP 5655-EC6 IBM Enterprise COBOL for z/OS 6.5.0 PXXXXXXX Date 05/31/2024. . .
PROCESS(CBL) statements:
CBL APOST
Options in effect:
  NOADATA
  ADV
  APOST          Process option PROGRAM 2
  ARITH(COMPAT)
  NOAWO
  NOBLOCK0
  BUFSIZE(4096)
  . . .
  NOCURRENCY     Installation default option for PROGRAM 2
  . . .
  FLAG(I)        Installation default option
  . . .
  NOTERM        INVOCATION option remains in effect
  . . .
End of compilation for program 2
```

例: バッチ・コンパイルでの LANGUAGE オプション

次の例は、バッチ環境での LANGUAGE コンパイラ・オプションの動作を示しています。デフォルトのインストール・オプションは ENGLISH (省略形は EN) で、呼び出しオプションは XX (存在しない言語) です。

```
CBL LANG(JP),FLAG(I,I),APOST (1)
IDENTIFICATION DIVISION.      (2)
PROGRAM-ID. COMPILE1.
. . .
```

```

END PROGRAM COMPILE1.
CBL LANGUAGE(YY) (3)
CBL LANGUAGE(JP),LANG(!!) (4)
IDENTIFICATION DIVISION. (2)
PROGRAM-ID. COMPILE2.
.
.
END PROGRAM COMPILE2.
IDENTIFICATION DIVISION.
PROGRAM-ID. COMPILE3.
.
.
END PROGRAM COMPILE3.
CBL LANGUAGE(JP),LANGUAGE(YY) (5)
. . .

```

(1)

インストールのデフォルトは EN です。呼び出しオプションは XX (存在しない言語) でした。EN は有効な言語です。

(2)

CBL ステートメントの走査後、JP が有効な言語になります。

(3)

CBL は、言語を EN にリセットします。YY は、JP によって置き換えられるため無視されます。

(4)

!! は英数字でなく、破棄されます。

(5)

CBL は、言語を EN にリセットします。YY は JP を置き換えますが、存在しません。

プログラム COMPILE1 の場合、コンパイラーが呼び出しオプションを走査するときには、デフォルト言語 ENGLISH (EN) が有効です。XX は存在しない言語 ID なので、診断メッセージは英大/小文字混合で出されます。デフォルトの EN は、コンパイラーが CBL ステートメントを走査するときにも有効なままです。CBL ステートメントの認識されないオプション APOST は、英大/小文字混合で診断されます。これは、CBL ステートメントが処理を完了しておらず、EN が最後の有効な言語オプションであったからです。コンパイラーが CBL オプションを処理した後は、有効となる言語は日本語 (JP) です。

プログラム COMPILE2 では、最初のプログラムが使用される前に有効な言語は英語なので、コンパイラーは、大/小文字混合の CBL ステートメント・エラーを診断します。LANGUAGE オプションが複数指定された場合は、最後に指定された有効な言語だけが使用されます。この例では、最後の有効な言語は日本語です (JP したがって、コンパイラが処理を終えると、日本語が有効な言語になります。CBL オプション。CBL および PROCESS ステートメントのオプションを日本語で診断したい場合は、COMPILE1 より前に有効な言語が日本語でなければなりません。

プログラム COMPILE3 に CBL ステートメントはありません。したがって、有効な言語である日本語 (JP) を以前のコンパイルから継承します。

コンパイラーは、COMPILE3 をコンパイルした後は、CBL ステートメントのために、有効な言語を英語 (EN) にリセットします。言語オプションは CBL 文は最後に指定された 2 文字の英数字言語識別子を解決し、YY. なぜなら YY 存在しない場合は、実際の言語は英語のままです。

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、エラーが発生した場所を示します (LINEID)。メッセージのテキストには、問題の内容が示されています。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。

このタスクについて

エラーを訂正するとしても、必ずしもすべての診断メッセージのソース・コードを修正する必要があるとは限りません。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、そのメッセージを除去するために、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ので、訂正しなければなりません。

低い方の 4 つのレベルの重大度とは対照的に、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。コンパイラー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、コンパイラーが強制的に早期終了させられ、完全なオブジェクト・コードまたは完全な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ー処理用のストレージの不足など) 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。コンパイラー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

[307 ページの『コンパイラー・メッセージのリストの生成』](#)

関連参照

[307 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』](#)

コンパイラー・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メッセージ番号、重大度、およびテキストの全リストを生成することができます。

このタスクについて

次に示すように、PROGRAM-ID 段落のみをコーディングして、その他のプログラムを省略することができます。

```
Identification Division.  
Program-ID. ErrMsg.
```

関連タスク

[895 ページの『コンパイラー・メッセージの重大度のカスタマイズ』](#)

関連参照

[307 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』](#)

[308 ページの『コンパイラー診断メッセージの形式』](#)

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、診断メッセージを発行します。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。

リストに示されたメッセージには、問題の性質、重大度、およびその問題が検出されたコンパイラー・フェーズについての情報が記載されます。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。

コンパイラー・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。

(行番号で順序付けされた) 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問題の要約は、リストの下部に表示されます。

関連タスク

[306 ページの『ソース・プログラムのエラーの訂正』](#)

[307 ページの『コンパイラー・メッセージのリストの生成』](#)

関連参照

[308 ページの『コンパイラー診断メッセージの形式』](#)

[308 ページの『コンパイラー診断メッセージの重大度コード』](#)

[402 ページの『FLAG』](#)

コンパイラー診断メッセージの形式

コンパイラー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 IGYppxxx-l message-text
```

nnnnnn

コンパイラーが処理していた最終行のソース・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、番号は元のソース・プログラム番号になります。 **NONUMBER** を指定すると、番号はコンパイラーによって生成された番号になります。

IGY

COBOL コンパイラーがメッセージを発行したことを識別する接頭部。

pp

メッセージの原因となった条件が、コンパイラーのどのフェーズまたはサブフェーズで検出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。コンパイラー・エラーの疑いがあると診断した場合は、この情報を含めて IBM にサポートを依頼してください。

xxxx

メッセージを識別する 4 桁の数字。

L

メッセージ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

[308 ページの『コンパイラー診断メッセージの重大度コード』](#)

[402 ページの『FLAG』](#)

コンパイラー診断メッセージの重大度コード

コンパイラーが検出できる条件は、5 つの重大度のレベルまたはカテゴリーに分けられます。

表 40. コンパイラー診断メッセージの重大度コード		
メッセージのレベル またはカテゴリー	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	エラーの可能性が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。

表 40. コンパイラー診断メッセージの重大度コード (続き)		
メッセージのレベル またはカテゴリ	戻りコード	目的
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。

コンパイル終了時の最終の戻りコードは、通常、コンパイル中のメッセージで最も高い戻りコードになります。

コンパイラー診断メッセージを抑制したり、その重大度を変更したりすることができますが、これは、最終のコンパイル戻りコードに影響する可能性があります。詳細については、関連情報を参照してください。

関連タスク

895 ページの『[コンパイラー・メッセージの重大度のカスタマイズ](#)』

関連参照

893 ページの『[MSGEXIT の処理](#)』

第 16 章 z/OS ユニックスのもとでのコンパイル

z/OS ユニックスのもとでは、cob2 コマンドを使用して、Enterprise COBOL プログラムをコンパイルします。z/OS ユニックスのもとでは、z/OS のもとでコンパイルできるすべての COBOL プログラムをコンパイルすることができます。COBOL コンパイラーによって生成されたオブジェクト・コードは、z/OS のもとで実行することができます。

このタスクについて

コンパイル・ステップの一部として、コンパイルに必要なファイルを定義し、プログラムおよび求める出力に必要なコンパイラー・オプションおよびコンパイラー指示ステートメントを指定します。

コンパイラーの主な仕事は、COBOL プログラムを、コンピュータで処理できる言語 (オブジェクト・コード) に変換することです。さらに、コンパイラーは、ソース・ステートメント内のエラーをリストして、プログラムのデバッグおよび調整に役立つ補足情報を提供します。

関連タスク

311 ページの『z/OS ユニックスのもとでの環境変数の設定』

312 ページの『z/OS ユニックスのもとでのコンパイラー・オプションの指定』

313 ページの『cob2 コマンドを使用したコンパイルおよびリンク』

319 ページの『スクリプトを使用したコンパイル』

321 ページの『z/OS ユニックスのもとでの OO アプリケーションのコンパイル、リンク、実行』

関連参照

291 ページの『z/OS のもとでコンパイラーによって使用されるデータ・セット』

301 ページの『z/OS のもとでのコンパイラー・オプションおよびコンパイラー出力』

z/OS ユニックスのもとでの環境変数の設定

環境変数は、文字ストリングに関連付けられる名前であり、プログラム環境のいくつかの変化する側面を定義します。環境変数を使用して、プログラム (コンパイラーを含む) が必要とする値を設定します。

このタスクについて

コンパイラーに必要な環境変数は、export コマンドを使用して設定します。例えば、SYSLIB 変数を設定するには、シェルまたはスクリプト・ファイルから export コマンドを出してください。

```
export SYSLIB=/u/mystuff/copybooks
```

環境変数に割り当てる値には、他の環境変数または変数自体を含めることができます。これらの変数の値が適用されるのは、export コマンドを出すシェルからコンパイルする場合だけです。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。環境変数名は大文字でなければなりません。

コンパイラーが使用するために設定できる環境変数は、次のとおりです。

COBOPT

コンパイラー・オプションをブランクまたはコンマで区切って指定します。サブオプションはコンマで区切ってください。変数値の始まりまたは終わりにあるブランクは無視されます。オプションのリストに、ブランクまたは z/OS ユニックス シェルにとって意味のある文字が含まれている場合には、リストを引用符で囲ってください。例:

```
export COBOPT="TRUNC(OPT) XREF"
```

COBOL_INSTALL_DIR

の cob2 ユーティリティと関連ファイルは通常 HFS ディレクトリの下にインストールされます /usr/lpp/IBM/cobol/igyv6r4。cob2 ユーティリティがシステム上の別の場所にインストールされている場合、このユーティリティを使用するために COBOL_INSTALL_DIR 環境変数をそのロケーションに設定する必要があります。

SYSLIB

COPY ステートメントで明示的なライブラリー名を指定しなかった場合は、COBOL コピーブックの検索の際に使用するディレクトリーへのパスを指定します。複数のパスはコロンで区切ってください。パスは、export コマンドの最初のパスから最後のパスへと順番に評価されます。同じ名前の複数のファイルを持つ変数を設定した場合は、そのファイルの最初に見つかったコピーが使用されます。

のために COPY 明示的なライブラリー名を記述していないステートメントでは、コンパイラはコピーブックを検索します。で z/OS ユニックスディレクトリこの順序で:

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス
4. デフォルト・ライブラリー名を使用するか、SYSLIB の明示ライブラリー名を指定する、COPYLOC オプションのインスタンスで指定された場所

コピーブックがいずれにも見つからない場合は、z/OS ユニックス検索するディレクトリがあり、COPYLOC オプションが明示的なライブラリー名なしで指定されている場合、またはライブラリー名 SYSLIB で指定されている場合、それらの場所は指定された順序で検索されます。

library-name

COPY ステートメントで明示的なライブラリー名を指定する場合の、コピー元の ディレクトリー・パスを指定します。この環境変数名は、プログラム内の *library-name* と同じです。それぞれのライブラリーごとに環境変数を設定しなければなりません。そうでない場合は、エラーになります。環境変数名 *library-name* は大文字でなければなりません。

text-name

テキストのコピー元のファイルの名前を指定します。この環境変数名は、プログラム内の *text-name* と同じです。環境変数名 *text-name* は大文字でなければなりません。

関連タスク

[312 ページの『z/OS ユニックスのもとでのコンパイラー・オプションの指定』](#)

[313 ページの『cob2 コマンドを使用したコンパイルおよびリンク』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

関連参照

[481 ページの『第 21 章 コンパイラー指示ステートメント』](#)

[371 ページの『COBOL コンパイラー・オプション』](#)

COPY ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

z/OS ユニックスのもとでのコンパイラー・オプションの指定

コンパイラーは、デフォルトのコンパイラー・オプションを使用してインストールされ、セットアップされます。コンパイラーのインストール時に、システム・プログラマーは、コンパイラー・オプションの設定を固定して、確実にパフォーマンスを向上させたり、ある特定の標準を維持したりすることができます。インストール先で固定されたコンパイラー・オプションはどれもオーバーライドすることはできません。

このタスクについて

固定されていないオプションについては、以下のいずれかの方法でコンパイラー・オプションを指定して、デフォルト設定をオーバーライドすることができます。

- COBOL ソースの PROCESS または CBL ステートメントにコーディングする。
- cob2 コマンドの -q オプションを指定する。

- COBOPT 環境変数を設定する。

コンパイラーは、上記の優先順位 (高位から下位の順) に従ってオプションを認識します。この優先順位の順序は、対立するオプションまたは互いに排他的なオプションが指定された場合に有効となるオプションも決定します。cob2 コマンドを使用してコンパイルするときは、コンパイラー・オプションは、以下の優先順位 (最高位から最下位へ) の順序で認識されます。

1. オーバーライド不能として固定されているインストール先デフォルト。
2. の価値 BUFSIZE、SQL、SQLIMS、そして OUTDD バッチコンパイルの最初のプログラムに有効なオプション
3. COBOL ソース・プログラム内の PROCESS または CBL ステートメントで指定された値。
4. cob2 コマンドの -q オプション・ストリングで指定された値。
5. COBOPT 環境変数で指定された値。
6. 固定されていないインストール先デフォルト

制約事項:

- z/OS ユニックス から開始されたコンパイル・ジョブで SQL コプロセッサを使用できます。これを機能させるには、以下の条件をすべて満たす必要があります。
 - Db2 コプロセッサ・サービスを含む Db2 データ・セットは、そのサービスが LNKLIST に入っていない場合は、STEPLIB に組み込まれていなければなりません。通常、このデータ・セットは xxxxxx.SDSNLOAD です。例えば、DB2® 11 の場合は DSNB10.SDSNLOAD となりますが、ご使用のインストール済み環境でこの名前が変更されている可能性もあります。
 - SQL コンパイラー・オプションが指定されていなければなりません。
 - cob2 の -dbrmlib オプションが指定されていなければなりません。file はコンパイル対象の入力 COBOL ファイルの名前であるとしします。
 - dbrmlib=xxx を使用して、データベース要求モジュール (DBRM) を既存の PDS データ・セットに送信します。ここで新規メンバー file が作成されます。
 - -dbrmlib (=xxx なし) を使用して、DBRM を HFS ファイル file.dbrm に送信します。

分離型 SQL プリコンパイラーは z/OS ユニックス の下では動作しません。

- z/OS ユニックス で、SQLIMS コンパイラー・オプションを使用しないでください。
- の OPTFILE オプションは、cob2 指揮下 z/OS ユニックス。

代わりに、COBOPT 環境変数を使用できます。これは、OPTFILE に同等の機能を提供します。

関連タスク

[300 ページの『PROCESS \(CBL\) ステートメントでのコンパイラー・オプションの指定』](#)

[311 ページの『z/OS ユニックス のもとでの環境変数の設定』](#)

[313 ページの『cob2 コマンドを使用したコンパイルおよびリンク』](#)

関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

[371 ページの『COBOL コンパイラー・オプション』](#)

cob2 コマンドを使用したコンパイルおよびリンク

z/OS ユニックス シェルから COBOL プログラムのコンパイルおよびリンクを行うには、cob2 コマンドを使用します。オプションおよび入力ファイル名は (オプションと名前をスペースで区切って) どのような順序で指定しても構いません。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。

このタスクについて

複数のファイルをコンパイルする (バッチ・コンパイル) には、複数のソース・ファイル名を指定してください。

z/OS ユニックス の場合、COBOL プログラムをコンパイルするには、RENT オプションが必要になります。cob2 コマンドは、COBOL コンパイラー・オプション RENT および TERM を自動的に組み込みます。

cob2 コマンドは、標準の MVS 探索順序で見つかった COBOL コンパイラーを呼び出します。COBOL コンパイラーが LNKLIST にインストールされていない場合、または複数レベルの IBM COBOL コンパイラーがシステムにインストールされている場合、STEPLIB 環境変数で、使用したいコンパイラー PDSE を指定できます。例えば、次のステートメントは IGY.V6R5M0.SIGYCOMP をコンパイラー PDSE として指定します。

```
export STEPLIB=IGY.V6R5M0.SIGYCOMP
```

cob2 コマンドは、リンク・ステップには z/OS ユニックス シェル・コマンド c89 を暗黙的に使用します。c89 リンカーへのシェルインターフェース (z/OS プログラム管理バインダー)。

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。

サフィックス .cbl を持つファイルだけがコンパイラーに渡されます。cob2 は他のすべてのファイルをリンカーに渡します。

COBOL ソース・プログラム *file.cbl* のコンパイルから要求したリスト出力は、*file.lst* に書き込まれます。リンカーから要求したリスト出力は、*stdout* に書き込まれます。

リンカーによって、最初のメインプログラムから実行が開始されます。

関連タスク

[314 ページの『z/OS ユニックスのもとでの DLL の作成』](#)

[322 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの準備』](#)

[UNIX システム・サービス ユーザーズ・ガイド](#)

関連参照

[315 ページの『cob2 の構文およびオプション』](#)

[318 ページの『cob2 入出力ファイル』](#)

[UNIX システム・サービス・コマンド解説書](#)

z/OS ユニックスのもとでの DLL の作成

z/OS ユニックス シェルから DLL を作成するには、cob2 オプション -bdll を指定する必要があります。

このタスクについて

```
cob2 -o mydll -bdll mysub.cbl
```

cob2 -bdll を指定すると、次のようになります。

- COBOL コンパイラーは、コンパイラー・オプション DLL、EXPORTALL、および RENT (これらは DLL に必要なものです) を使用します。
- リンク・ステップは、DLL によってエクスポートされるそれぞれの名前 ごとに IMPORT 制御ステートメントを含んでいる、DLL 定義サイド・ファイルを作成します。

DLL 定義サイド・ファイルの名前は、出力ファイル名に基づきます。出力名にサフィックスが付いている場合は、そのサフィックスを x で置き換えて、サイド・ファイル名が作られます。例えば、出力ファイル名が *foo.dll* の場合、サイド・ファイル名は *foo.x* になります。

後で DLL を呼び出すモジュールを作成するときに DLL 定義サイド・ファイルを使用するには、リンクする必要のある他の任意のオブジェクト・ファイル (*file.o*) と一緒にサイド・ファイルを指定してください。例えば、次のコマンドは、*myappl.cbl* をコンパイルし、DLL オプションを使用して *myappl.o* が DLL を参照できるようにし、リンクを行ってモジュール *myappl* を作成します。

```
cob2 -o myappl -qdll myappl.cbl mydll.x
```

[315 ページの『例: z/OS ユニックスのもとでの cob2 によるコンパイルおよびリンク』](#)

関連タスク

[639 ページの『第 31 章 DLL または DLL アプリケーションの作成』](#)

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

関連参照

[315 ページの『cob2 の構文およびオプション』](#)

[318 ページの『cob2 入出力ファイル』](#)

例: z/OS ユニックスのもとでの cob2 によるコンパイルおよびリンク

以下の例は、cob2 の使用法を示しています。

- alpha.cbl という名前の 1 つのファイルをコンパイルするには、次のように入力します。

```
cob2 -c alpha.cbl
```

コンパイルされたファイルには、alpha.o という名前が付けられます。

- alpha.cbl および beta.cbl という名前の 2 つのファイルをコンパイルするには、次のように入力します。

```
cob2 -c alpha.cbl beta.cbl
```

コンパイルされたファイルには、alpha.o および beta.o という名前が付けられます。

- 2 つのファイルをリンクするには、-c オプションを指定せずにコンパイルします。例えば、alpha.cbl および beta.cbl をコンパイルしてリンクし、gamma を生成するには、次のように入力します。

```
cob2 alpha.cbl beta.cbl -o gamma
```

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、gamma という名前の実行可能プログラムが作成されます。

- LIST オプションと NOADATA オプションを指定して alpha.cbl をコンパイルするには、次のように入力します。

```
cob2 -qlist,noadata alpha.cbl
```

- SQL オプションを指定して alpha.cbl をコンパイルし、データベース要求モジュール (DBRM) が既存の PDS データ・セット USER.COBOL.DBRMLIB のメンバー "alpha" に書き込まれるようにするには、次のように入力します。

```
cob2 -qsql alpha.cbl -o alpha -dbrmllib=USER.COBOL.DBRMLIB
```

注: これが機能するためには、SQL コプロセッサが STEPLIB になければなりません。

- SQL オプションを指定して alpha.cbl をコンパイルし、データベース要求モジュール (DBRM) を z/OS UNIX ファイル alpha.dbrm に書き込むには、次のように入力します。

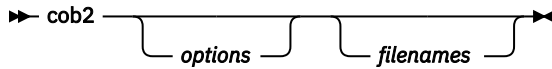
```
cob2 -qsql alpha.cbl -o alpha -dbrmllib
```

注: これを機能させるには、SQL コプロセッサが STEPLIB になければなりません。また、APAR PI88171 が適用された Db2 12 用の SQL コプロセッサを使用する必要があります。

cob2 の構文およびオプション

cob2 コマンドでは以下のオプションを使用することができます。(cob2 を大文字で使わないでください。)

cob2 コマンドの構文



オプションも入力ファイルも指定せずに `cob2` を指定すると、コンパイラー・マニュアル・ページが表示されます。

-bxxx

ストリング `xxx` をパラメーターとしてリンカーに渡します。 `xxx` は、コンマで区切られた、*name=value* 形式のリンカー・オプションのリストです。名前と値の両方を (以下に示す特別な場合を除いて) 略さずにフルスペルで書く必要があります。名前と値は、大/小文字が区別されません。 `-b` と `xxx` の間にスペースを入れてはなりません。

オプションの値を指定しなかった場合は、デフォルト値として `YES` が使用されます。ただし、以下のオプションは例外で、これらは指定されたデフォルト値を持っています。

- `LIST=NOIMPORT`
- `ALIASES=ALL`
- `COMPAT=CURRENT`
- `DYNAM=DLL`

`xxx` に対する 1 つの特殊値は `dll` で、実行可能モジュールが DLL であることを指定します。このストリングはリンカーに渡されません。

-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 `cob2` 動作を制御します。戻りコードが `n` 以下であれば、`cob2` 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。コンパイラーから戻された戻りコードが `n` より大きい場合は、`cob2` は同じ戻りコードで終了します。リンク・ステップの `cob2` によって `c89` コマンドが暗黙的に呼び出された場合は、`c89` コマンドからの終了値が、`cob2` コマンドからの戻りコードとして使用されます。

デフォルトは `-comprc_ok=4` です。

-dbrmllib=xxx

生成されたデータベース要求モジュール (DBRM) に使用されるロケーションを指定します。SQL コンパイラー・オプションも指定されている場合のみ有効です。

- `xxx` が指定されないと、DBRM は z/OS ユニックス ファイルに書き込まれます。コンパイル操作の入力ファイルの名前が `file.cbl` であれば、DBRM ファイルの名前は `file.dbrm` になります。
- `xxx` が指定される場合、この `xxx` は、生成された DBRM を保持する既存のデータ・セットの名前を表します。指定された名前 `xxx` は、それ以上の修飾は行われず、そのままの状態で使用されることに注意してください。

-e xxx

モジュールの入り口点として使用するプログラムの名前を指定します。このプログラムは、モジュールに組み込まれるプログラムのいずれかでなければなりません。 `-e` を指定しなかった場合、デフォルト入り口点は、`cob2` コマンド呼び出しでファイル名として指定された最初のプログラム (`file.cbl`) またはオブジェクト・ファイル (`file.o`) になります。

-g

デバッグに備えてプログラムを準備します。サブオプションなしで `TEST` オプションを指定するのと等価です。

-help

コンパイラーのマニュアル・ページを表示します。cob2 -help を指定すると、入力ファイルが指定されたかどうかに関係なく、コンパイラー・マニュアル・ページが表示され、コンパイルは停止します。このオプションには、-? と同じ効果があります。

-Ixxx

library-name が指定されていない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。

複数のパスを指定するには、複数の -I オプションを使用するか、単一の -I オプション値の中に複数のパス名をコロンで区切って指定してください。

COPY ステートメントに明示的なライブラリー名がコード化されていない場合、コンパイラーは、以下の順序でコピーブックを検索します。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス
4. デフォルト・ライブラリー名を使用するか、SYSLIB の明示ライブラリー名を指定する、COPYLOC オプションのインスタンスで指定された場所

-L xxx

-l オペランドで指定されたアーカイブ・ライブラリーを探索するために使用するディレクトリー・パスを指定します。

-l xxx

リンカーのためのアーカイブ・ライブラリーの名前を指定します。cob2 コマンドは、libxxx.a という名前を、-L オプションで指定されたディレクトリーで探索し、そのあと通常の探索順序で探索します。(このオプションは、小文字の l です。大文字の I ではありません。)

-M

コンパイラに入力されるソース・ファイル用の、すべての z/OS UNIX ベースのコピーブック・ファイルのパスとファイル名を含む make 依存ファイルを生成する。以下に、出力例を示します。

```
mkdepend.o: mkdepend.cbl
mkdepend.o: /home/usera/test/f171592/WSCOPY1.cpy
mkdepend.o: /home/usera/test/f171592/PDCOPY1.cpy
mkdepend.o: /home/usera/test/f171592/PDCOPY2.cpy
```

-o xxx

オブジェクト・モジュール xxx を指定します。-o オプションが使用されない場合、オブジェクト・モジュールの名前は a.out になります。

-qxxx

xxx をコンパイラーに渡します (xxx は、ブランクまたはコンマで区切られたコンパイラー・オプションのリストです)。

括弧がオプションまたはサブオプションの一部である場合、またはブランクを使用してオプションを区切る場合は、xxx を引用符で囲んでください。-q と xxx の間にスペースを入れないでください。

-v

コンパイル・ステップおよびリンク・ステップで cob2 によって出される生成済みコマンド (渡されるオプションを含む) を表示し、それらを実行します。以下に、出力例を示します。

```
cob2 -v -o mini -qssrange mini.cbl
compiler: ATTCRCTL PARM=RENT,TERM,SSRANGE /u/userid/cobol/mini.cbl
PP 5655-EC6 IBM Enterprise COBOL for z/OS 6.5.0 in progress ...
End of compilation 1, program mini, no statements flagged.
linker: /bin/c89 -o mini -e // mini.o
```

-#

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。

-?

コンパイラーのマニュアル・ページを表示します。cob2 -? を指定すると、入力ファイルが指定されたかどうかに関係なく、コンパイラー・マニュアル・ページが表示され、コンパイルは停止します。このオプションには、-help と同じ効果があります。

関連タスク

[313 ページの『cob2 コマンドを使用したコンパイルおよびリンク』](#)

[314 ページの『z/OS ユニックスのもとでの DLL の作成』](#)

[311 ページの『z/OS ユニックスのもとでの環境変数の設定』](#)

cob2 入出力ファイル

cob2 コマンドを使用するときは、入力ファイル名として以下のファイルを指定することができます。

表 41. cob2 コマンドへの入力ファイル		
ファイル名	説明	コメント
file.cbl	コンパイルおよびリンクされる COBOL ソース・ファイル	cob2 オプション -c を指定した場合は、リンクされません。
file.a	アーカイブ・ファイル	リンク・エディット・フェーズで使用できるように、ar コマンドによって作成されます。
file.o	リンク・エディットされるオブジェクト・ファイル	COBOL コンパイラー、C/C++ コンパイラー、またはアセンブラーによって作成することができます。
file.x	DLL 定義サイド・ファイル	ダイナミック・リンク・ライブラリー (DLL) を参照するアプリケーションのリンク・エディット・フェーズで使用されます。

cob2 コマンドを使用すると、以下のファイルが現行ディレクトリーに作成されます。

表 42. cob2 コマンドからの出力ファイル		
ファイル名	説明	コメント
file	実行可能モジュールまたは DLL	cob2 オプション -o file を指定した場合に、リンカーによって作成されます。
a.out	実行可能モジュールまたは DLL	cob2 オプション -o を指定しなかった場合に、リンカーによって作成されます。
file.adt	入力 COBOL ソース・プログラム file.cbl に対応する関連データ (ADATA) ファイル	コンパイラー・オプション ADATA を指定した場合に、コンパイラーによって作成されます。
file.dbg	入力 COBOL ソース・プログラム file.cbl に対応する、デバッグ・ツール用の記号情報テーブル	コンパイラー・オプション TEST(. . . ,SEP,. . .) を指定した場合に、コンパイラーによって作成されます。
file.dbrm	データベース要求モジュール (DBRM)	-dbrmlib オプションを単独で (xxx なしで) 指定した場合に、コンパイラーによって作成されます。
file.dek	ライブラリー処理からの拡張 COBOL ソース出力	コンパイラー・オプション MDECK を指定した場合に、コンパイラーによって作成されます。
file.lst	入力 COBOL ソース・プログラム file.cbl に対応するリスト・ファイル	コンパイラーによって作成されます。
file.o	入力 COBOL ソース・プログラム file.cbl に対応するオブジェクト・ファイル	コンパイラーによって作成されます。

表 42. cob2 コマンドからの出力ファイル (続き)		
ファイル名	説明	コメント
ファイル.u	現在のコンパイルに必要な、z/OS UNIX -ベースのコピーブック・ファイルのパスとファイル名を含む依存ファイルを作成する	コンパイラーによって作成されます。
file.x	DLL 定義サイド・ファイル	file.dll の作成時に cob2 リンク・フェーズで作成されます。
class.java	Java クラス定義 (ソース)	クラス定義をコンパイルしたときに作成されます。

関連タスク

313 ページの『cob2 コマンドを使用したコンパイルおよびリンク』

関連参照

378 ページの『ADATA』

419 ページの『MDECK』

448 ページの『TEST』

UNIX システム・サービス・コマンド解説書

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 タスクを自動化するためには、無効なストリングをシェルが cob2 に渡さないようにするため、オプション構文を慎重にコーディングしてください。

このタスクについて

次のようにスクリプトでオプション・ストリングをコーディングします。

- コンパイラー・サブオプションを指定するには、左括弧および右括弧ではなく、それぞれ、等号とコロンの使用してください。たとえば、コード -qOPTIMIZE=1:XREF の代わりに -qOPTIMIZE(1),XREF。
- コンパイラー・オプションのサブオプションを区切るのにアポストロフィが必要な場合は、アポストロフィではなく下線を使用してください。
- オプション・ストリングの中でブランクを使用しないでください。

第 17 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、z/OS ユニックス 環境で行うことが推奨されます。関連タスクで説明されている制限はありますが、標準のバッチ JCL または TSO/E コマンドを使用して、OO COBOL アプリケーションをコンパイル、リンク、または実行することも可能です。

このタスクについて

関連タスク

[321 ページの『z/OS ユニックスのもとでの OO アプリケーションのコンパイル、リンク、実行』](#)

[325 ページの『JCL または TSO/E での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』](#)

[329 ページの『Java SDKs for z/OS の使用』](#)

z/OS ユニックスのもとでの OO アプリケーションのコンパイル、リンク、実行

z/OS ユニックス 環境で、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、および実行を行うとき、アプリケーション・コンポーネントは、z/OS UNIX ファイルシステムに存在します。これらは、シェル・コマンドを使用してコンパイルおよびリンクし、シェル・コマンド・プロンプトで、または JCL または TSO/E から BPXBATCH ユーティリティを使用して、実行します。

このタスクについて

関連タスク

[321 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションのコンパイル』](#)

[322 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの準備』](#)

[324 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行』](#)

z/OS ユニックスのもとでのオブジェクト指向アプリケーションのコンパイル

z/OS ユニックス シェルで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 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。

このタスクについて

次のようなオブジェクト指向構文を含む COBOL ソースコードをコンパイルするには、INVOKE ステートメントまたはクラス定義、またはジャワサービスを利用するにはこれらのコンパイラオプション:RENT、DLL、THREAD、そして DBCS。（RENT そして DBCS オプションはデフォルトです。）

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。指定できるのは include サブディレクトリを使用して -I オプションの cob2 コマンドを実行するか、SYSLIB 環境変数を設定します。

関連タスク

- [311 ページの『第 16 章 z/OS ユニックスのもとでのコンパイル』](#)
- [322 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの準備』](#)
- [324 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行』](#)
- [580 ページの『環境変数の設定およびアクセス』](#)
- [785 ページの『JNI サービスへのアクセス』](#)

関連参照

- [315 ページの『cob2 の構文およびオプション』](#)
- [392 ページの『DBCS』](#)
- [396 ページの『DLL』](#)
- [435 ページの『RENT』](#)
- [453 ページの『THREAD』](#)

z/OS ユニックスのもとでのオブジェクト指向アプリケーションの準備

使用 cob200 COBOL アプリケーションをリンクするコマンド。

このタスクについて

オブジェクト指向 COBOL クライアント・プログラムを実行できるよう準備するには、オブジェクト・ファイルと次の 2 つの DLL サイド・ファイルをリンクして、実行可能モジュールを作成します。

- libjvm.x。IBM Java ソフトウェア開発キット で提供されています。
- igzccjava.x、これは libCOBOL インストールディレクトリのサブディレクトリ z/OS UNIX ファイルシステム。この DLL サイド・ファイルは、SCEELIB PDS (言語環境の一部) のメンバー IGZCJAVA としても使用可能です。

COBOL クラス定義を実行できるように準備する。

手順

- 上記の 2 つの DLL サイド・ファイルを使用して、オブジェクト・ファイルをリンクし、実行可能 DLL モジュールを作成します。

結果として生成される DLL モジュールの名前は、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、DLL モジュール名ではピリオドを下線に変更する必要があります。たとえば、クラス Account が com.acme パッケージの外部クラス名 (REPOSITORY クラスの段落エントリ) は com.acme.Account、そしてその DLL モジュールクラスは libcom_acme_Account.so。

- 生成された Java ソースを Java コンパイラーでコンパイルして、クラス・ファイル (.class) を生成します。

タスクの結果

Classname 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 43. 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 -c -qdll,thread Classname.cbl	Classname.cbl	Classname.o, Classname.java

表 43. クラス定義のコンパイルおよびリンクのコマンド (続き)		
コマンド	入力	出力
cob2 -bdll -o libClassname.so Classname.o /usr/lpp/java/IBM/J8.0/bin/j9vm/libjvm.x /usr/lpp/cobol/igyv6r4/lib/igzccjava.x	Classname.o	libClassname.so
javac Classname.java	Classname.java	Classname.class

cob2 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、実行可能 DLL モジュール 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。

323 ページの『例: z/OS ユニックス の下での COBOL クラス定義のコンパイルおよびリンク』

関連タスク

311 ページの『第 16 章 z/OS ユニックス のもとでのコンパイル』

748 ページの『クラス定義用の REPOSITORY 段落』

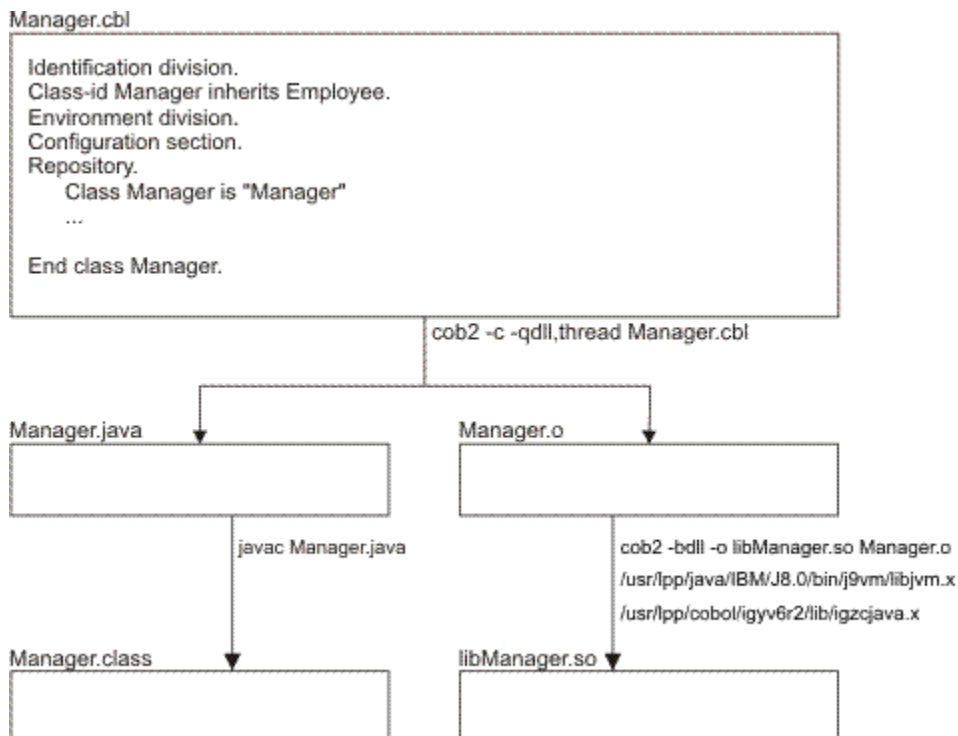
関連参照

315 ページの『cob2 の構文およびオプション』

330 ページの『オブジェクト指向構文、および Java 6 以降』

例: z/OS ユニックス の下での COBOL クラス定義のコンパイルおよびリンク

この例では、z/OS ユニックス シェル・コマンドを使用して、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および生成されるファイル、Manager.cbl を示しています。



クラス・ファイル Manager.class および DLL モジュール libManager.so は、アプリケーションの実行可能なコンポーネントであり、現行作業ディレクトリーで生成されます。

z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行

z/OS ユニックス アプリケーションとしてオブジェクト指向 COBOL アプリケーションを実行することを推奨します。アプリケーションがジャワプログラムまたは mainCOBOL クラスのファクトリ メソッド。

このタスクについて

COBOL クラスのための DLL 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。

注: CLASSPATH 環境変数にファイル名のワイルドカード (*) が含まれていても、COBOL ランタイムで JVM が初期化された場合、ワイルドカードは展開されません。

関連タスク

[324 ページの『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』](#)

[325 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』](#)

[325 ページの『J2EE COBOL クライアントの実行』](#)

[579 ページの『第 26 章 z/OS ユニックスのもとでの COBOL プログラムの実行』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

[743 ページの『第 40 章 オブジェクト指向プログラムの作成』](#)

[782 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 とジャワアプリケーションは main の方法ジャワクラスまたは mainCOBOL クラスのファクトリーメソッドを使用してアプリケーションを実行するには、java コマンドと、そのコマンドを含むクラスの名前を指定することによって、main 方法。

このタスクについて

の java コマンドは初期化しますジャワ仮想マシン (JVM)。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 44. 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・プロパティーを指定する	-Dname=value
ガーベッジ・コレクションに関する詳細メッセージを JVM が生成するよう要求する	-verbose:gc
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	-verbose:class
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・アクティビティー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	-verbose:jni
Java の初期ヒープ・サイズを value バイトに設定する	-Xmsvalue
Java の最大ヒープ・サイズを value バイトに設定する	-Xmxvalue

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

[IBM SDK for Java - Tools Documentation](#)

[WebSphere for z/OS: アプリケーション](#) (Java のネーミングおよびディレクトリー・インターフェース (JNDI))

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。

このタスクについて

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。例:

```
export COBJVMINITOPTIONS="-Xms100000000 -Xmx200000000 -verbose:gc"
```

関連タスク

[329 ページの『Java SDKs for z/OS の使用』](#)

[579 ページの『第 26 章 z/OS ユニックスのもとでの COBOL プログラムの実行』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

関連参照

[IBM SDK for Java - Tools Documentation](#)

[WebSphere for z/OS: アプリケーション](#) (Java のネーミングおよびディレクトリー・インターフェース (JNDI))

J2EE COBOL クライアントの実行

COBOL プログラムでオブジェクト指向構文を使用すると、Java 2 Platform, Enterprise Edition (J2EE) クライアントをインプリメントできます。例えば、WebSphere for z/OS 環境で稼働する Enterprise Bean 上でメソッドを呼び出すことができます。

このタスクについて

COBOL J2EE クライアントを実行する前に、Java システム・プロパティ `java.naming.factory.initial` を設定し、WebSphere のネーム・サービスにアクセスする必要があります。例:

```
export COBJVMINITOPTIONS
="-Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory"
```

[797 ページの『例: COBOL で書かれた J2EE クライアント』](#)

JCL または TSO/E での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

オブジェクト指向構文を使用するアプリケーションのコンパイル、リンク、および実行は、z/OS ユニックス環境で行うことが推奨されます。

このタスクについて

ただし、限られた環境では、標準のバッチ JCL または TSO/E 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コンパイル、リンク、または実行することも可能です。そのためには、関連タスクにある

ガイドラインに従う必要があります。例えば、COBOL のメインプログラムとサブプログラムで構成され、次のような機能を持つアプリケーションの場合は、この方法に従う必要があります。

- すべて Java でインプリメントされたオブジェクトにアクセスする。
- WebSphere サーバーで稼働する Enterprise Bean にアクセスする。

関連タスク

[326 ページの『JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル』](#)

[327 ページの『JCL または TSO/E でのオブジェクト指向アプリケーションの準備と実行』](#)

[321 ページの『z/OS ユニックス のもとでの OO アプリケーションのコンパイル、リンク、実行』](#)

JCL または TSO/E でのオブジェクト指向アプリケーションのコンパイル

バッチ JCL または TSO/E を使用して、OO COBOL プログラムまたはクラス定義をコンパイルする場合、生成されたオブジェクト・ファイルは、通常どおり、DD 名 SYSLIN または SYSPUNCH を持つデータ・セットに書き込まれます。コンパイラー・オプション RENT、DLL、THREAD、および DBCS を使用する必要があります。(RENT および DBCS はデフォルトです。)

このタスクについて

COBOL プログラムまたはクラス定義が、JNI 環境構造を使用して JNI 呼び出し可能サービスにアクセスする場合は、SIGYLIB データ・セットを SYSLIB DD ステートメントのデータ・セット連結に追加し、COBOL ソースで COPY JNI 形式の COPY ステートメントを使用します。

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、COBOL クラス定義に対応するクラス定義を含む Java ソース・プログラムが、オブジェクト・ファイルに加えて生成されます。SYSJAVA DD 名を使用して、生成された Java ソース・ファイルを z/OS UNIX ファイルシステム内のファイルに書き込みます。例:

```
//SYSJAVA DD PATH='/u/userid/java/Classname.java',  
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),  
// PATHMODE=SIRWXU,  
// FILEDATA=TEXT
```

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。

z/OS ユニックス シェル・コマンド・プロンプトから javac コマンドを使用するか、BPXBATCH ユーティリティーを使用して、Java クラス定義をコンパイルします。

[328 ページの『例: JCL による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、実行』](#)

関連タスク

[279 ページの『JCL を使用したコンパイル』](#)

[287 ページの『TSO のもとでのコンパイル』](#)

[296 ページの『ソース・ライブラリーの指定 \(SYSLIB\)』](#)

[297 ページの『Java ソース出力ファイル \(SYSJAVA\) の定義』](#)

[785 ページの『JNI サービスへのアクセス』](#)

[321 ページの『z/OS ユニックス のもとでのオブジェクト指向アプリケーションのコンパイル』](#)

[322 ページの『z/OS ユニックス のもとでのオブジェクト指向アプリケーションの準備』](#)

関連参照

[392 ページの『DBCS』](#)

[396 ページの『DLL』](#)

[435 ページの『RENT』](#)

[453 ページの『THREAD』](#)

[906 ページの『JNI.cpy コピーブック』](#)

UNIX システム・サービス ユーザーズ・ガイド (BPXBATCH ユーティリティー)

JCL または TSO/E でのオブジェクト指向アプリケーションの準備と実行

オブジェクト指向アプリケーションは、z/OS ユニックス 環境で実行することを推奨します。 バッチ JCL または TSO/E からオブジェクト指向アプリケーションを実行するには、BPXBATCH ユーティリティを使用する必要があります。

このタスクについて

ただし、限られた環境においては、標準バッチ JCL (EXEC PGM=COBPROG) または TSO/E の CALL 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できます。 そのためには、アプリケーションを準備する際に、以下の要件に従う必要があります。

- COBOL プログラムで開始するようにアプリケーションを構成します。(アプリケーションが、Java プログラムまたは COBOL クラスの main ファクトリー・メソッドで始まる場合は、z/OS ユニックス でアプリケーションを実行し、アプリケーション・コンポーネントを z/OS UNIX ファイルシステム に格納する必要があります。)
- リンク・エディットの考慮事項: COBOL プログラムのプログラム・オブジェクトを PDSE にリンクします。 オブジェクト指向構文を含む COBOL プログラムは、AMODE 31。
- アプリケーションが使用する COBOL または Java のクラスに関連付けられたクラス・ファイルおよび DLL は、z/OS UNIX ファイルシステム に存在していなければなりません。 z/OS UNIX のもとでの オブジェクト指向アプリケーションの準備に関する関連タスクで説明されているように、クラス・ファイルと DLL の名前を設定する必要があります。
- メインプログラムにオブジェクト・デックをバインドするときは、DLL サイド・ファイルの libjvm.x と igzccjava.x に対して、INCLUDE 制御ステートメントを指定します。 例:

```
INCLUDE '/usr/lpp/java/IBM/J8.0/bin/j9vm/libjvm.x'  
INCLUDE '/usr/lpp/cobol/igyv6r4/lib/igzccjava.x'
```

- Java に必要な環境変数設定を含むファイルを作成します。例えば、ファイル /u/userid/javaenv には、PATH、LIBPATH、および CLASSPATH 環境変数を設定するために、以下に示す 3 行が含まれていることがあります。

```
PATH=/bin:/usr/lpp/java/IBM/J8.0/bin  
LIBPATH=/lib:/usr/lib:/usr/lpp/java/IBM/J8.0/bin:/usr/lpp/java/IBM/J8.0/bin/j9vm  
CLASSPATH=./u/userid/applications
```

アプリケーションで使用する JVM の初期設定をカスタマイズするには、同じファイルで COBJVMINITOPTIONS 環境変数を設定します。例えば、WebSphere サーバーで動作する Enterprise Bean にアクセスするには、Java システム・プロパティ `java.naming.factory.initial` を設定する必要があります。詳細については、オブジェクト指向アプリケーションの実行に関する関連タスクを参照してください。

COBOL プログラムで開始するオブジェクト指向アプリケーションを、標準バッチ JCL または TSO/E の CALL コマンドを使用して実行するときは、以下の指針に従ってください。

- CEE_ENVFILE 環境変数を使用して、Java で必要な環境変数設定を含むファイルの場所を指定します。 _CEE_ENVFILE を設定するには、ENVAR ランタイム・オプションを使用します。
- POSIX(ON) および XPLINK(ON) ランタイム・オプションを指定します。
- DD ステートメントを使用して、Java の標準入力、出力、およびエラーのストリームに対する z/OS UNIX ファイルシステム 中のファイルを指定します。
 - `c=System.in.read();` などのステートメントからの 入力に対しては JAVAIN DD を使用します。
 - `System.out.println(string);` などのステートメントからの出力に対しては JAVAOUT DD を使用します。
 - `System.err.println(string);` などのステートメントからの出力に対しては JAVAERR DD を使用します。

- 確実に SCEERUN2 SCEERUN ロードライブラリは、例えば、STEPLIB DD 声明。

328 ページの『例: JCL による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、実行』

関連タスク

322 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの準備』

324 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行』

782 ページの『OO アプリケーションの構造化』

UNIX システム・サービス ユーザーズ・ガイド (BPXBATCH ユーティリティ)

言語環境プログラミングガイド (バッチでのアプリケーションの実行)

関連参照

XL C/C++ プログラミング・ガイド (_CEE_ENVFILE)

言語環境プログラミング・リファレンス (ENVAR)

例: JCL による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、実行

この例は、Java メソッドを呼び出す COBOL クライアントの、コンパイル、リンク、および実行に使用できるサンプル JCL を示しています。

この例は次のものを示しています。

- オブジェクト指向 COBOL プログラムをコンパイル、リンク、および実行するための JCL である TSTHELLO。
- COBOL プログラムが起動するメソッドを含んでいる Java クラス定義 HelloJ。
- Java が必要とする環境変数の設定値を含んでいる z/OS ユニックスファイル ENV。

プログラム TSTHELLO に対する JCL

```
//TSTHELLO JOB ,
//  TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,REGION=200M,
//  NOTIFY=&SYSUID,USER=&SYSUID
//*
// SET COBPRFX='IGY.V6R5M0'
// SET LIBPRFX='CEE'
//*
//COMPILE EXEC PGM=IGYCRCTL,
//SYSLIN DD DSN=&&OBJECT(TSTHELLO),UNIT=VIO,DISP=(NEW,PASS),
//        SPACE=(CYL,(1,1,1))
//SYSPRINT DD SYSOUT=*
//STEPLIB DD DSN=&COBPRFX..SIGYCOMP,DISP=SHR
//        DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//        DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSUT1 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT8 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT9 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT10 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT11 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT12 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT13 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT14 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT15 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSMDECK DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSIN DD *
        cbl dll,thread
        Identification division.
        Program-id. "TSTHELLO" recursive.
        Environment division.
        Configuration section.
        Repository.
            Class HelloJ is "HelloJ".
        Data Division.
        Procedure division.
            Display "COBOL program TSTHELLO entered"
            Invoke HelloJ "sayHello"
```

```

                Display "Returned from java sayHello to TSTHELLO"
                Goback.
            End program "TSTHELLO".
/*
//LKED EXEC PGM=IEWL,PARM='RENT,LIST,LET,DYNAM(DLL),CASE(MIXED)'
//SYSLIB DD DSN=&LIBPRFX..SCEELKED,DISP=SHR
// DD DSN=&LIBPRFX..SCEELKEX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSLMOD DD DSN=&&GOSET(TSTHELLO),DISP=(MOD,PASS),UNIT=VIO,
// SPACE=(CYL,(1,1,1)),DSNTYPE=LIBRARY
//SYSDEFSD DD DUMMY
//OBJMOD DD DSN=&&OBJECT,DISP=(OLD,DELETE)
//SYSLIN DD *
// INCLUDE OBJMOD(TSTHELLO)
// INCLUDE '/usr/lpp/java/IBM/J8.0/bin/j9vm/libjvm.x'
// INCLUDE '/usr/lpp/cobol/igv6r4/lib/igzjava.x'
/*
//GO EXEC PGM=TSTHELLO,COND=(4,LT,LKED),
// PARM='/ENVAR("_CEE_ENVFILE=/u/userid/octest/tsthello/ENV")
// POSIX(ON) XPLINK(ON)'
//STEPLIB DD DSN=*.LKED.SYSLMOD,DISP=PASS
// DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD DUMMY
//JAVAOUT DD PATH='/u/userid/octest/tsthello/javaout',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR,SIRGRP)
//JAVAERR DD PATH='/u/userid/octest/tsthello/javaerr',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR,SIRGRP)

```

クラス *HelloJ* の定義

```

class HelloJ {
    public static void sayHello() {
        System.out.println("Hello World, from Java!");
    }
}

```

HelloJ.java は、javac コマンドを使用してコンパイルされます。結果として得られる.class ファイルは、z/OS UNIX ファイルシステムディレクトリ `u/userid/octest/tsthello`、これは **CLASSPATH** 環境変数設定ファイル内の環境変数。

環境変数設定ファイル *ENV*

```

PATH=/bin:/usr/lpp/java/IBM/J8.0/bin
LIBPATH=/lib:/usr/lib:/usr/lpp/java/IBM/J8.0/bin:/usr/lpp/java/IBM/J8.0/bin/j9vm
CLASSPATH=./u/userid/octest/tsthello

```

環境変数設定ファイルは、`u/userid/octest/tsthello` ディレクトリーにも存在しています。このディレクトリーは、JCL の **_CEE_ENVFILE** 環境変数で指定されています。

Java SDKs for z/OS の使用

Java SDKs for z/OS は、言語環境で定義されている XPLINK リンケージ規約に基づきます。

このタスクについて

アプリケーションがジャワプログラムまたは mainCOBOL クラスのファクトリーメソッドでは、XPLINK 環境は自動的に起動されます。java JVM を起動してアプリケーションを実行するコマンド。

アプリケーションが COBOL プログラムで始まり、COBOL またはジャワクラスを指定する場合は、XPLINK(ON) XPLINK 環境が初期化されるようにランタイム オプションを設定します。XPLINK(ON) た

だし、デフォルト設定としては推奨されません。XPLINK(ON)特にそれを必要とするアプリケーションにのみ適用されます。

アプリケーションを実行している場合 z/OS ユニックス、設定することができます
XPLINK(ON)_CEE_RUNOPTS 環境変数を使用して次のようにオプションを設定します。

```
_CEE_RUNOPTS="XPLINK(ON)"
```

エクスポート _CEE_RUNOPTS="XPLINK(ON)" 全体に適用されるように z/OS ユニックスただし、シェルセッションは推奨されません。例えば、OO COBOL アプリケーションが App1Driver という名前の COBOL プログラムから開始されるとしましょう。XPLINK オプションの影響を App1Driver アプリケーションの実行に限定する方法の 1 つは、App1Driver のコマンド行呼び出しで _CEE_RUNOPTS 変数を次のように設定することです。

```
_CEE_RUNOPTS="XPLINK(ON)" App1Driver
```

関連タスク

[324 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

関連参照

[330 ページの『オブジェクト指向構文、および Java 6 以降』](#)

[581 ページの『ランタイム環境変数』](#)

言語環境プログラミング・リファレンス (XPLINK)

XL C/C++ プログラミング・ガイド (_CEE_RUNOPTS)

オブジェクト指向構文、および Java 6 以降

Java インターオペラビリティのためにオブジェクト指向構文を使用する Enterprise COBOL 5.2 以降のアプリケーションは、Java 6 以降でサポートされます。

Java インターオペラビリティのためにオブジェクト指向構文を使用する旧バージョンの Enterprise COBOL アプリケーションは、Java SDK 1.4.2 および Java 5 でサポートされていました。これらのアプリケーションを Java 6 以降で実行するには、以下の手順を実行します。

1. Enterprise COBOL 5.2 以降を使用してアプリケーションを再コンパイルおよび再リンクします。
2. 生成されたものを再コンパイルするジャワ各オブジェクト指向 COBOL クラスに関連付けられたクラス。javac からの命令ジャワ 6 以降。

関連タスク

[322 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの準備』](#)

第 18 章 Java と相互運用する非 OO COBOL アプリケーションのコンパイル、リンク、および実行

Enterprise COBOL は、オブジェクト指向 (OO) COBOL フレームワークによって提供されるものよりも包括的で柔軟な形式の Java 相互相互運用性をサポートします。

相互運用可能な非 OO アプリケーションの構築ジャワ

このセクションでは、組み込みの Java インターオペラビリティ機能を使用して、Java と相互運用する非 OO COBOL アプリケーションの構築について説明します。アプリケーションの一部である既存の COBOL プログラムは、通常通りにコンパイルできますが、新しいリンクに関する考慮事項が必要になる場合があります。COBOL と Java の間のインターオペラビリティを容易にする「グルー・コード」を実行する際に、アプリケーションが実行時に必要とする DLL へのコンパイラ生成スタブ・ファイルの構築を支援するために、cjbuild というユーティリティ・プログラムが提供されています。

このタスクについて

一般に、Java と相互運用する COBOL アプリケーションの構築は、以下の 4 つのステージで行われます。

1. アプリケーション内のすべてのユーザー プログラムをコンパイルします。非 OO COBOL/Java 相互運用性機能（すなわち、JAVA-CALLABLE 指示文、JAVA-SHAREABLE 指示文、静的 Java メソッドの呼び出し）を含むプログラムは、ステップ 3 でリンクされなければならない。しかし、OO 以外の COBOL/Java 相互運用機能を含まないユーザー・プログラムは、ステップ 3 でリンクしなければならないプログラムへの依存関係がない限り、このステップで通常通りリンクすることができます。

注：アプリケーション内の COBOL プログラムに 1 つ以上の Java インターオペラビリティ構成が含まれている場合、Java とのインターオペラビリティを容易にするために、コンパイラ生成の COBOL 「スタブ・プログラム」が自動的に生成されます。これらのスタブ・プログラムが生成される原因となる COBOL 構成は、以下の通りです。

- JAVA-CALLABLE ディレクティブ。結果として COBOL ネイティブ・メソッド・スタブ・プログラムになります。
- JAVA-SHAREABLE 指令により、作業用ストレージ初期設定スタブ・プログラム、および COBOL プログラム中の共用可能なすべての WORKING-STORAGE データ項目にアクセスするためのインターフェースを提供する Java クラスを生成します。
- CALL ステートメントに「Java.Java クラス名.java-static-method-name」の形式のリテラル呼び出しターゲットを指定すると、Java 呼び出しスタブ・プログラムが、固有の Java 呼び出しターゲットごとに 1 つずつ作成されます。

作成されるすべてのスタブ・ファイルは、後続のステップでコンパイルされ、リンクされます。簡単にするため、1 つの相互運用アプリケーションのすべてのスタブファイルは、z/OS UNIX の同じディレクトリに置くことを推奨する。これは、相互運用アプリケーションの各 COBOL プログラムをビルドするときに、JAVAIOP コンパイラ・オプションの OUTPATH サブオプションで同じ出力ディレクトリを指定する必要があることを意味する。このディレクトリは、cjbuild の -c/--cobol-dir オプションの引数として指定する。

2. ステップ 1 で作成したすべての COBOL ネイティブ・メソッド・スタブ・プログラム、WORKING-STORAGE 初期設定スタブ・プログラム、および Java 呼び出しスタブ・プログラムから DLL を構築するには、cjbuild ユーティリティを使用します。この DLL のサイド・デックも生成され、ステップ 3 でリンクするために必要となります。

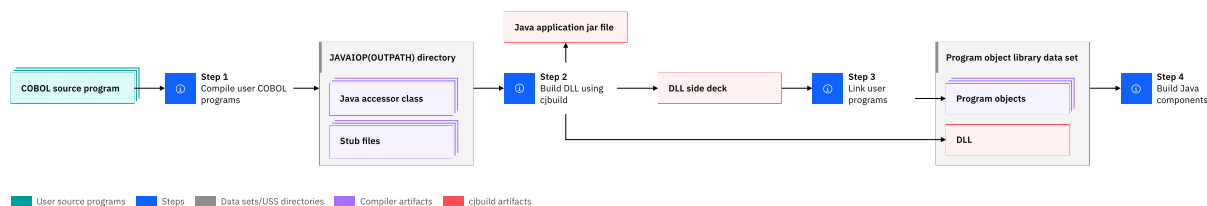
注：cjbuild ユーティリティは、COBOL ネイティブ・メソッドの呼び出し時または COBOL WORKING-STORAGE メモリへのアクセス時に Java アプリケーションが必要とするクラス用に、複数の Java ソース・ファイルも生成します。cjbuild ユーティリティは、これらのファイル構築を処理します。

3. ユーザー COBOL プログラムをリンクします。そうでないプログラム OO でない COBOL/Java 相互運用性機能を使用する場合、ステップ 3 でリンクする必要があるプログラムへの依存関係がなければ、普通

にリンクすることができ、必要であれば、ステップ 1 の間にリンクすることができる。しかし、OO 以外の COBOL/Java 相互運用機能を使用するプログラムは、ステップ 2 で構築された DLL 用に生成されたサイドデッキでリンクされなければならない。

4. アプリケーション・ユーザー Java の一部をコンパイルします。

次の図は、Java 相互運用可能 COBOL アプリケーションの構築の 4 つのステージを示しています。



関連タスク

CALL ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

JAVA-CALLABLE (*Enterprise COBOL for z/OS 言語解説書*)

JAVA-SHAREABLE (*Enterprise COBOL for z/OS 言語解説書*)

COBPROD アプリケーションの紹介

このセクションでは、Java と相互運用する非 OO COBOL アプリケーションを構築する方法を説明する場合に後続のセクションで使用される、COBPROD という単純なアプリケーションについて説明します。

このタスクについて

COBPROD アプリケーションは、以下の z/OS UNIX ファイルで構成されます。

cobprod.cbl

JAVA-SHAREABLE ディレクティブを使用して、Java と共用する WORKING-STORAGE データ項目を識別する Java 呼び出し可能 COBOL プログラム。

addlist.cbl

3 つの `java.math.BigDecimal` オブジェクトの配列を取り、それらの合計を計算し、Java 静的メソッドを呼び出して結果を端末にダンプしてから、実際の合計を Java 呼び出し元に戻す Java 呼び出し可能 COBOL プログラム。

CobProd.java

ジャワアプリケーションを起動し、COBOL ネイティブ メソッド COBPROD を呼び出し、COBPROD の WORKING-STORAGE メモリ内の PROD-INFO テーブルから情報を取得し、COBOL ネイティブ メソッド 'addlist' を呼び出すプログラム。

COBPROD アプリケーションは、ジャワ「メイン」メソッドの側に立つ CobProd クラスは COBOL ネイティブメソッド COBPROD を呼び出し、初期化します。ジャワ WORKING-STORAGE メモリ内の製品情報の共有可能なテーブル。制御が Java に戻ると、製品情報テーブルからの価格の一部が配列に収集され、COBOL ネイティブメソッド「addlist」に渡されます。このメソッドは価格を合計し、Java に戻って合計を端末に表示します。

z/OS UNIX ファイルについての詳細は、[338 ページの『例: COBPROD アプリケーション - ビルドと実行』](#)を参照してください。

Java と相互運用する非 OO COBOL アプリケーションをコンパイル、バインド、および実行するための準備

このセクションでは、Java と相互運用する非 OO COBOL プログラムを構築して実行するための前提条件について説明します。

このタスクについて

この説明では、コンパイルとリンクは、cob2 ユーティリティを使用して z/OS UNIX から実行され、アプリケーション実行時に必要とするスタブ・プログラム DLL の構築は、cjbuild ユーティリティを使用して z/OS UNIX から実行されます。しかし、プログラム・オブジェクトと DLL の両方を MVS データセットに送信する方法の例が提供されています。、JCL からアプリケーションをビルドして実行する JCL の例も提供されています。詳しくは、355 ページの『COBOL-calls-java アプリケーションをビルドして実行するためのサンプル JCL』を参照してください。

注：cob2、cjbuild ユーティリティともに、z/OS UNIX ファイル・システムの COBOL インストール・ディレクトリの bin ディレクトリに存在する。デフォルトの場所は /usr/lpp/cobol/igvyXrY です。ここで、X はコンパイラ・バージョン、Y はコンパイラ・リリースです。このディレクトリは、UNIX PATH 環境変数に含める必要があります。

cjbuild ユーティリティは、UNIX 専用ユーティリティとして提供されていますが、BPXBATCH を使用して JCL から起動することができます。

JAVA_HOME 環境変数が、システム上の Java SDK インストール・ディレクトリに設定されていることを確認します。例えば、Java 8 を使用する場合に、ディレクトリは、AMODE 31 (31 ビット) Java の場合は /usr/lpp/java/IBM/J8.0、AMODE 64 (64 ビット) Java の場合は /usr/lpp/java/IBM/J8.0_64 になります。

注：JCL や z/OS UNIX コマンド・ラインなど、パーツをコンパイルおよびリンクするための任意の方法を選択できます。z/OS UNIX メソッドが使用されるのは、UNIX が、Java の自然環境であるためです。

ステップ 1: ユーザー COBOL プログラムのコンパイル

このトピックでは、Java と相互運用する非 OO COBOL アプリケーションで、ユーザー COBOL プログラムをコンパイルする方法について説明します。

このタスクについて

一般に、アプリケーション内のユーザー COBOL プログラムは、ユーザーが通常ユーザー COBOL プログラムをコンパイルするのと同じ方法でコンパイルできます。ただし、JAVA-CALLABLE ディレクティブまたは JAVA-SHAREABLE ディレクティブを含むプログラム、または CALL ステートメントを使用して静的 Java メソッドを呼び出すプログラムは、Java インターオペラビリティ・オプション JAVAIOp を指定することを検討する必要があります。これにより、Java 相互運用可能プログラムのさまざまな側面を制御できます。

特に、JAVAIOp オプションを使用すると、以下を実行することができます。

- コンパイラが生成したスタブファイルが書き込まれる場所を制御する z/OS ユニックスコンパイル時にファイルシステムを OUTPATH サブオプション。これらのスタブ・プログラムは、Java との相互運用性を容易にします。

注：コンパイラの実行前に z/OSJAVAIOp の OUTPATH サブオプションに指定された UNIX ディレクトリは、コンパイラを実行する前に存在していなければならない。簡単にするため、1 つの相互運用アプリケーションのすべてのスタブファイルは、z/OS UNIX 上の同じディレクトリに配置することを推奨する。これは、相互運用アプリケーションの各 COBOL プログラムをビルドするときに、JAVAIOp コンパイラ・オプションの OUTPATH サブオプションで同じ出力ディレクトリを指定する必要があることを意味する。そのディレクトリを、cjbuild ユーティリティの -c/--coboldir オプションの引数として指定する。

- 相互運用可能アプリケーションが、デフォルトで 31 ビット Java をターゲットにするか、JAVA64 サブオプションを使用して 64 ビット Java をターゲットにするかを制御します。JAVA64 サブオプションは、コンパイラ・オプション LP(32) が有効な場合にのみ使用可能であることに注意してください。LP(64) が有効な場合、64 ビットの Java が必要です。
- JVMINITOPTIONS サブオプションを使用して、Java 仮想マシン (JVM) 初期化オプションのストリングを指定します。これが必要となるのは、COBOL 側で開始するアプリケーションの場合、および Java 静的メソッドに対する CALL ステートメントを含むアプリケーションで COBOL プログラムをコンパイルする

場合のみです。この場合、COBOL が実行時に JVM を開始する必要がある可能性があります。
JVMINITOPTIONS サブオプションを指定して JAVAIOp オプションを使用することは、プログラムの実行時に COBJVMINITOPTIONS 環境変数を設定することに代わる方法です。これは、OO COBOL で必要な方法であり、OO Java 以外のインターオペラビリティにも使用できます。

COBPROD アプリケーションの場合、以下のコマンドは COBOL ファイルを構築し、生成されたスタブ・ファイルおよび Java ファイルを z/OS UNIX ディレクトリ `${HOME}/jtoc/docs` に送信するようにコンパイラーに指示します。

```
cob2 -c cobprod.cbl "-qjavaiop(OUTPATH('${HOME}/jtoc/docs'))"  
cob2 -c addlist.cbl "-qjavaiop(OUTPATH('${HOME}/jtoc/docs'))"
```

デフォルトの LP(32) オプションを使用してこれらのプログラムをコンパイルするため、コンパイラーは 31 ビットの Java をターゲットとしていると想定します。31 ビット Java の代わりに 64 ビット Java をターゲットにするには、JAVAIOp オプションの JAVA64 サブオプションを組み込むようにコンパイル・コマンドを変更するだけです。

```
cob2 -c cobprod.cbl "-qjavaiop(OUTPATH('${HOME}/jtoc/docs'),java64)"  
cob2 -c addlist.cbl "-qjavaiop(OUTPATH('${HOME}/jtoc/docs'),java64)"
```

注：このアプリケーションが Java 側で開始される場合、JVM を開始するのは Java であるため、JVMINITOPTIONS サブオプションを指定する必要はありません。

関連参照

JAVAIOp

338 ページの『例: COBPROD アプリケーション - ビルドと実行』

JAVA 呼び出し可能 (*Enterprise COBOL for z/OS 言語解説書*)

JAVA 共有可能 (*Enterprise COBOL for z/OS 言語解説書*)

ステップ 2: Java スタブ・プログラムの DLL を構築するための cjbuild ユーティリティの実行

このトピックでは、Java スタブ・プログラムの DLL を構築するために cjbuild ユーティリティを実行する方法について説明します。

このタスクについて

コンパイル時に cobprod.cbl そして addlist.cbl ファイル 333 ページの『ステップ 1: ユーザー COBOL プログラムのコンパイル』指定された 3 つのスタブファイルが生成されました z/OS ユニックス出力ディレクトリ。これらのファイルは、COBOL/によって実行時に使用される DLL に組み込む必要があります。ジャワ相互運用可能なアプリケーション。

指定された方法で生成された 3 つのファイルのファイル名は、z/OS ユニックス出力ディレクトリは次のとおりです。

- COBPROD_java_native.cbl (COBPROD の COBOL ネイティブ・メソッド・スタブ)
- COBPROD_java_native_WS.cbl (COBPROD の WORKING-STORAGE 初期化スタブ)
- addlist_java_native.cbl (addlist の COBOL ネイティブ・メソッド・スタブ)

これらのファイルを必要な DLL に構築するのを支援するために、cjbuild ユーティリティが提供されています。一般に、これらのファイルに注意する必要はありません。代わりに、cjbuild にジャワ相互運用性機能 (JAVA_CALLABLE 指令、JAVA_SHAREABLE 指令、または静的 ジャワ メソッドを呼び出す CALL 文) を含む COBOL プログラムの名前のリスト。この COBOL プログラムのリストは、cjbuild コマンドの入力として指定するファイルで提供することもできるし、cjbuild コマンド行で `-i/--progid` オプションを 1 つ以上使用することもできる。また、`-i/--progid` オプションと `<program-file>` 引数の両方を省略することもできる。これは、`-c/--cobolddir` オプションで示される COBOL 成果物ディレクトリのすべての成果物を cjbuild で処理することを示す。

注：の COBPROD アプリケーションには、リテラルではなく大文字と小文字を区別しない名前を持つプログラムが 1 つと、リテラルで大文字と小文字を区別する名前を持つプログラムが 1 つ意図的に含まれています。これはデモンストレーションを目的としています。これらのプログラムのリストを cjbuild に提供する場合、ケースを考慮してください。

スタブファイルのディレクトリから cjbuild ユーティリティを実行すると、次の cjbuild コマンドを使用して、z/OS ユニックスディレクトリ内のファイルシステム \${HOME}/java/demo:

```
cjbuild -v -i COBPROD -i addlist -c ${HOME}/jtoc/docs -j ${HOME}/java/demo \
-m PURE31 -d ${HOME}/java/demo \
-p enterprise.COBOL app1
```

この表には、この例で cjbuild に渡される引数とそれらの意味の集計がリスト化されています。

拡張	意味
-v	冗長な出力。DLL をコンパイルしてリンクし、関連する Java ソース・ファイルを構築するための事実上のコマンド (例えば、cob2、javac など) が表示されます。 詳細出力を強くお勧めします。
-c \${HOME}/jtoc/docs	コンパイラによって生成された COBOL スタブ・ファイルのリポジトリを見つける場所を cjbuild に指示します。これらのファイルは、 333 ページの『ステップ 1: ユーザー COBOL プログラムのコンパイル』 で対応する COBOL ファイルを構築するときに使用される JAVAIO オプションの OUTPATH サブオプションによって指示されたディレクトリに書き込まれました。 デフォルト: 現行ディレクトリ
-j \${HOME}/java/demo	コンパイラまたは cjbuild 自体のいずれかによって生成された Java ファイルをコンパイルするときに、cjbuild によって生成された Java .class ファイルを出力する場所を cjbuild に指示します。 デフォルト: 現行ディレクトリ
-m PURE31	LP(32) が有効な AMODE 31 COBOL プログラム、および 31 ビット Java を使用してアプリケーションを構築することを cjbuild に指示します。代わりに、MIX_31_64 と PURE64 があります。 デフォルト。PURE31

表 45. (続き)

拡張	意味
-d \${HOME}/java/demo	DLL を出力する場所を cjbuild に指示します。z/OS UNIX パスまたは MVS データ・セット名にすることができます。 DLL を USER.COBOL.LOAD という名前の MVS データ・セットにメンバー LIBAPP1 として出力するには、次の -d オプションを使用します。 <pre>-d "//COBOL.LOAD"</pre> または <pre>-d "'//USER.COBOL.LOAD'"</pre> 注: によると MVs バインダー規則では、DLL がデータセットに出力されると、対応するサイドデッキファイルが、DLL 出力データセットの名前の最後の名前部分が EXP に置き換えられた名前のデータセットに書き込まれます。この場合は USER.COBOL.EXP。ただし、cjbuild は現在のサイドデッキのコピーも作成します。z/OS ユニックスユーザービルドツールがそこからアクセスする必要がある場合に備えて、ディレクトリを作成します。 デフォルト: 現行ディレクトリー
-p enterprise.COBOL	progs クラスと strg クラスが属するパッケージ名を cjbuild に指示します。progs クラスおよび strg クラスは、Java から COBOL ネイティブ・メソッドおよび COBOL 作業用ストレージ項目へのインターフェースをそれぞれ提供します。 デフォルト: enterprise.COBOL

DLL を USER.COBOL.LOAD という MVS データ・セットに出力するには、次のコマンドを発行します。

```
cjbuild -v -c ${HOME}/jtoc/docs -j ${HOME}/java/demo \
-d "//USER.COBOL.LOAD" -m PURE31 app1
```

注: DLL が MVS データ・セットに出力される場合は、実行時にデータ・セットが STEPLIB に組み込まれていることを確認してください。組み込まれていない場合、JVM はデータ・セットを見つけることができません。DLL が z/OS UNIX ディレクトリーに出力される場合は、java.library.path プロパティを使用して、そのディレクトリーを JVM に指定する必要があります。

cjbuild を実行すると、以下の 3 つの Java ファイルも作成され、コンパイルされます。

- cobol.java
- progs.java
- strg.java

これらのファイルには、COBOL ネイティブ・メソッドおよび作業用ストレージ項目へのアクセスを可能にするインターフェースを提供する Java クラス progs および strg が含まれています。これらのクラス名は、-p オプションを使用して指定したパッケージ名によって修飾されます。

例えば、この場合、-p は enterprise.COBOL をパッケージ名として指定するため、Java プログラムで、以下のようにネイティブ・メソッドおよび WORKING-STORAGE 項目にアクセスできます。

- enterprise.COBOL.progs.COBPROD
- enterprise.COBOL.strg.COBPROD

JAVA-SHAREABLE ディレクティブを含む COBOL プログラムは、COBOL WORKING-STORAGE メモリーと直接インターフェースするために Java アプリケーションから使用できるクラスを定義する Java ファイルも生成します。これらの Java ファイルは、メソッド・ファイル内の対応する COBOL プログラム名から識別され、cjbuild によって自動的に構築されます。結果のクラス・ファイルは、cjbuild の -j オプションによって示されるディレクトリに配置されます。あるいは、代わりに -r/--jar <jar-file> オプションを指定して、生成されたすべての.class ファイルを <jar-file> という名前の.jar ファイルにビルドすることもできます。

cjbuild ユーティリティーの出力例を以下に示します。

```
cjbuild, version 2.0.0
compatible stub file version: 6.0
=====

PROGRAM_FILE      = none
LIB_BASE_NAME     = app1
COBOL_DIR         = ./out
DLL_OUT_DIR       = //COBOL.LOAD
JAVA_CLASS_DIR    = ./class
JAVA_SRC_DIR      = .
JAR_FILE          = ./libapp1.jar
CLEANUP_SCRIPT_NAME = cjbuild_clean.sh
PACKAGE_NAME      = enterprise.COBOL
INTEROP_MODE      = MIX_31_64
JAVA_HOME         = /usr/lpp/java/J11.0_64
COBOL_INSTALL_DIR = /home/user/cobol/igyv6r5

User env STEPLIB:
USERID.IGY.V6R5M0.SIGYCOMP:CEE.SCEERUN:CEE.SCEERUN2

Including file in app DLL: addlist_stub_app1.cbl
Including file in app DLL: COBPROD_stub_app1.cbl

# Compiling COBOL stub file addlist_stub_app1.cbl
/home/user/cobol/igyv6r5/bin/cob2 -qlist -qopt(1),nsymbol(national),dbcs,dispsign(sep) -c
addlist_stub_app1.cbl -I /home/user/cobol/igyv6r5/include
# Command return code 0

# Compiling COBOL stub file COBPROD_stub_app1.cbl
/home/user/cobol/igyv6r5/bin/cob2 -qlist -qopt(1),nsymbol(national),dbcs,dispsign(sep) -c
COBPROD_stub_app1.cbl -I /home/user/cobol/igyv6r5/include
# Command return code 0

# Linking application DLL
/home/user/cobol/igyv6r5/bin/cob2 -bdll,map -o //COBOL.LOAD(app1) addlist_stub_app1.o
COBPROD_stub_app1.o /usr/lpp/java/J11.0_64/lib/j9vm/libjvm31.x /home/user/cobol/igyv6r5/lib/
igzxjni2.x > libapp1.map
# Command return code 0

# Copying DLL side deck to current directory
cp -F bin //COBOL.EXP(app1) libapp1.x
# Command return code 0

# Compiling Java source program into bytecode
/usr/lpp/java/J11.0_64/bin/javac -d ./class ./cobol.java ./progs.java /home/user/
jtoc/test/cobprod/COBPRODWS.java ./strg.java
# Command return code 0

# Building the .jar file
/usr/lpp/java/J11.0_64/bin/jar cvf ./libapp1.jar -C ./class enterprise/COBOL
# Command return code 0

cjbuild completed successfully.
=====
```

関連参照

JAVAIOP

361 ページの『cjbuild コマンドリファレンス』

338 ページの『例: COBPROD アプリケーション - ビルドと実行』

JAVA 呼び出し可能 (Enterprise COBOL for z/OS 言語解説書)

JAVA 共有可能 (Enterprise COBOL for z/OS 言語解説書)

ステップ 3: COBOL プログラムのリンク

このトピックでは、Java と相互運用する非 OO COBOL アプリケーションで、ユーザー COBOL プログラムをコンパイルする方法について説明します。

このタスクについて

非 OO COBOL を使用する COBOL プログラム/ジャワ相互運用性機能は、cjbuild ユーティリティによって生成された DLL のサイドデッキにリンクされている必要があります。334 ページの『[ステップ 2: Java スタブ・プログラムの DLL を構築するための cjbuild ユーティリティの実行](#)』。

COBPROD アプリケーションの場合、このサイド・デッキは libapp1.x と呼ばれ、cjbuild ユーティリティが呼び出された z/OS UNIX ディレクトリにあります。

注: もし、-d cjbuild のオプションは、出力 DLL がデータ セットに書き込まれることを示している場合、サイドデッキも、最後の名前の部分が EXP に置き換えられることを除いて、出力データセットと同じ名前のデータセットで使用可能になります。たとえば、DLL がデータセットに書き込まれた場合 USER.COBOL.LOAD サイドデッキファイルは、データセットで利用可能になります。USER.COBOL.EXP。cjbuild は常に、cjbuild ユーティリティが実行される z/OS UNIX ディレクトリにサイドデッキのコピーがあることを保証する。

以下のコマンドを使用して、COBOL プログラム cobprod と addlist をリンクし、結果のプログラム・オブジェクトを MVS データ・セット USER.COBOL.LOAD に送信します。スタブ・ファイルのディレクトリから、コマンドを実行します。

```
cob2 cobprod.o -o "//COBOL.LOAD(cobprod)" ./libapp1.x
cob2 addlist.o -o "//COBOL.LOAD(addlist)" ./libapp1.x
```

関連参照

338 ページの『[例: COBPROD アプリケーション - ビルドと実行](#)』

ステップ 4: Java プログラムのビルド

このトピックでは、Java とのインターオペラビリティを実現 (相互運用) する非 OO COBOL アプリケーションの一部である Java ファイルのビルド方法について簡単に説明します。

このタスクについて

アプリケーション内の Java ファイルは、通常どおりにビルドする必要があります。ただし、Java クラスが COBOL ネイティブ・メソッドを呼び出すか、COBOL WORKING-STORAGE に直接アクセスすることで、このフレームワークを使用して COBOL と対話する場合は、cjbuild ユーティリティによって生成された COBOL クラスを含む Java パッケージをインポートする必要があります。これは、cjbuild の -p オプションによって指定されたパッケージ名であり、デフォルトは enterprise.COBOL です。

例えば、以下は、Java ファイルの先頭に追加されるデフォルトのインターオペラビリティ・パッケージ名のインポート・ステートメントです。

```
import enterprise.COBOL.*;
```

例: COBPROD アプリケーション - ビルドと実行

このセクションでは、COBPROD アプリケーションをビルド/実行する方法について説明します。z/OS ユニックス。アプリケーションの構築と実行方法の説明 JCL より次のセクション「COBPROD アプリケーショ

ンをビルドおよび実行するためのサンプル JCL」で提供されます。COBPROD アプリケーションは、人工的に作られた COBOL/ジャワこのセクション全体でデモンストレーション目的で使用される相互運用可能なアプリケーション。COBPROD アプリケーションのバージョンは、以下のファイルセットとして提供されています。demo/cobol_java_interop_sample COBOL インストールディレクトリのサブディレクトリ z/OS ユニックスファイルシステム。ファイルのバージョンはコンパイラの SIGYSAMP データセット (名前が接頭辞 igycjdn で始まるメンバーを参照)。

このタスクについて

COBPROD アプリケーションは、COBPROD と「addlist」という 2 つの COBOL プログラムと、ジャワクラスと呼ばれる CobProd。アプリケーションはジャワ側では、COBPROD COBOL プログラムを呼び出して WORKING-STORAGE 内の製品情報のテーブルを初期化し、その情報に直接アクセスします。ジャワ帰国後。のジャワ次に、価格リストを「addlist」COBOL プログラムに渡し、リストを合計して戻るジャワ合計が端末に表示される。

注: 大きな配列を渡す `java.math.BigDecimal` オブジェクトまたは `java.lang.String` オブジェクトから COBOL ヘジャワデータ型の変換によりコストが高くなる可能性があるため、パフォーマンスが重要な場合には適切ではない可能性があります。小さな配列を渡す `java.math.BigDecimal` ここでは、オブジェクトを慎重に使用した場合やパフォーマンスがそれほど重要でない場合に、このような操作が便利であることを示すために説明します。

アプリケーションでは、次の 2 つの相互運用性の実証されています。

- ジャワ COBOL の呼び出し
- ジャワ COBOL WORKING-STORAGE に直接アクセスする

デモには次のユーザー ソース ファイルが含まれています。

cobprod.cbl

WORKING-STORAGE を、アクセス可能な製品情報のテーブルで初期化する COBOL プログラム。ジャワ。

addlist.cbl

価格リストを加算し、その結果をターミナルに出力する COBOL プログラム。ジャワ。

CobProd.java

Cobprod を定義するジャワクラスはジャワベースのデモ ドライバー。

COBPROD アプリケーションをビルドおよび実行するための次のスクリプト ファイルも提供されています。

build.sh

COBPROD アプリケーションを構築するためのシェルスクリプト。スクリプトを使用してビルドするための手順は、スクリプト自体に含まれています。

呼び出す `build.sh` で z/OS ユニックスコマンドライン。混合 AMODE アプリケーション (AMODE 31 COBOL と AMODE 64) ジャワ) が建設されます。

Make ファイル

代替として `build.sh` スクリプトでは、`makefile` を好むユーザー向けに、`makefile` を使用して相互運用可能なアプリケーションを構築する方法を示す `makefile` が提供されています。メイクファイルのビルドでは、混合 AMODE アプリケーション (AMODE 31 COBOL、AMODE 64 ジャワ) が常に構築されます。

開始する次のコマンドを実行して `makefile` ベースのビルドを実行します。z/OS ユニックス `makefile` を含むディレクトリ内のコマンドライン:

```
make
```

run.sh

相互運用可能なアプリケーションを実行するためのシェルスクリプト。いずれかを使用して構築されたアプリケーションで動作します `build.sh` またはメイクファイル。実行方法の詳細についてはスクリプトを参照してください。

その他のさまざまなファイルが提供されています:

README

COBPROD アプリケーションの構築と実行に関する詳細情報が含まれています。

cobprod.cbl

```
*****
*
* IBM Enterprise COBOL for z/OS
* Version 6 Release 4 Modification 0
*
* LICENSED MATERIALS - PROPERTY OF IBM.
*
* 5655-EC6 COPYRIGHT IBM CORP. 2022, 2024
*
* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
* ADP SCHEDULE CONTRACT WITH IBM CORP.
*
*****
* cobprod initializes a working-storage item with some
* product information to be shared with Java
* identification division.
*   program-id. cobprod.
*   data division.
*   working-storage section.
*   Product info table, shared with Java
*   NOTE: This is for demonstration purposes only.
*   Data should NOT be placed in your COBOL program code.
*   01 prod-list-size pic s9(9) comp-5 value 5.
*   01 prod-info-data.
*     03 filler.
*       05 filler pic x(20) value 'chair'.
*       05 filler pic s9(9)v9(2) value 29.99.
*     03 filler.
*       05 filler pic x(20) value 'table'.
*       05 filler pic s9(9)v9(2) value 45.98.
*     03 filler.
*       05 filler pic x(20) value 'bed'.
*       05 filler pic s9(9)v9(2) value 149.45.
*     03 filler.
*       05 filler pic x(20) value 'blanket'.
*       05 filler pic s9(9)v9(2) value 19.99.
*     03 filler.
*       05 filler pic x(20) value 'sofa'.
*       05 filler pic s9(9)v9(2) value 239.99.
*   >>java-shareable on
*   01 prod-info redefines prod-info-data.
*     03 prod-list occurs 5 times.
*       05 prod-name pic x(20).
*       05 prod-price pic s9(9)v9(2).
*   >>java-shareable off
*   linkage section.
*   01 prod-list-size-out pic s9(9) comp-5.
*   >>java-callable
*   procedure division returning prod-list-size-out.
*   MainProgram.
*   * Populate table
*     move prod-list-size to prod-list-size-out
*     goback.
*   end program cobprod.
```

addlist.cbl

```
      cbl pgmname(longmixed)
*****
*
* IBM Enterprise COBOL for z/OS
* Version 6 Release 4 Modification 0
*
* LICENSED MATERIALS - PROPERTY OF IBM.
*
* 5655-EC6 COPYRIGHT IBM CORP. 2022, 2024
*
* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
```

```

*  DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
*  ADP SCHEDULE CONTRACT WITH IBM CORP.
*
*****
*  Receives a list of decimal items and calculates
*  and returns the sum.  Maxium list size 3
identification division.
program-id. 'addlist'.
data division.
working-storage section.
local-storage section.
01 idx pic s9(9) comp-5.
linkage section.
01 pricelist.
   03 price pic s9(9)v9(2) comp-3 occurs 3 times.
01 listsize pic s9(9) comp-5.
01 result pic s9(9)v9(2) comp-3.
>>java-callable
procedure division using by reference pricelist
                        by value listsize
                        returning result.

MainProgram.
*  Compute sum of list
  display '<<COBOL: addlist: entered>>'

  compute result = 0

  perform varying idx from 1 by 1
    until idx > 3
      compute result = result + price(idx)
    end-perform

  display '<<COBOL: addlist: exited>>'
  goback.
end program 'addlist'.

```

CobProd.java

```

//*****
//
//  IBM Enterprise COBOL for z/OS
//  Version 6 Release 4 Modification 0
//
//  LICENSED MATERIALS - PROPERTY OF IBM.
//
//  5655-EC6 COPYRIGHT IBM CORP. 2022, 2024
//
//  US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
//  DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
//  ADP SCHEDULE CONTRACT WITH IBM CORP.
//
//*****
import enterprise.COBOL.*;
import java.math.*;

public class CobProd
{
    public static void main(String[] args)
    {
        System.out.println("<<Java: CobProd.main: entered>>");

        System.out.println();
        System.out.println("Java: Access COBOL product info from COBPROD");

        // Call out to COBOL to initialize the product table in COBPROD
        int tableSize = enterprise.COBOL.progs.COBPROD();
        System.out.println("Java: prod table size = " + tableSize);

        // Display the product names and prices in the table
        for (int i = 0; i < tableSize; i++)
        {
            System.out.print("Java: Product name = " +
                enterprise.COBOL.strg.COBPROD.PROD_INFO.PROD_LIST[i].PROD_NAME.get());
            System.out.print(" Price = $");

            System.out.println(enterprise.COBOL.strg.COBPROD.PROD_INFO.PROD_LIST[i].PROD_PRICE.get());
        }
        System.out.println();

        // Calculate the sum of the first 3 prices in the table
    }
}

```

```

        BigDecimal[] prices = new BigDecimal[3];
        assert(tableSize >= 3);
        assert(prices.length >= 3);
        prices[0] = enterprise.COBOL.strg.COBPROD.PROD_INFO.PROD_LIST[0].PROD_PRICE.get();
        prices[1] = enterprise.COBOL.strg.COBPROD.PROD_INFO.PROD_LIST[1].PROD_PRICE.get();
        prices[2] = enterprise.COBOL.strg.COBPROD.PROD_INFO.PROD_LIST[2].PROD_PRICE.get();

        // Call out to COBOL routine addlist to do the actual calculation
        System.out.println("Java: call COBOL addlist to add prices");

        BigDecimal result = enterprise.COBOL.progs.addlist(prices, prices.length);

        System.out.println();
        System.out.println("Java: Price of chair+table+sofa is: $" + result);

        System.out.println();
        System.out.println("<<Java: CobProd.main: exited>>");
    }
}

```

build.sh

```

#!/bin/sh
#####
#
# IBM Enterprise COBOL for z/OS
#          Version 6 Release 4 Modification 0
#
# LICENSED MATERIALS - PROPERTY OF IBM.
#
# 5655-EC6 COPYRIGHT IBM CORP. 2022, 2024
#
# US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
# DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
# ADP SCHEDULE CONTRACT WITH IBM CORP.
#
#####
# Build script for the COBOL/Java interoperability demo COBPROD
#####
#
# Invocation:
# -----
#
# build.sh          - build mixed amode interoperable application
#
# NOTE:
# ----
#
# 1) Make sure cjbuild and cob2 are in your PATH before running this script
#
# 2) Make sure JAVA_HOME is set appropriately
#
# 3) Make sure your STEPLIB includes an Enterprise COBOL V6.4 or later
#    compiler and a runtime that is sufficient to run COBOL programs
#    compiled with the Enterprise COBOL V6.4 or later compiler
#
# 4) You must have an existing data set to receive COBOL program objects
#    and DLLs and it must be called <USER>.COBOL.LOAD, where <USER> is
#    your high-level qualifier
#
#####

if [ -z $STEPLIB ]; then
    echo "STEPLIB environment variable must be set"
    exit 1
fi

if [ -z $JAVA_HOME ]; then
    echo "JAVA_HOME environment variable must be set"
    exit 1;
fi

OUT_DIR=./out
CLASSOUT_DIR=./class

# Set the AMODE variables
JAVAMODE="JAVA64"
CJMODE="MIX_31_64"

```

```

# Make the output directory
rm -fr ${OUT_DIR} ${CLASSOUT_DIR}
mkdir ${OUT_DIR} ${CLASSOUT_DIR}

# STEP 1: Compile user COBOL programs, specifying JAVAIOp option
cob2 -c cobprod.cbl "-qjavaio(OUTPATH('${OUT_DIR}'),${JAVAMODE})"
cob2 -c addlist.cbl "-qjavaio(OUTPATH('${OUT_DIR}'),${JAVAMODE})"

# STEP 2: Build application DLL and .jar file using cjbuild
cjbuild -v \
  --jar libapp1.jar \
  --javaclassdir ${CLASSOUT_DIR} \
  --mode ${CJMODE} \
  --pkgname enterprise.COBOL \
  --coboldir ${OUT_DIR} \
  --dlloutdir "//COBOL.LOAD" \
  app1

# STEP 3: Link user programs
cob2 cobprod.o -o "//COBOL.LOAD(cobprod)" libapp1.x
cob2 addlist.o -o "//COBOL.LOAD(addlist)" libapp1.x

# STEP 4: Build Java parts of the application
export CLASSPATH=${CLASSOUT_DIR}:${CLASSPATH}

${JAVA_HOME}/bin/javac -d ${CLASSOUT_DIR} CobProd.java

```

makefile

```

#####
# Makefile for the COBOL/Java interoperability demo COBPROD (Mixed 31/64 mode)
#####
#
# NOTE:
# ----
#
# 1) Make sure cjbuild and cob2 are in your PATH before using this makefile
#
# 2) Make sure your STEPLIB is set to include an Enterprise V6.4 or later
#    compiler and runtime
#
# 3) Make sure JAVA_HOME points to your 64-bit Java JDK
#
#    e.g., JAVA_HOME=/usr/lpp/java/IBM/J8.0_64 or
#           JAVA_HOME=/usr/lpp/java/IBM/J11.0_64
#
# 4) You must have an existing data set to receive COBOL program objects
#    and DLLs and it must be called <USER>.COBOL.LOAD, where <USER> is
#    your high-level qualifier
#
#####

MYSTEPLIB=$(USER).COBOL.LOAD:$(STEPLIB)
OUT_DIR=./out
CLASSOUT_DIR=./class
MYCLASSPATH=$(CLASSOUT_DIR):$(CLASSPATH)

all: setup libapp1 cobprod addlist CobProd.class

setup:
    @rm -fr out class
    @mkdir -m 0755 out class

libapp1: cobprod.o addlist.o methods
    STEPLIB=$(MYSTEPLIB) cjbuild -v -j $(CLASSOUT_DIR) -p enterprise.COBOL -m MIX_31_64 -c
    $(OUT_DIR) -d "//COBOL.LOAD" methods app1

cobprod: cobprod.o
    STEPLIB=$(MYSTEPLIB) cob2 cobprod.o -o "//COBOL.LOAD(cobprod)" libapp1.x

addlist: addlist.o libapp1
    STEPLIB=$(MYSTEPLIB) cob2 addlist.o -o "//COBOL.LOAD(addlist)" libapp1.x

cobprod.o: cobprod.cbl
    STEPLIB=$(MYSTEPLIB) cob2 -c cobprod.cbl "-qjavaio(OUTPATH('${OUT_DIR}'),JAVA64)"

addlist.o: addlist.cbl
    STEPLIB=$(MYSTEPLIB) cob2 -c addlist.cbl "-qjavaio(OUTPATH('${OUT_DIR}'),JAVA64)"

CobProd.class: CobProd.java
    CLASSPATH=$(MYCLASSPATH) ${JAVA_HOME}/bin/javac -d $(CLASSOUT_DIR) CobProd.java

```

```

clean:
    rm -fr *java_native* *WS.java *WS_base.java *.o *.class *.sig CEEDUMP.* Java.* *.x
    *.so Snap.* javacore.* jittedump.* *.lst
    rm -fr cobol.java progs.java strg.java
    rm -fr LIBAPP1* libapp1*
    rm -fr out
    rm -fr class

```

run.sh

```

#!/bin/sh
#####
#
# IBM Enterprise COBOL for z/OS
#          Version 6 Release 4 Modification 0
#
# LICENSED MATERIALS - PROPERTY OF IBM.
#
# 5655-EC6 COPYRIGHT IBM CORP. 2022, 2024
#
# US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
# DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
# ADP SCHEDULE CONTRACT WITH IBM CORP.
#
#####
# Run script for the COBOL/Java interoperability demo COBPROD
#####
#
# NOTE:
# ----
#
# 1) Make sure your STEPLIB includes an Enterprise COBOL runtime
#     that is sufficient to run programs compiled with the Enterprise
#     COBOL V6.4 or later compiler.
#
#     The dataset <USER>.COBOL.LOAD that received the program objects
#     and DLL for your interoperable application during the build
#     step must also be included in your STEPLIB. We have added this
#     in automatically for you below.
#
# 2) Set JAVA_HOME to an appropriate 64-bit Java SDK:
#
#     e.g., JAVA_HOME=/usr/lpp/java/IBM/J8.0_64 or
#           JAVA_HOME=/usr/lpp/java/IBM/J11.0_64
#
# 3) If you modify this example so that cjbuild stores the DLL it
#     produces to a z/OS UNIX directory, then you need to modify
#     the last line of this script to the following:
#
#     ${JAVA_HOME}/bin/java -Djava.library.path=<dll_out_dir> CobProd
#
#####

if [ -z $STEPLIB ]; then
    echo "STEPLIB environment variable must be set"
    exit 1
fi

if [ -z $JAVA_HOME ]; then
    echo "JAVA_HOME environment variable must be set"
    exit 1
fi

export STEPLIB=${USER}.COBOL.LOAD:${STEPLIB}
export CLASSPATH=./libapp1.jar:./class:${CLASSPATH}
export IBM_JAVA_OPTIONS="-XX:+Enable3164Interoperability"
export _CEE_RUNOPTS="POSIX(ON)"
export LIBPATH=./:${JAVA_HOME}/lib/s390x/j9vm:${JAVA_HOME}/lib/s390x:${JAVA_HOME}/bin/j9vm:${
    JAVA_HOME}/lib/j9vm:${LIBPATH}

${JAVA_HOME}/bin/java CobProd

```

関連参照

[331 ページの『第 18 章 Java と相互運用する非 OO COBOL アプリケーションのコンパイル、リンク、およ](#)

COBPROD アプリケーションをビルドして実行するためのサンプル JCL

このセクションには、COBPROD アプリケーションをビルドおよび実行するためのサンプル JCL が含まれています。前のセクションでは、COBPROD デモ・アプリケーションと、z/OS UNIX シェル・スクリプトと makefile を使ったビルドと実行方法について説明した。しかし、一部のユーザーは、JCL を使用して非 OO COBOL/Java 相互運用アプリケーションを構築し、実行することを好むかもしれない。提供された JCL は、COBPROD デモ・アプリケーション専用であるが、OO 以外の COBOL/Java 相互運用可能なアプリケーションにも容易に適応できるはずである。

このタスクについて

JCL の多くは、COBOL アプリケーションを構築およびリンクするための一般的な JCL です。主な違いは、ビルドプロセスに、cjbuild ツールを使用して相互運用可能なアプリケーションの DLL をビルドする手順が含まれていることです。cjbuild は z/OS UNIX のネイティブ・プログラムですが、サンプルでは BPXBATCH ユーティリティを使って JCL から簡単に実行できる方法を示しています。

サンプル JCL を使用して COBPROD アプリケーションをビルドおよび実行する前に考慮すべき事項。

1. JCL 自体には、ご使用の環境で JCL を正常に実行するために JCL に加える必要のある変更を説明する包括的なコメントが冒頭に含まれています。
2. JCL の指示に注意深く従い、次の重要な条件を満たしていることを確認してください。
 - JCL の z/OS UNIX パスの userid のインスタンスはすべて、JCL が実行されるアカウントのユーザー ID に置き換えられます（z/OS UNIX のユーザー ID は通常、z/OS のユーザー ID の小文字バージョンです）。
 - JCL ファイルの先頭で定義されているさまざまな JCL パス変数 (WORKDIR、ZFSCBDIR、OUTDIR など) は、環境に合わせて適切に設定されます。
 - ビルド・ジョブに使用できるシステム上の z/OS UNIX 作業ディレクトリの場所が特定され、cjbuild によって生成されるさまざまなリストやその他の成果物ファイルが作成される。作業ディレクトリとそのすべてのサブディレクトリは、サンプル JCL によって自動的に作成されます。最初に自分で作成する必要はありません。WORKDIR JCL 変数を使用して作業ディレクトリの場所を定義するだけで済みます。
 - の LNGPRFX そして LIBPRFXJCL 変数は、コンパイラとランタイム レベルをそれぞれ示すように適切に設定されます。
3. JCL を初めて実行する前に、COBPROD アプリケーション内の 2 つの COBOL ユーザーソースファイルが、名前の付いたデータセットにコピーされていることを確認する必要があります。
USERID.COBO.L.COBO ここで、USERID は JCL が実行されるアカウントのユーザー ID です。異なる名前のデータセットが必要な場合は、それに応じて JCL を変更する必要があります。2 つのユーザーソース ファイルは次のとおりです。

- cobprod.cbl
- addlist.cbl

これらは COBPROD アプリケーションのデモ ディレクトリに存在します。

```
demo/cobol_java_interop_sample
```

これは、z/OS UNIX ファイル・システムの COBOL インストール・ディレクトリにあり、通常 /usr/lpp/cobol/igyv6r4 のようなパスを持っている。これらのファイルをデータセットにコピーするには、

cobol_java_interop_sample ディレクトリからコマンドを実行すると仮定して、以下の z/OS UNIX コマンドを使用する：

```
cp cobprod.cbl "'/USERID.COBOL.COBOL(COBPROD)'"
cp addlist.cbl "'/USERID.COBOL.COBOL(ADDLIST)'"
```

注：実稼働ビルド環境では、ユーザーの COBOL ソース ファイルが既にデータセット内に配置されている可能性が高くなります。この余分なコピーは、z/OS UNIX ファイル・システムの COBOL インストール・ディレクトリに存在するデモ・ファイルをビルドするためにのみ必要である。

4. 提供されている JCL は、JCL から z/OS UNIX コマンドを実行するために BPXBATCH ユーティリティを使用しています。BPXBATCH によって z/OS UNIX シェルが起動され、z/OS UNIX ホームディレクトリにある .profile ファイルが「ソース」される（つまり、環境にロードされて実行される）。いくつかの z/OS UNIX 環境では、コマンドを使用して .profile ファイルの最後に他のデフォルトでない UNIX シェルを起動するのが一般的である：

```
exec <alternative-shell-program>
```

注：.profile からデフォルト以外のシェルが実行されない場合は、この点の残りの部分は無視しても問題ありません。

非デフォルト・シェルとは、システムに設定されているデフォルト・シェルではないシェル・プログラムのことで、通常は sh UNIX コマンドである。これにより、BPXBATCH で問題が発生する可能性があります。このような問題を防ぐ確実な方法は、対話型シェルセッションを実行している場合にのみカスタム シェル プログラムが実行されるようにすることです。これは、.profile ファイルの exec コマンドを次のようにガードすることで実現できる：

```
if [[ $- == *i* ]]; then
    exec <alternative-shell-program>
fi
```

この方法が何らかの理由で機能しない場合は、サンプル JCL はシェル変数を設定します。NO_CUSTOM_SHELL ユーザーはこれを .profile で使用して、カスタム シェルの実行を防ぐことができます。シェルコードは次のようになります。

```
if [[ -z $NO_CUSTOM_SHELL ]]; then
    exec <alternative-shell-program>
fi
```

5. JCL を実行するユーザーの z/OS UNIX ホーム・ディレクトリの .profile ファイルに JAVA_HOME 環境変数を設定することが重要です。サンプル JCL 内の 31 ビット COBPROD アプリケーションが正常に実行されるように .profile ファイルに追加する行は次のとおりです。

```
export JAVA_HOME=/usr/lpp/java/IBM/J8.0
```

注：これを行わないと、JCL の STEP 2 で Java プログラムである cjbuild ユーティリティの起動に失敗する可能性があります。

アプリケーションの実行が成功した場合の出力

アプリケーションが JCL から正常に実行されると、ジョブ ログに次の出力が含まれます。

```
<<Java: CobProd.main: entered>>
Java: Access COBOL product info from COBPROD
Java: prod table size = 5
Java: Product name = chair           Price = $29.99
Java: Product name = table          Price = $45.98
Java: Product name = bed             Price = $149.45
Java: Product name = blanket         Price = $19.99
Java: Product name = sofa            Price = $239.99
Java: call COBOL addlist to add prices
```

```
<<COBOL: addlist: entered>>
Java (called from COBOL): printResult: total price is: $225.42
<<COBOL: addlist: exited>>

<<Java: CobProd.main: exited>>
```

COBPROD JCL サンプル ファイル

このサンプル JCL は、Enterprise COBOL コンパイラーの SIGYSAMP データ セットのメンバー IGYCJDMJ で使用できます。また、z/OS UNIX ファイル・システムの COBOL インストール・ディレクトリのサブディレクトリ demo/cobol_java_interop_sample の z/OS UNIX ファイル cobprod.jcl として利用できる。

```
//USERID2 JOB 'JAVA/COBOL interop',
// MSGLEVEL=(1,1),MSGCLASS=S,CLASS=C,NOTIFY=&SYSUID,
// REGION=0M
//*****
//*          IBM Enterprise COBOL for z/OS          */
//*****
//*****
//*          LICENSED MATERIALS -- PROPERTY OF IBM.          *
//*          5655-EC6          *
//*          COPYRIGHT IBM CORP. 2023, 2025          *
//*          US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR          *
//*          DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM          *
//*          CORP.          *
//*          *****          *
//*****
//* Sample JCL for building and running the COBPROD application
//* demonstrating non-00 COBOL/Java interoperability.
//*
//*****
//* Prerequisites:
//*
//* 1) COBPROD demo source files cobprod.cbl and addlist.cbl from
//*    the ZFS COBOL install directory
//*
//*    demo/cobol_java_interop_sample
//*
//*    must be copied to a PDS or PDSE data set with name
//*    userid.COBOL.COBOL.
//*
//*    Alternatively, the JCL could be modified to read the
//*    source files directly from the zFS install directory
//*    in steps COBOL1 and COBOL2 below.
//*
//* 2) A PDSE data set named userid.COBOL.OBJECT must exist
//*    to receive object program files generated by the COBOL
//*    compiler.
//*
//* 3) A PDSE data set userid.COBOL.LOAD must exist to receive
//*    program object files. These program objects come from
//*    linking object files from user programs and also from
//*    linking the object files associated with stub programs
//*    into a DLL.
//*
//*    A PDS data set (FB 80) called userid.COBOL.EXP must also
//*    exist prior to running this JCL. This data set will
//*    receive the side deck for the DLL produced by cjbuild
//*    and is used during subsequent link steps.
//*
//* 4) You must ensure the following JCL variables are set below,
//*    which control various build parameters. This includes the
//*    z/OS UNIX directory into which stub files will be generated
//*    by the compiler and the z/OS UNIX work directory to be
//*    used by the cjbuild utility. The cjbuild utility uses this
//*    directory to build the DLL needed for the interoperable
//*    application.
//*
//*    For this sample, we assume the z/OS UNIX directory
//*    structure to be used for building the application is as
```

```

//* follows:
//*
//* /home/userid/jtoc/cobprod
//* - main work directory
//* /home/userid/jtoc/cobprod/out
//* - stub file output directory (compiler places stub
//* files here)
//* /home/userid/jtoc/cobprod/class
//* - directory where cjbuild places any generated .class
//* files
//*
//* The following variables must be set:
//*
//* JAVAHOME
//* ZFSCBDIR
//* OUTDIR
//* INTMODE
//* WORKDIR
//* CLASSOUT
//* JVSRCOUT
//* PKGNAME
//* DLLNAME
//*
//* See below for detailed descriptions of how to set these
//* variables. Example values are provided.
//*
//* NOTE: z/OS UNIX directories for stub files and the work
//* directory for cjbuild do not have to exist. The
//* JCL below create the directories if they do not
//* exist prior to running this job (see STEP 0).
//*
//*****
// EXPORT SYMLIST=*
//*****
//* Set compiler and run time library levels below
//*****
// SET LIBPRFX='CEE'
// SET LNGPRFX='COBOL.COMPILER'
//*****
//* JAVAHOME: Set this to your Java home directory. This will be used
//* to set the JAVA_HOME environment variable needed by
//* Java operations
//*****
// SET JAVAHOME='/usr/lpp/java/J11.0_64'
//*****
//* ZFSCBDIR: This should point to the location of your COBOL zFS
//* install directory (e.g., /usr/lpp/cobol/igyv6r4)
//*
//* The bin directory of this directory is where the cob2
//* and cjbuild utilities reside. They will be needed
//* when building the interoperable application
//*****
// SET ZFSCBDIR='/home/userid/cobol/igyv6r4'
//*****
//* OUTDIR: Set this to the z/OS UNIX directory that will receive
//* the stub files and other artifact files generated by
//* the COBOL compiler compiling programs that contain
//* the JAVA-CALLABLE and JAVA-SHAREABLE directives or
//* programs that make calls to Java using a CALL statement
//* of the form:
//*
//* CALL 'Java.class-name.static-method-name' ...
//*
//* The cjbuild utility (see STEP 2) will get the files from
//* this directory (see --coboldir option) so the directory
//* specified for OUTDIR should be used both in the OUTPATH
//* suboption of JAVAIOF when compiling the user COBOL
//* program as well as the --coboldir option of cjbuild.
//*****
// SET OUTDIR='/home/userid/jtoc/cobprod/out'
//*****
//* INTMODE: Set this to the interoperability mode to be used
//* by your application (needed by the cjbuild utility)
//*
//* PURE31 - AMODE 31 COBOL, AMODE 31 Java
//* PURE64 - AMODE 64 COBOL, AMODE 64 Java
//* MIX_31_64 - AMODE 31 COBOL, AMODE 64 Java
//*
//* NOTE: if you specify MIX_31_64, you must add the
//* JAVA64 suboption to the JAVAIOF option of
//* all related user COBOL programs.
//*****

```

```

//      SET INTMODE='MIX_31_64'
//*****
//* WORKDIR: Set this to the z/OS UNIX directory to be used as a work
//*           directory for building the interoperable application.
//*           cjbuild will be run from this directory and basic
//*           artifact files it generates (object files, listing,
//*           etc.) go to this directory automatically.
//*****
//      SET WORKDIR='/home/userid/jtoc/cobprod'
//*****
//* CLASSOUT: Set this to the directory you want any .class files
//*            generated by cjbuild to go. These are the .class
//*            files for Java accessor classes for JAVA-SHAREABLE
//*            items, and also .class files for any Java files
//*            generated by cjbuild, such as cobol.java, progs.java,
//*            and strg.java.
//*****
//      SET CLASSOUT='/home/userid/jtoc/cobprod/class'
//*****
//* JVSRCOUT: Set this to the directory you want any .java files
//*            generated by cjbuild to go (cobol.java, progs.java,
//*            strg.java).
//*****
//      SET JVSRCOUT='/home/userid/jtoc/cobprod'
//*****
//* PKGNAME: Set this to the name of the Java package that you
//*           want the 'progs' and 'strg' class to be a part of.
//*           The default is enterprise.COBOL. This affects how
//*           COBOL programs are invoked from Java. e.g., if the
//*           package name is enterprise.COBOL., then you would
//*           invoke a COBOL program called PROG1 as follows:
//*
//*           enterprise.COBOL.progs.PROG1();
//*****
//      SET PKGNAME='enterprise.COBOL'
//*****
//* DLLNAME: The name of the DLL that cjbuild creates and which
//*           is needed at runtime when interoperable activities
//*           occur in the application.
//*
//*           If the DLL is targeted for a data set (which is
//*           the usual way, see --dlloutdir option of cjbuild)
//*           then the DLL will be stored as a member of the
//*           specified data set, and is called <DLLNAME>.
//*
//*           If the DLL is targeted to the z/OS UNIX file system,
//*           it will be called <dllname>.so.
//*****
//      SET DLLNAME=APP1
//*****
//PROCLIB JCLLIB ORDER=(&LNGPRFX..SIGYPROC)
//*****
//***** STEP 0: Ensure required z/OS UNIX directories exist
//*****
//CRDIRS EXEC PGM=BPXBATCH
//STDPARM DD *,SYMBOLS=EXECSYS
SH mkdir -p &OUTDIR;
mkdir -p &WORKDIR;
mkdir -p &CLASSOUT;
mkdir -p &JVSRCOUT
//*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *
NO_CUSTOM_SHELL=1
//*
//*****
//***** STEP 1: Compile user COBOL programs, specifying JAVAIO option
//*****
//      SET INFILE='COBPROD'
//*****
// IF (CRDIRS.RUN AND (CRDIRS.RC EQ 0)) THEN
//COBOL1 EXEC IGYWC,
//      LNGPRFX=&LNGPRFX,
//      LIBPRFX=&LIBPRFX,
//      PARM.COBOLE='OPTFILE'
//SYSLIB DD DSN=&SYSUID..COBOL.COBOLE,DISP=SHR
//SYSIN DD DSN=&SYSUID..COBOL.COBOLE(&INFILE),DISP=SHR

```

```

//SYSLIN DD DSN=&SYSUID..COBOL.OBJECT(&INFILE),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOPTF DD *,SYMBOLS=EXECSYS
  JAVAIO(OUTPATH(' &OUTDIR'),JAVA64)
/*
//SYSOUT DD SYSOUT=*
// ENDIF
//*****
// SET INFILE='ADDLIST'
//*****
// IF (COBOL1.COBOL.RUN AND (COBOL1.COBOL.RC EQ 0)) THEN
//COBOL2 EXEC IGYWC,
// LNGPRFX=&LNGPRFX,
// LIBPRFX=&LIBPRFX,
// PARM.COBOLE='OPTFILE'
//SYSLIB DD DSN=&SYSUID..COBOL.COBOLE,DISP=SHR
//SYSIN DD DSN=&SYSUID..COBOL.COBOLE(&INFILE),DISP=SHR
//SYSLIN DD DSN=&SYSUID..COBOL.OBJECT(&INFILE),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOPTF DD *,SYMBOLS=EXECSYS
  JAVAIO(OUTPATH(' &OUTDIR'),JAVA64)
/*
//SYSOUT DD SYSOUT=*
// ENDIF
//*****
/* STEP 2: Build stub file DLL using cjbuild
//*****
// IF (COBOL2.COBOL.RUN AND (COBOL2.COBOL.RC EQ 0)) THEN
//CJBUILD EXEC PGM=BPXBATCH
//STDPARM DD *,SYMBOLS=EXECSYS
SH export JAVA_HOME=&JAVAHOME;
PATH=./bin:usr/bin:usr/local/bin:&ZFSCBDIR/bin;
cd &WORKDIR;

&ZFSCBDIR/bin/cjbuild
-v
--cobolldir &OUTDIR
--dlloutdir "///'&SYSUID..COBOL.LOAD'"
--mode &INTMODE
--pkgname &PKGNAME
--javaclssdir &CLASSOUT
--javasrcdir &JVSRCOUT
&DLLNAME
/*
//STEPLIB DD DSN=&LNGPRFX..SIGYCOMP,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *,SYMBOLS=EXECSYS
NO_CUSTOM_SHELL=1
_C89_LSYSLIB=&LIBPRFX..SCEELKD
/*
// ENDIF
//*****
//*****
/* STEP 3: Link user programs. User programs may call out to
/* programs in the DLL produced by cjbuild in STEP 2
/* thus should be linked with the side deck produced in
/* that step
//*****
//*****
// SET INFILE='COBPROD'
//*****
// IF (CJBUILD.RUN AND (CJBUILD.RC EQ 0)) THEN
//LKED1 EXEC PGM=IEWL,
// PARM='RENT,LIST,LET,DYNAM(DLL),CASE(MIXED)'
//SYSLIB DD DSN=&LIBPRFX..SCEELKD,DISP=SHR
// DD DSN=&LIBPRFX..SCEELKEX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSLMOD DD DSN=&SYSUID..COBOL.LOAD(&INFILE),DISP=SHR
//SYSDEFSD DD DUMMY
//OBJMOD DD DSN=&SYSUID..COBOL.OBJECT(&INFILE),DISP=SHR
//SIDEDK DD DSN=&SYSUID..COBOL.EXP(&DLLNAME),DISP=SHR
//SYSLIN DD *
  INCLUDE OBJMOD
  INCLUDE SIDEDK
/*
// ENDIF
//*****
// SET INFILE='ADDLIST'

```

```

//*****
// IF (LKED1.RUN AND (LKED1.RC EQ 0)) THEN
//LKED2 EXEC PGM=IEWL,
// PARM='RENT,LIST,LET,DYNAM(DLL),CASE(MIXED)'
//SYSLIB DD DSN=&LIBPRFX..SCEELKED,DISP=SHR
//      DD DSN=&LIBPRFX..SCEELKEX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSLMOD DD DSN=&SYSUID..COBOL.LOAD(&INFILE),DISP=SHR
//SYSDEFSD DD DUMMY
//OBJMOD DD DSN=&SYSUID..COBOL.OBJECT(&INFILE),DISP=SHR
//SIDEDK DD DSN=&SYSUID..COBOL.EXP(&DLLNAME),DISP=SHR
//SYSLIN DD *
//      INCLUDE OBJMOD
//      INCLUDE SIDEDK
//*
// ENDIF
//*****
//*****
//* STEP 4: Builds Java application
//*****
//*****
// IF (LKED2.RUN AND (LKED2.RC EQ 0)) THEN
//BLDJAVA EXEC PGM=BPXBATSL
//STDPARM DD *,SYMBOLS=EXEC SYS
PGM &JAVAHOME/bin/javac
-d &CLASSOUT
&ZFSCBDIR/demo/cobol_java_interop_sample/CobProd.java
//*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *,SYMBOLS=EXEC SYS
JAVA_HOME=&JAVAHOME
CLASSPATH=.:&CLASSOUT
PATH=.:bin:/usr/bin:/usr/local/bin:&ZFSCBDIR/bin
//*
// ENDIF
//*****
//*****
//* STEP 5: Run COBOL/Java interoperable application
//*****
//*****
// IF (BLDJAVA.RUN AND (BLDJAVA.RC EQ 0)) THEN
//GO EXEC PGM=BPXBATSL
//STDPARM DD *,SYMBOLS=EXEC SYS
PGM &JAVAHOME/bin/java CobProd
//*
//STEPLIB DD DSN=&SYSUID..COBOL.LOAD,DISP=SHR
//      DD DSN=&LNGPRFX..SIGYCOMP,DISP=SHR
//      DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//      DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *,SYMBOLS=EXEC SYS
JAVA_HOME=&JAVAHOME
PATH=.:bin:/usr/bin:/usr/local/bin:&ZFSCBDIR/bin
_C89_LSYSLIB=&LIBPRFX..SCEELKED
CLASSPATH=.:&CLASSOUT
CEE_RUNOPTS=POSIX(ON)
IBM_JAVA_OPTIONS=-XX:+Enable3164Interoperability
//*
// ENDIF

```

関連参照

331 ページの『[第 18 章 Java と相互運用する非 OO COBOL アプリケーションのコンパイル、リンク、および実行](#)』

例: COBOL 呼び出しジャワアプリケーション - 構築と実行

このセクションでは、アプリケーションをビルドして実行する方法について説明します。z/OS®ユニックス静的メソッドの呼び出しを示すジャワ COBOL からのメソッド。JCL からアプリケーションをビルドして実行する方法については、次のセクションで説明します。このアプリケーションのバージョンは、以下のファイルセットとして提供されています。demo/cobol_java_interop_sample2 subdirectory COBOL インストールディレクトリの z/OS ユニックスファイルシステム。ファイルのバージョンはコンパイラの SIGYSAMP データセット (名前が接頭辞 igycjdn で始まるメンバーを参照)。

このタスクについて

このデモでは COBOL プログラムを紹介します。prog1 静的呼び出しを行うジャワ方法、myjavaapps.TestApp.doStuff COBOL グループを引数として渡します。COBOL グループ引数はバイト配列として渡されますジャワ、引数はジャワ byte[]型のパラメータとしてサイドを渡し、ジャワコードは単純な方法で引数からデータを抽出します ByteBuffer インターフェース。より複雑なグループの場合は、IBM バイト配列を分解するレコードジェネレーター製品。

デモには次のユーザー ソース ファイルが含まれています。

prog1.cbl

グループを初期化する COBOL プログラム g1 WORKING-STORAGE に保存し、static に渡すジャワ方法 myjavaApps.TestApp.doStuff。

TestApp.java

定義する TestApp 静的メソッドを持つクラス doStuff。

COBPROD アプリケーションをビルドおよび実行するための次のスクリプト ファイルも提供されています。

build.sh

アプリケーションを構築するためのシェル スクリプト。スクリプトの使用法はスクリプト自体に含まれています。

呼び出す build.sh で z/OS ユニックスコマンドライン。

run.sh

相互運用可能なアプリケーションを実行するためのシェル スクリプト。実行方法の詳細についてはスクリプトを参照してください。

その他のさまざまなファイルが提供されています:

README

COBPROD アプリケーションの構築と実行に関する詳細情報が含まれています。

prog1.cbl

```

cbl pgmname(longmixed)
*****
*
* IBM Enterprise COBOL for z/OS
*          Version 6 Release 4 Modification 0
*
* LICENSED MATERIALS - PROPERTY OF IBM.
*
* 5655-EC6 COPYRIGHT IBM CORP. 2024
*
* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
* ADP SCHEDULE CONTRACT WITH IBM CORP.
*
*****
* Simple program that calls a static Java method passing a
* a group argument that is passed to Java as a byte array (byte[])
identification division.
program-id. 'prog1'.
data division.
working-storage section.
01 g1.
   03 num1 pic s9(9) comp-5 value 5112.
   03 num2 pic s9(9) comp-5 value -38236.
   03 msg pic n(24) national value "Hello, from COBOL prog1!".
procedure division.
MainProgram.
   display '<<Entering COBOL program: prog1>>'
   call 'Java.myjavaapps.TestApp.doStuff' using g1
   on exception
      display 'Error calling myjavaapps.TestApp.doStuff'
   end-call
   display '<<Exiting COBOL program: prog1>>'
   goback.
end program 'prog1'.
```

TestApp.java

```
//*****
//
// IBM Enterprise COBOL for z/OS
//          Version 6 Release 4 Modification 0
//
// LICENSED MATERIALS - PROPERTY OF IBM.
//
// 5655-EC6 COPYRIGHT IBM CORP. 2024
//
// US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
// DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
// ADP SCHEDULE CONTRACT WITH IBM CORP.
//
//*****
package myjavaapps;

import java.math.*;
import com.ibm.jzos.ZUtil;
import java.nio.charset.StandardCharsets;
import java.nio.*;

public class TestApp
{
    public static void doStuff(byte[] g1Bytes) throws Exception
    {
        ByteBuffer bb = ByteBuffer.wrap(g1Bytes);

        //NOTE: uncomment the following line when running from JCL
        //      to see output on STDOUT/STDERR DD streams
        //ZUtil.redirectStandardStreams();
        System.out.println("<<Entering Java: TestApp>>");
        System.out.println("doStuff: Hello from TestApp.doStuff!");
        int num1 = bb.getInt();
        int num2 = bb.getInt();
        System.out.println("doStuff: Message from COBOL: " +
            StandardCharsets.UTF_16.decode(bb).toString());
        System.out.println("doStuff: num1 = " + num1);
        System.out.println("doStuff: num2 = " + num2);
        System.out.println("<<Exiting Java: TestApp>>");
    }
}
```

build.sh

```
#!/bin/sh
#####
#
# IBM Enterprise COBOL for z/OS
#          Version 6 Release 4 Modification 0
#
# LICENSED MATERIALS - PROPERTY OF IBM.
#
# 5655-EC6 COPYRIGHT IBM CORP. 2024
#
# US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
# DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
# ADP SCHEDULE CONTRACT WITH IBM CORP.
#
#####
# Build script for the COBOL/Java interoperability demo
#####
#
# Invocation:
# -----
#
# build.sh          - build mixed amode interoperable application
#
# NOTE:
# -----
#
# 1) Make sure cjbuild and cob2 are in your PATH before running this script
#
# 2) Make sure JAVA_HOME is set appropriately
#
# 3) Make sure your STEPLIB includes an Enterprise COBOL V6.4 or later
```

```

# compiler and a runtime that is sufficient to run COBOL programs
# compiled with the Enterprise COBOL V6.4 or later compiler
#
# 4) You must have an existing data set to receive the DLL built
# by cbuild, and the data set must be called <USER>.COBOL.LOAD,
# where <USER> is your high-level qualifier
#
#####

if [ -z $STEPLIB ]; then
    echo "STEPLIB environment variable must be set"
    exit 1
fi

if [ -z $JAVA_HOME ]; then
    echo "JAVA_HOME environment variable must be set"
    exit 1;
fi

OUT_DIR=./out
CLASSOUT_DIR=./class

# Set the AMODE variables
JAVAMODE="JAVA64"
CJMODE="MIX_31_64"

# Make the output directory
rm -fr ${OUT_DIR} ${CLASSOUT_DIR}
mkdir ${OUT_DIR} ${CLASSOUT_DIR}

# ATTN: ensure the STEPLIB environment variable is set appropriately
# ATTN: ensure the directory containing cob2 is in your UNIX PATH

# STEP 1: Compile user COBOL programs, specifying JAVAIOP option
cob2 -c prog1.cbl "-qjavaiop(OUTPATH('${OUT_DIR}'),${JAVAMODE})"

# STEP 2: Build stub file DLL using cbuild
cbuild -v \
    --cobol_dir ${OUT_DIR} \
    --dllout_dir "//COBOL.LOAD" \
    --jar libapp1.jar \
    --java_class_dir ${CLASSOUT_DIR} \
    --mode ${CJMODE} \
    --pkgname enterprise.COBOL \
    app1

# STEP 3: Link user programs
cob2 prog1.o -o prog1 libapp1.x

# STEP 4: Build Java parts of the application
${JAVA_HOME}/bin/javac -d ${CLASSOUT_DIR} TestApp.java

```

run.sh

```

#!/bin/sh
#####
#
# IBM Enterprise COBOL for z/OS
# Version 6 Release 4 Modification 0
#
# LICENSED MATERIALS - PROPERTY OF IBM.
#
# 5655-EC6 COPYRIGHT IBM CORP. 2022, 2024
#
# US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
# DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
# ADP SCHEDULE CONTRACT WITH IBM CORP.
#
#####
# Run script for the COBOL/Java interoperability demo
#####
#
# NOTE:
# ----
#
# 1) Make sure your STEPLIB includes an Enterprise COBOL runtime
# that is sufficient to run programs compiled with the Enterprise
# COBOL V6.4 or later compiler.
#

```

```
# The dataset <USER>.COBOL.LOAD that received the program objects
# and DLL for your interoperable application during the build
# step must also be included in your STEPLIB. We have added this
# in automatically for you below.
#
# 2) Set JAVA_HOME to an appropriate 64-bit Java SDK:
#
# e.g., JAVA_HOME=/usr/lpp/java/IBM/J8.0_64 or
#        JAVA_HOME=/usr/lpp/java/IBM/J11.0_64
#
#####

if [ -z $STEPLIB ]; then
    echo "STEPLIB environment variable must be set"
    exit 1
fi

if [ -z $JAVA_HOME ]; then
    echo "JAVA_HOME environment variable must be set"
    exit 1
fi

# ATTN: ensure the STEPLIB environment variable is set appropriately
export STEPLIB=${USER}.COBOL.LOAD:$STEPLIB
export CLASSPATH=./class:$CLASSPATH
export IBM_JAVA_OPTIONS="-XX:+Enable3164Interoperability"
export JAVA_HOME=/usr/lpp/java/J11.0_64
export _CEE_RUNOPTS="POSIX(ON)"
export LIBPATH=./:${JAVA_HOME}/lib/s390x/j9vm:${JAVA_HOME}/lib/s390x/${JAVA_HOME}/lib:${JAVA_HOME}/lib/j9vm:${LIBPATH}

./prog1
```

関連参照

[355 ページの『COBOL-calls-java アプリケーションをビルドして実行するためのサンプル JCL』](#)

COBOL-calls-java アプリケーションをビルドして実行するためのサンプル JCL

このセクションには、COBOL 呼び出しを構築および実行するためのサンプル JCL が含まれています。ジャワ応用。前のセクションでは、このアプリケーションと、それをビルドして実行する方法について説明しました。z/OS®ユニックスシェルスクリプト。しかし、一部のユーザーは、非 OO COBOL/ジャワ JCL を使用した相互運用可能なアプリケーション。提供されている JCL は、このデモアプリケーション専用ですが、非 OO COBOL/ジャワ COBOL 側で開始し、COBOL の静的呼び出し機能を備えた相互運用可能なアプリケーションジャワ方法。

このタスクについて

JCL の多くは、COBOL アプリケーションを構築およびリンクするための一般的な JCL です。主な違いは、ビルドプロセスに、cjbuild ツールを使用して相互運用可能なアプリケーションの DLL をビルドする手順が含まれていることです。cjbuild はネイティブ z/OS ユニックスこのサンプルでは、BPXBATCH ユーティリティを使用して JCL から簡単にプログラムを実行する方法を示します。

非 OO COBOL/の構築と実行に関する考慮事項の詳細については、ジャワ JCL からの相互運用可能なアプリケーションについては、[COBPROD アプリケーションをビルドして実行するためのサンプル JCL](#)。

COBOL-calls-java アプリケーションをビルドして実行するためのサンプル JCL

```
//USERID2 JOB 'JAVA/COBOL interop',
// MSGLEVEL=(1,1),MSGCLASS=S,CLASS=C,NOTIFY=&SYSUID,
// REGION=0M
//*****
//*          IBM Enterprise COBOL for z/OS          */
//*****
//*****
//*****
//*          *
//*          LICENSED MATERIALS -- PROPERTY OF IBM.  *
//*          *
//*          5655-EC6                                *
//*          COPYRIGHT IBM CORP. 2024                *
```

```

//*
//*      US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR
//*      DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM
//*      CORP.
//*
//*****
//*****
//*
//* Sample JCL for building and running a non-OO COBOL/Java
//* interoperable application starting on the COBOL side with
//* COBOL calling a static Java method.
//*
//*****
//*
//* Prerequisites:
//*
//* 1) User COBOL source files must be copied to data set:
//*     USER.COBOL.COBOL.
//*
//* 2) A PDSE data set named USER.COBOL.OBJECT must exist
//*     to receive object program files generated by the COBOL
//*     compiler.
//*
//* 3) A PDSE data set USER.COBOL.LOAD must exist to receive
//*     program object files. These program objects come from
//*     linking object files from user programs and also from
//*     linking the object files associated with stub programs
//*     into a DLL.
//*
//*     A PDS data set (FB 80) called USER.COBOL.EXP must also
//*     exist prior to running this JCL. This data set will
//*     receive the side deck for the DLL produced by cjbuild
//*     and is used during subsequent link steps.
//*
//* 4) You must ensure the following JCL variables are set below,
//*     which control various build parameters. This includes the
//*     z/OS UNIX directory into which stub files will be generated
//*     by the compiler and the z/OS UNIX work directory to be
//*     used by the cjbuild utility. The cjbuild utility uses this
//*     directory to build the DLL needed for the interoperable
//*     application.
//*
//*     For this sample, we assume the z/OS UNIX directory
//*     structure to be used for building the application is as
//*     follows:
//*
//*     /home/userid/jtoc/test/simple/ccj/work
//*     - main work directory
//*     /home/userid/jtoc/test/simple/ccj/out
//*     - stub file output directory (compiler places stub
//*       files here)
//*     /home/userid/jtoc/test/simple/ccj/class
//*     - directory where cjbuild places any generated .class
//*       files
//*
//*     The following variables must be set:
//*
//*     JAVAHOME
//*     ZFSCBDIR
//*     OUTDIR
//*     INTMODE
//*     WORKDIR
//*     CLASSOUT
//*     JVSRCOUT
//*     PKGNAME
//*     DLLNAME
//*
//*     See below for detailed descriptions of how to set these
//*     variables. Example values are provided.
//*
//*     NOTE: z/OS UNIX directories for stub files and the work
//*           directory for cjbuild do not have to exist. The
//*           JCL below create the directories if they do not
//*           exist prior to running this job (see STEP 0).
//*
//*****
//* EXPORT SYMLIST=*
//*****
//* Set compiler and run time library levels below
//*****
//*     SET LIBPRFX='CEE'
//*     SET LNGPRFX='COBOL.COMPILER'

```

```

//*****
//* JAVAHOME: Set this to your Java home directory. This will be used
//*           to set the JAVA_HOME environment variable needed by
//*           Java operations
//*****
//      SET JAVAHOME='/usr/lpp/java/J11.0_64'
//*****
//* ZFSCBDIR: This should point to the location of your COBOL zFS
//*           install directory (e.g., /usr/lpp/cobol/igyv6r4)
//*
//*           The bin directory of this directory is where the cob2
//*           and cjbuild utilities reside. They will be needed
//*           when building the interoperable application
//*****
//      SET ZFSCBDIR='/home/userid/cobol/igyv6r4'
//*****
//* OUTDIR: Set this to the z/OS UNIX directory that will receive
//*          the stub files and other artifact files generated by
//*          the COBOL compiler compiling programs that contain
//*          the JAVA-CALLABLE and JAVA-SHAREABLE directives or
//*          programs that make calls to Java using a CALL statement
//*          of the form:
//*
//*          CALL 'Java.class-name.static-method-name' ...
//*
//*          The cjbuild utility (see STEP 2) will get the files from
//*          this directory (see --coboldir option) so the directory
//*          specified for OUTDIR should be used both in the OUTPATH
//*          suboption of JAVAIOPI when compiling the user COBOL
//*          program as well as the --coboldir option of cjbuild.
//*****
//      SET OUTDIR='/home/userid/jtoc/test/sample2/out'
//*****
//* INTMODE: Set this to the interoperability mode to be used
//*          by your application (needed by the cjbuild utility)
//*
//*          PURE31 - AMODE 31 COBOL, AMODE 31 Java
//*          PURE64 - AMODE 64 COBOL, AMODE 64 Java
//*          MIX_31_64 - AMODE 31 COBOL, AMODE 64 Java
//*
//*          NOTE: if you specify MIX_31_64, you must add the
//*          JAVA64 suboption to the JAVAIOPI option of
//*          all related user COBOL programs.
//*****
//      SET INTMODE='MIX_31_64'
//*****
//* WORKDIR: Set this to the z/OS UNIX directory to be used as a work
//*          directory for building the interoperable application.
//*          cjbuild will be run from this directory and basic
//*          artifact files it generates (object files, listing,
//*          etc.) go to this directory automatically.
//*
//*          This is where the "methods" file will be located for
//*          this sample.
//*****
//      SET WORKDIR='/home/userid/jtoc/test/sample2'
//*****
//* CLASSOUT: Set this to the directory you want any .class files
//*            generated by cjbuild to go. These are the .class
//*            files for Java accessor classes for JAVA-SHAREABLE
//*            items, and also .class files for any Java files
//*            generated by cjbuild, such as cobol.java, progs.java,
//*            and strg.java.
//*****
//      SET CLASSOUT='/home/userid/jtoc/test/sample2/class'
//*****
//* JVSRCOUT: Set this to the directory you want any .java files
//*            generated by cjbuild to go (cobol.java, progs.java,
//*            strg.java).
//*****
//      SET JVSRCOUT='/home/userid/jtoc/test/sample2'
//*****
//* PKGNAME: Set this to the name of the Java package that you
//*           want the 'progs' and 'strg' class to be a part of.
//*           The default is enterprise.COBOL. This affects how
//*           COBOL programs are invoked from Java. e.g., if the
//*           package name is enterprise.COBOL., then you would
//*           invoke a COBOL program called PROG1 as follows:
//*
//*           enterprise.COBOL.progs.PROG1();
//*****
//      SET PKGNAME='enterprise.COBOL'

```

```

//*****
//* DLLNAME: The name of the DLL that cjbuild creates and which
//*           is needed at runtime when interoperable activities
//*           occur in the application.
//*
//*           If the DLL is targeted for a data set (which is
//*           the usual way, see --dlloutdir option of cjbuild)
//*           then the DLL will be stored as a member of the
//*           specified data set, and is called <DLLNAME>.
//*
//*           If the DLL is targeted to the z/OS UNIX file system,
//*           it will be called lib<dllname>.so.
//*****
//      SET DLLNAME=APP1
//*****
//PROCLIB JCLLIB ORDER=(&LNGPRFX..SIGYPROC)
//*****
//*****
//* STEP 0: Ensure required z/OS UNIX directories exist
//*****
//CRDIRS   EXEC PGM=BPXBATCH
//STDPARM  DD *,SYMBOLS=EXECSYS
SH mkdir -p &OUTDIR;
mkdir -p &WORKDIR;
mkdir -p &CLASSOUT;
mkdir -p &JVSRCOUT
/*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD *
NO_CUSTOM_SHELL=1
/*
//*****
//*****
//* STEP 1: Compile user COBOL programs, specifying JAVAIO option
//*****
//*****
// SET INFILE='PROG1'
//*****
// IF (CRDIRS.RUN AND (CRDIRS.RC EQ 0)) THEN
//COBOL1   EXEC IGYWC,
//  LNGPRFX=&LNGPRFX,
//  LIBPRFX=&LIBPRFX,
//  PARM.COBOLE='OPTFILE'
//SYSLIB   DD DSN=&SYSUID..COBOL.COBOLE,DISP=SHR
//SYSIN    DD DSN=&SYSUID..COBOL.COBOLE(&INFILE),DISP=SHR
//SYSLIN   DD DSN=&SYSUID..COBOL.OBJECT(&INFILE),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOPTF  DD *,SYMBOLS=EXECSYS
//          JAVAIO(OUTPATH(' &OUTDIR'),JAVA64)
/*
//SYSOUT   DD SYSOUT=*
// ENDIF
//*****
//* STEP 2: Build stub file DLL using cjbuild
//*****
// IF (COBOL1.COBOLE.RUN AND (COBOL1.COBOLE.RC EQ 0)) THEN
//CJBUILD EXEC PGM=BPXBATCH
//STDPARM  DD *,SYMBOLS=EXECSYS
SH export JAVA_HOME=&JAVAHOME;
PATH=./bin:/usr/bin:/usr/local/bin:&ZFSCBDIR/bin;
cd &WORKDIR;

&ZFSCBDIR/bin/cjbuild
-v
--cobolldir &OUTDIR
--dlloutdir "/*'&SYSUID..COBOLE.LOAD'"
--pkgname &PKGNAME
--javaclssdir &CLASSOUT
--javasrcdir &JVSRCOUT
--mode &INTMODE
&DLLNAME
/*
//STEPLIB DD DSN=&LNGPRFX..SIGYCOMP,DISP=SHR
//        DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//        DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//STDOUT  DD SYSOUT=*
//STDERR  DD SYSOUT=*
//STDENV  DD *,SYMBOLS=EXECSYS
NO_CUSTOM_SHELL=1
_C89_LSYSLIB=&LIBPRFX..SCEELKED

```



```

/*
// ENDIF
//*****
//*****
/* STEP 3: Link user programs. User programs may call out to
/* programs in the DLL produced by cjbuild in STEP 2
/* thus should be linked with the side deck produced in
/* that step
//*****
//*****
// SET INFILE='PROG1'
//*****
// IF (CJBUILD.RUN AND (CJBUILD.RC EQ 0)) THEN
//LKED1 EXEC PGM=IEWL,
// PARM='RENT,LIST,LET,DYNAM(DLL),CASE(MIXED)'
//SYSLIB DD DSN=&LIBPRFX..SCEELKED,DISP=SHR
// DD DSN=&LIBPRFX..SCEELKEX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSMOD DD DSN=&SYSUID..COBOL.LOAD(&INFILE),DISP=SHR
//SYSDEFSD DD DUMMY
//OBJMOD DD DSN=&SYSUID..COBOL.OBJECT(&INFILE),DISP=SHR
//SIDEDK DD DSN=&SYSUID..COBOL.EXP(&DLLNAME),DISP=SHR
//SYSLIN DD *
INCLUDE OBJMOD
INCLUDE SIDEDK
/*
// ENDIF
//*****
//*****
/* STEP 4: Builds Java application
//*****
//*****
// IF (LKED1.RUN AND (LKED1.RC EQ 0)) THEN
//BLDJAVA EXEC PGM=BPXBATSL
//STDPARM DD *,SYMBOLS=EXECSYS
PGM &JAVAHOME/bin/javac
-d &CLASSOUT
&WORKDIR/TestApp.java
/*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//STDENV DD *,SYMBOLS=EXECSYS
JAVA_HOME=&JAVAHOME
CLASSPATH=.:&CLASSOUT
PATH=./bin:/usr/bin:/usr/local/bin:&ZFSCBDIR/bin
/*
// ENDIF
//*****
//*****
/* STEP 5: Run COBOL/Java interoperable application
//*****
//*****
// IF (BLDJAVA.RUN AND (BLDJAVA.RC EQ 0)) THEN
//GO EXEC PGM=PROG1
//STEPLIB DD DSN=&SYSUID..COBOL.LOAD,DISP=SHR
// DD DSN=&LNGPRFX..SIGYCOMP,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//CEEOPPTS DD *,SYMBOLS=EXECSYS
POSIX(ON)
ENVAR(
"JAVA_HOME=&JAVAHOME",
"IBM_JAVA_OPTIONS=-XX:+Enable3164Interoperability",
"PATH=./bin:/usr/bin:/usr/local/bin",
"CLASSPATH=&CLASSOUT",
"LIBPATH=&JAVAHOME/lib:&JAVAHOME/lib/j9vm")
/*
// ENDIF

```

関連参照

[351 ページの『例: COBOL 呼び出しジャワアプリケーション - 構築と実行』](#)

トラブルシューティング

非 OO COBOL アプリケーションを構築して実行する際に、以下の方法を使用して問題をトラブルシューティングできます。ジャワ。

このタスクについて

1. 必ず cob2 および cjbuild は PATH 環境変数内にあります。JCL からビルドする場合は、ZFSCBDIR JCL 変数を zFS COBOL インストール・ディレクトリの場所に設定してください。
2. JAVA_HOME が環境に設定され、適切なパスを指していることを確認してください。ジャワ JBDK 1.0.1 以降。例:

```
export JAVA_HOME=/usr/lpp/java/IBM/J8.0_64
```

または:

```
export JAVA_HOME=/usr/lpp/java/IBM/J11.0_64
```

JCL からビルドして実行する場合は、JAVAHOME JCL 変数が適切に設定されています。

3. 実行時に次の環境変数が設定されていることを確認してください。

のために z/OS ユニックススクリプト:

```
export IBM_JAVA_OPTIONS="-XX:+Enable3164Interoperability"  
export _CEE_RUNOPTS="POSIX(ON)"
```

JCL から実行する場合は、CEE0PTS DD 次のように処理されます。

```
//CEE0PTS DD *,SYMBOLS=EXEC SYS  
POSIX(ON)  
ENVAR(  
  "JAVA_HOME=&JAVAHOME",  
  "IBM_JAVA_OPTIONS=-XX:+Enable3164Interoperability",  
  "PATH=./bin:/usr/bin:/usr/local/bin",  
  "CLASSPATH=&CLASSOUT",  
  "LIBPATH=&JAVAHOME/lib:&JAVAHOME/lib/j9vm")
```

4. COBOL 側で起動したアプリケーションが Java を呼び出し、Java がターミナルに表示するが、JCL から実行すると出力が表示されない場合は、Java プログラムの先頭に次の行を追加してみてください:

```
ZUtil.redirectStandardStreams();
```

5. cjbuild ユーティリティの実行中にリンク エラーが発生した場合は、環境に次のエクスポートを追加してみてください。

```
export _C89_LSYSLIB=CEE.SCEELKED
```

注: ランタイム データセットを別の HLQ にインストールしている場合は、代わりにそれらのデータセットに対応する SCEELKED を使用します。

リンクエラーメッセージ:

```
# Linking native method DLL  
/home/userid/cobol/igyv6r4/bin/cob2 -bdll -o //COBOL.LOAD(libapp1)  
COBPROD_java_native_app1.o COBPROD_java_native_WS_app1.o addlist_java_native_app1.o  
Java.CobProd.printResult.o /home/userid/jtoc/demo2/java_new5/J8.0_64/bin/j9vm/libjvm.x /home/  
userid/cobol/igyv6r4/lib/igzcnj2.x  
# Command return code 3:  
IEW2456E 9207 SYMBOL IGZXLERR UNRESOLVED. MEMBER COULD NOT BE INCLUDED FROM  
THE DESIGNATED CALL LIBRARY.  
IEW2456E 9207 SYMBOL IGZXDER UNRESOLVED. MEMBER COULD NOT BE INCLUDED FROM THE  
DESIGNATED CALL LIBRARY.  
IEW2456E 9207 SYMBOL IGZXDDE UNRESOLVED. MEMBER COULD NOT BE INCLUDED FROM THE  
DESIGNATED CALL LIBRARY.  
IEW2638S 5384 AN EXECUTABLE VERSION OF MODULE *NULL* EXISTS AND CANNOT BE
```

非 OO COBOL/Java 相互運用可能アプリケーションの実行に関する AMODE の考慮事項

このトピックでは、非 OO COBOL/Java 相互運用可能アプリケーションに関する AMODE 関連の考慮事項について説明します。

このタスクについて

Java 8 で実行される混合 AMODE アプリケーション (つまり AMODE 31 COBOL および 64 ビット Java アプリケーション) の場合は、LIBPATH を以下のように設定します。

```
export LIBPATH=${JAVA_HOME}/lib/s390x/j9vm:${JAVA_HOME}/lib/s390x/:${LIBPATH}
```

Java 11、Java 17、または Java 21 上で動作する混合 AMODE アプリケーションの場合、LIBPATH を以下のように設定する：

```
export LIBPATH=${JAVA_HOME}/lib:${JAVA_HOME}/lib/j9vm:${LIBPATH}
```

どちらの場合も、混合 AMODE Java 相互運用可能 COBOL アプリケーションに対して、実行時に以下の環境変数を設定します。

```
export IBM_JAVA_OPTIONS="-XX:+Enable3164Interoperability"
```

cjbuild コマンドリファレンス

cjbuild ユーティリティは、相互運用可能な非 OO COBOL アプリケーションの構築を支援するために提供されています。ジャワ。このようなアプリケーションの COBOL ファイルがコンパイルされると、COBOL とジャワ。これらのスタブ ファイルは、アプリケーションの実行時に使用できる DLL に組み込む必要があります。

注：cjbuild ユーティリティは、z/OS ユニックスコマンドラインまたは JCL から BPXBATCH ユーティリティを使用して実行します。ただし、ユーティリティが生成する DLL は、MVS データセットまたは UNIX ディレクトリをターゲットにすることができます。z/OS UNIX ディレクトリをターゲットにすることができます。

cjbuild コマンドの使用方法和パラメータの説明

cjbuild ユーティリティを実行するには、次のコマンドを実行します。

```
cjbuild [-h/--help] [-v/--verbose] [--version]
        [-c/--cobol-dir <cobol-dir>]
        [-d/--dllout-dir <dll-out-dir>]
        [-i/--progid <program-id>] ...
        [-j/--java-class-dir <java-class-dir>]
        [-k/--cleanscript <cleanup-script-name>]
        [-p/--pkgname <package-name>]
        [-r/--jar <jar-file>]
        [-s/--java-src-dir <java-src-dir>]
        [<program-file>] <lib-base-name>
```

ここで、それぞれ以下のとおりです。

<program-file>

注：この引数は、cjbuild の以前のバージョンでは <methods-file> と呼ばれ、必須引数でした。現在はオプションとなっています。

オプションの入力ファイル。相互運用可能なアプリケーション内の COBOL プログラムのリストを含み、そのスタブファイルは COBOL アーティファクトディレクトリ (-c/--cobol-dir オプション) は

cjbuild によって処理されます。いつ <program-file>指定されていない、そして-i も指定されていない場合は、ディレクトリ内のすべてのスタブ ファイルが処理されます。リストアップされるプログラムは、アプリケーション内で相互運用性機能を持つプログラムである必要があります。 >>JAVA-CALLABLE 指令は、>>JAVA-SHAREABLE 指令、または CALL 次の形式の文:

```
CALL 'Java.<java-class-name>.<java-static-method-name>'
```

コンパイルされたプログラムは>>JAVA-CALLABLE ディレクティブは、指定されたパッケージ内の「progs」というクラスにグループ化されます (-p)、および範囲に含まれるデータ項目>>JAVA-SHAREABLE リストされた COBOL プログラムのいずれかのディレクティブは、指定されたパッケージ内の 'strg' というクラスにグループ化されます。例えば、2つの要素から成るアプリケーションのプログラムファイルの内容は、ジャワ-呼び出し可能な COBOL プログラム COBPROG1 そして COBPROG2 そのうちの1つは静的な呼び出しを行うジャワ方法 'Java.com.acme.MyClass.myMethod' 次のようになります。

```
COBPROG1
COBPROG2
```

<lib-base-name>

相互運用可能なアプリケーションの実行時に必要なスタブコードを含む DLL の名前を作成するために使用されます (ジャワ COBOL ではこの DLL が必要になる場合があります)。この名前は、.jar ファイルの作成時にベース名としても使用されます。-r/--jar オプションは指定されておらず、.jar ファイル名がデフォルトになります。例えば、<lib-base-name> が app1 である場合、-d オプションの引数が z/OS UNIX ディレクトリであれば DLL は libapp1 と呼ばれ、-d オプションの引数が MVS PDS データセットであれば DLL は APP1 と呼ばれる。 .jar ファイルは、--jar <jar-file> オプションが指定されていない場合 libapp1.jar。

注: Enterprise COBOL 6.5 以前は、DLL が MVS PDS データセットを対象としている場合でも、指定された DLL 名に 3 文字の接頭辞 LIB が付加されていた。このような場合に接頭辞が追加されなくなったので、ビルド JCL のリンク・ステップを更新して、関連する DLL 側のデッキ・ファイルを参照するときに LIB 接頭辞を削除する必要があります。

cjbuild コマンド リファレンスのオプションについては、次の表を参照してください。

表 46. オプションと説明	
オプション	説明
--cleanscript <cleanup-script-name> -k <cleanup-script-name>	コンパイラが生成するクリーンアップ・スクリプトの名前をカスタマイズする。ユーザー指定のファイル名は'.sh'で終わっていなければならない。このスクリプトは、-c、-s、-j、-d の各ディレクトリにある cjbuild によって生成された成果物をクリーンアップするためのものである。COBOL コンパイラによって生成されたスタブ・ファイルも削除するために、クリーンアップ・スクリプトを起動するときにオプション --all-artifacts を指定することができる。--force-clean cjbuild と COBOL Compiler の以前のすべての起動によって生成されたカレント・ディレクトリと 内のすべてのファイルを削除するクリーンアップ・スクリプトを起動するときに指定できる。-c/--coboldir デフォルト。'cjbuild_clean.sh'

表 46. オプションと説明 (続き)	
オプション	説明
--coboldir <cobol-dir> -c <cobol-dir>	COBOL 成果物ディレクトリーへのパス。このディレクトリーには、COBOL プログラムをコンパイルする際にコンパイラによって生成されたファイルが含まれています。JAVA-CALLABLE および/または JAVA-SHAREABLE 指令または CALL' 形式のリテラル呼び出しターゲットを持つステートメント Java.java-class-name.java-static-method-name'。 デフォルト: 現行ディレクトリー
--dlloutdir <dll-out-dir> -d <dll-out-dir>	スタブコードの DLL を配置するパス。PDS の場合は、次の形式のパスを使用します。 <pre>//<DATASET-NAME></pre> 例: <pre>//COBOL.LOAD or //'USER.COBOL.LOAD'</pre> デフォルト: 現行ディレクトリー
--help -h	詳細なヘルプを表示します。 デフォルト: 適用外
--progid <program-id> -i	指定されたディレクトリー内のスタブファイルを持つ COBOL プログラムの名前を示します。-c /--coboldir オプションは cjbuild によって処理されます。 このオプションは、cjbuild の処理に含めるプログラムごとに 1 回ずつ、複数回指定できます。これは、同じ目的を果たす<program-file>引数に似ていますが、-i 入力ファイルではなくコマンドラインで cjbuild に同じ情報を提供できるようになります。 <program-id> は、プログラムの program-id 句で指定された COBOL プログラムの名前に対応します。COBOL プログラムで名前を引用符なしで指定する場合は、-i オプション。COBOL プログラムで名前がリテラルとして指定されている場合、-i オプションは、COBOL プログラムの program-id 句と正確に一致する必要があります。 デフォルト: 適用外
--javaclassdir <java-class-dir> -j <java-class-dir>	生成された出力クラスファイルの場所へのパスジャワコード。このパスはあなたのクラスパス環境変数。 デフォルト: 現行ディレクトリー
--javasrcdir <java-src-dir> -s <java-src-dir>	出力場所へのパスジャワソースファイル。 デフォルト: 現行ディレクトリー

表 46. オプションと説明 (続き)	
オプション	説明
<pre>--jar <jar-file> -r</pre>	作成する Java .jar コンパイルによって生成された.class ファイルからファイルジャワ COBOL コンパイラと cjbuild によって生成されたソース ファイル。 .jar ファイルの名前は、<jar-file>口論。 .jar ファイルはジャワ実行時の CLASSPATH。 デフォルト: 名前の jar ファイル lib<lib-base-name>.jar 現在のディレクトリに作成されます。
<pre>--mode <PURE31 PURE64 MIX_31_64> -m <PURE31 PURE64 MIX_31_64> (DEPRECATED)</pre>	ジャワ/COBOL 相互運用性モード。 特定 PURE31AMODE 31 の場合 COBOL/31-bit ジャワ応用。 特定 PURE64AMODE 64 の場合 COBOL/64-bit ジャワ応用。 特定 MIX_31_64AMODE 31 の場合 COBOL/64-bit ジャワ応用。 注: COBOL ファイルをビルドするには、JAVAIOP (JAVA64) オプション MIX_31_64 モードアプリケーション。 デフォルト。PURE31 注: このオプションは cjbuild では不要になった。モードはスタブファイルから自動的に推測されるようになった。それでも指定は可能だが、推論モードと衝突しない限り、その値は無視され、その場合はエラーメッセージが発行される。
<pre>--pkgname <package-name> -p <package-name></pre>	これらのネイティブ メソッドが関連付けられるパッケージの名前。各アプリケーションには独自のパッケージ名が必要です。 例: <pre>com.acme.COBOL.myapp1</pre> このパッケージの一部として 3 つのクラスが作成されます。 <pre>com.acme.COBOL.myapp1.cobol com.acme.COBOL.myapp1.progs com.acme.COBOL.myapp1.strg</pre> デフォルト。enterprise.COBOL
<pre>--verbose -v</pre>	冗長な出力。実行されたコンパイル/リンク コマンドを表示します。 デフォルト: 適用外
<pre>--version</pre>	cjbuild のバージョン情報を表示する。

関連タスク

331 ページの『第 18 章 Java と相互運用する非 OO COBOL アプリケーションのコンパイル、リンク、およ

[び実行』](#)

338 ページの『[例: COBPROD アプリケーション - ビルドと実行』](#)

関連参照

JAVA 呼び出し可能 (*Enterprise COBOL for z/OS* 言語解説書)

JAVA-SHAREABLE (*Enterprise COBOL for z/OS* 言語解説書)

CALL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

第 19 章 COBOL AMODE 64 アプリケーションのコンパイル、バインド、および実行

Enterprise COBOL AMODE 64 アプリケーションは、z/OS 環境下で、JCL、TSO コマンド、CLIST、または ISPF パネルを使用してコンパイル、バインド、および実行できます。この処理は、COBOL AMODE 31 アプリケーションの同様のタスクとほぼ同じです。主な違いと制限については、関連タスクを参照してください。

このタスクについて

関連タスク

367 ページの『z/OS の下で AMODE 64 プログラムをコンパイルおよびバインド』

368 ページの『DLL に関する考慮事項』

369 ページの『COBOL AMODE 64 アプリケーションの実行』

369 ページの『COBOL 専用ランタイム・オプションの指定』

z/OS の下で AMODE 64 プログラムをコンパイルおよびバインド

Enterprise COBOL AMODE 64 アプリケーションは、z/OS の下では、ジョブ制御言語 (JCL)、TSO コマンド、CLIST、または ISPF パネルを使用してコンパイルできます。この処理は、COBOL AMODE 31 アプリケーションの同様のタスクと同じです。

このタスクについて

主な違いは以下のとおりです。

- LP(64) コンパイラー・オプションを指定すること
- 言語環境・システム・ライブラリー SCEEBND2 をバインド・ステップの SYSLIB DD に組み込むこと
- 言語環境・システム・ライブラリー PDSE メンバー SCEELIB(CELQV004) をバインド・ステップの SYSLIN DD に組み込むこと
- バインダー・オプション RENT を組み込むこと
- LP(64) COBOL プログラムのバインド・ステップで明示的な ENTRY ステートメントを指定する場合、そのステートメントは ENTRY CELQSTRT を指定していなければなりません。言い換えれば、LP(64) プログラムの入り口点は PROGRAM-ID ステートメントでの名前ではなく、CELQSTRT になります。
- のために LP(64) COBOL プログラムを動的に呼び出す場合、program-name の中に PROGRAM-ID 段落は対応するプログラム オブジェクト名と同一である必要があります。詳細については、607 ページの『動的呼び出しの作成』を参照してください。

JCL を使ったコンパイルにおいて、IBM は一連のカatalog式プロシージャーを用意しています。これを使用すれば、書かなければならない JCL コーディングの量を減らすことができます。Catalog式プロシージャーが要件に合わない場合は、独自の JCL を書くことができます。JCL を使用して、単一のプログラムをコンパイルすること、または複数のプログラムをバッチ・ジョブの中でコンパイルすることができます。

説明のために、次の JCL コード・フラグメントは、既存のコンパイルおよびバインドのステップ JCL を変更して COBOL AMODE 64 プログラムを作成する方法を示しています。

```
//COMPILE EXEC PGM=IGYCRCTL,REGION=0M,
//  PARM='LP(64)'                    <- add LP(64) option
// ...
//BIND EXEC. PGM=IEWBLINK,REGION=0M,
//  PARM='RENT,DYNAM(DLL)'           <- add RENT and DYNAM(DLL) options
//SYSLIB DD. DSN=CEE.SCEEBND2,DISP=SHR <- include .SCEEBND2
// ...
//SYSLIN DD ...
//  DD DSN=CEE.SCEELIB(CELQV004),DISP=SHR <- include member CELQV004
// ...
```

また、以下のカタログ式プロシージャーを使用することもできます。

1. IGYQC - COBOL AMODE 64 プログラムをコンパイルするための単一ステップのカタログ式プロシージャー
2. IGYQCB - COBOL AMODE 64 プログラムをコンパイルおよびバインドするための 2 ステップのカタログ式プロシージャー
3. IGYQCBG - COBOL AMODE 64 プログラムをコンパイル、バインド、および実行するための 3 ステップのカタログ式プロシージャー

DLL に関する考慮事項

いつ LP(64)が指定されている場合、生成された AMODE 64 COBOL プログラムは、DLL コンパイラー オプションに関係なく DLL になります。

このタスクについて

モジュールを他の LP(64) モジュールに静的にバインドできます。

実行時に呼び出される DLL にバインドする場合は、EXPORTALL オプションを使用して、DLL の呼び出し元が使用するプログラム名を公開する必要があります。

静的の場合も実行時の場合も、バインドには常に DYNAM(DLL) を指定する必要があります。

関連タスク

[396 ページの『DLL』](#)

[398 ページの『DYNAM』](#)

[401 ページの『EXPORTALL』](#)

[415 ページの『LP』](#)

z/OS ユニックス の下で AMODE 64 プログラムをコンパイルおよびバインド

z/OS ユニックス で COBOL AMODE 64 プログラムをコンパイルするためのセットアップおよび準備は、AMODE 31 での同様のセットアップおよび準備と同じです。詳しくは、[311 ページの『第 16 章 z/OS ユニックスのもとでのコンパイル』](#)を参照してください。

このタスクについて

コンパイルは cob2 コマンドの -q64 オプションを使用して行われます。例えば、次のコマンドは、現行ディレクトリー内の COBOL ソース・ファイル hello.cb1 をコンパイルしてバインドします。バインダー出力は、デフォルトの実行可能ファイル a.out に書き込まれます。

```
cob2 -q64 hello.cb1
```

-o オプションを使用すれば、バインダー出力ファイルを明示的に指定できます。次の cob2 コマンドは、実行可能ファイル hello.exe を生成します。

```
cob2 -q64 hello.cb1 -o hello.exe
```

AMODE 31 で cob2 コマンドを使用するときと同じ方法で -q オプションを使用すれば、他のコンパイラー・オプションを指定できます。例えば、次のコマンドは、SOURCE オプションと LIST オプションを使用して COBOL コンパイラーを呼び出します。

```
cob2 -q64 -q"SOURCE,LIST" hello.cb1
```

-q64 は cob2 コマンド行ではコンパイラー・オプション LP(64) の簡易形式であることに注意してください。これは、ほかに指定されるコンパイラー・オプションとは別に cob2 コマンド行にそのまま指定する

必要があります。-q64 の代わりに他のコンパイラー・オプションとともに LP(64) を直接使用することができます。

```
cob2 -q"LP(64),SOURCE,LIST" hello.cbl
```

同様に -q32 はコンパイラー・オプション LP(32) の簡易形式です。-q64 や -q32 を明示的に指定せずに cob2 が呼び出された場合のデフォルトは -q32 です。

環境変数

AMODE 31 でサポートされる環境変数のほかに、次の 2 つの追加環境変数が AMODE 64 でサポートされます。

_C89_L6SYSLIB

ライブラリデータセットの連結を指定します SYSLIB DD バインドステップで使用されます。データ・セット名はコロンで区切られます。例えば、次の環境変数設定があるとします。

```
export _C89_L6SYSLIB=MYSAMPLE.SCEEBND2:CEE.SCEEBND2
```

この場合は、バインド・ステップで次の SYSLIB DD 連結が提供されます。

```
//SYSLIB DD DSN=MYSAMPLE.SCEEBND2,DISP=SHR
// DD DSN=CEE.SCEEBND2,DISP=SHR
```

デフォルトでは、_C89_L6SYSLIB が指定されない場合は、次の同等の設定が使用されます。

```
export _C89_L6SYSLIB=CEE.SCEEBND2
```

_C89_L6SYSIX

これは、バインド・ステップで使用する SYSLIN のサイド・デック・データ・セット連結を指定します。データ・セット名はコロンで区切られます。例えば、次の環境変数設定があるとします。

```
export _C89_L6SYSIX="MYSAMPLE.SCEELIB(CELQV004):CEE.SCEELIB(CELQV004)"
```

この場合は、バインド・ステップで次の SYSLIN DD 連結が提供されます。

```
//SYSLIN DD DSN=MYSAMPLE.SCEELIB(CELQV004),DISP=SHR
// DD DSN=CEE.SCEELIB(CELQV004),DISP=SHR
```

デフォルトでは、_C89_L6SYSIX が指定されない場合は、次の同等の設定が使用されます。

```
export _C89_L6SYSIX="CEE.SCEELIB(CELQV004)"
```

COBOL AMODE 64 アプリケーションの実行

COBOL AMODE 64 プログラムを含むアプリケーションの実行を準備する場合は、STEPLIB DD ステートメントなどを使用して、SCEERUN2 および SCEERUN 言語環境 ロード・ライブラリーがシステム・ライブラリー検索順序で使用可能であることを確認してください。

このタスクについて

注: AMODE 64 COBOL アプリケーションは z/OS 2.2 以上でコンパイルできますが、AMODE 64 COBOL アプリケーションを実行するには z/OS 2.3 以上でなければなりません。

COBOL 専用ランタイム・オプションの指定

AMODE 64 COBOL 専用ランタイム・オプションは、IGZOPTS DD ステートメントを使用することによって、または環境変数 _IGZ_RUNOPTS を設定することによって指定できます。

このタスクについて

以下に CBLQDA オプションおよび DEBUG オプションの設定例を示します。

```
//...  
//IGZOPTS DD *  
  DEBUG,CBLQDA(ON)  
/*  
//*
```

オプション構文は、言語環境 CEEOPTS DD ステートメントを使用して AMODE 31 で指定される同様の構文と同じです。

z/OS ユニックス 環境下でプログラムを実行する場合は、`_IGZ_RUNOPTS` 環境変数を使用して COBOL 専用ランタイム・オプションを指定できます。オプション構文は、LE 環境変数 `_CEE_RUNOPTS` を使用して AMODE 31 で指定される同様の構文と同じです。次の例では、DEBUG オプションおよび CBLQDA オプションが指定されています。

```
export _IGZ_RUNOPTS="DEBUG,CBLQDA(ON)"
```

AMODE 64 COBOL 専用ランタイム・オプションは、これら 2 つのメソッドを使用することによってのみ指定できます。COBOL ランタイムオプションの指定方法の詳細については、[以下を参照してください。470 ページの『COBOL ランタイム オプションの指定』](#)。

注：CBLOPTS ランタイム・オプションは AMODE 64 ではサポートされていません。メインプログラムのパラメーター・ストリング上で言語環境・ランタイム・オプションを指定する場合、ランタイム・オプションはプログラム引数の前に指定する必要があります。この振る舞いは他の言語環境高水準言語と整合します。次の例は、プログラム引数「20190901」およびランタイム・オプション POSIX(ON) を使用して COBOL メインプログラムを呼び出すための JCL を示しています。

```
//GOSTEP EXEC PGM=COBMAIN,PARM='POSIX(ON) / 20190901'
```

第 20 章 COBOL コンパイラとランタイムオプション

このセクションでは、COBOL コンパイラおよび COBOL ランタイム環境で利用できるオプションについて詳しく説明します。

COBOL コンパイラ・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・オプションを使用する方法と、コンパイラ指示ステートメント (コンパイラ指示) を使用する方法とがあります。

コンパイラ・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 47. コンパイラ・オプション			
プログラムの局面	コンパイラ・オプション	デフォルト	オプションの省略形
ソース言語	379 ページの『 APOST/QUOTE 』	QUOTE	APOST Q
	381 ページの『 ARITH 』	ARITH (COMPAT)	AR (C E)
	384 ページの『 CICS 』	NOCICS	なし
	385 ページの『 CODEPAGE 』	CODEPAGE (1140)	CP (<i>ccsid</i>)
	390 ページの『 CURRENCY 』	NOCURRENCY	CURR NOCURR
	392 ページの『 DBCS 』	DBCS	なし
	421 ページの『 NSYMBOL 』	NSYMBOL (NATIONAL)	NS (DBCS NAT)
	422 ページの『 番号 』	NONUMBER	NUM NONUM
	434 ページの『 QUALIFY 』	QUALIFY (COMPAT)	QUA (C E)
	439 ページの『 SEQUENCE 』	SEQUENCE	SEQ NOSEQ
	443 ページの『 SQL 』	NOSQL	なし
	444 ページの『 SQLCCSID 』	SQLCCSID	SQLC NOSQLC
	445 ページの『 SQLIMS 』	NOSQLIMS	なし
	448 ページの『 SUPPRESS 』	SUPPRESS	SUPP
	461 ページの『 WORD 』	NOWORD	WD NOWD
	462 ページの『 XMLPARSE 』	XMLPARSE (XMLSS)	XP (X) XP (C)
日付処理	408 ページの『 INTDATE 』	INTDATE (ANSI)	なし

表 47. 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
マップおよびリスト	コンディション	CONDCOMP(NOSKIPSRC)	CC(SKIPS NOSKIPS)
	413 ページの『 LANGUAGE 』	LANGUAGE(ENGLISH)	LANG(EN UE JA JP)
	414 ページの『 LINECOUNT 』	LINECOUNT(60)	LC
	414 ページの『 LIST 』	NOLIST	なし
	418 ページの『 MAP 』	NOMAP	なし
	427 ページの『 OFFSET 』	N0OFFSET	OFF N0OFF
	442 ページの『 SOURCE 』	SOURCE	S NOS
	443 ページの『 SPACE 』	SPACE(1)	なし
	448 ページの『 TERMINAL 』	NOTERMINAL	TERM NOTERM
	458 ページの『 VBREF 』	NOVBREF	なし
	463 ページの『 外部参照 』	XREF(FULL)	X NOX
オブジェクト・デックの生成	388 ページの『 COMPILE 』	NOCOMPILE(S)	C NOC
	390 ページの『 COPYRIGHT 』	NOCOPYRIGHT	CPYR NOCPYR
	393 ページの『 DECK 』	NODECK	D NOD
	421 ページの『 名前 』	NAME のみが指定されている場合 NONAME、または NAME(NOALIAS)	なし
	427 ページの『 OBJECT 』	OBJECT	OBJ N0OBJ
	432 ページの『 PGMNAME 』	PGMNAME(COMPAT)	PGMN(CO LU LM)
	440 ページの『 サービス 』	NOSERVICE	SERV NOSERV

表 47. 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
オブジェクト・コード制御	378 ページの『ADV』	ADV	なし
	378 ページの『AFP』	AFP (NOVOLATILE)	なし
	379 ページの『ARCH』	ARCH (10)	なし
	382 ページの『AWO』	NOAWO	なし
	383 ページの『BLOCK0』	NOBLOCK0	なし
	393 ページの『DEFINE』	NODEFINE	DEF NODEF
	395 ページの『DISPSIGN』	DISPSIGN (COMPAT)	DS (S C)
	396 ページの『DLL』	NODLL	なし
	401 ページの『EXPORTALL』	NOEXPORTALL	EXP NOEXP
	402 ページの『FASTSRT』	NOFASTSRT	FSRT NOFSRT
	405 ページの『HGPR』	HGPR (PRESERVE)	なし
	407 ページの『INITIAL』	NOINITIAL	なし
	407 ページの『INLINE』	INLINE	INL NOINL
	408 ページの『INVDATA』	NOINVDATA	NOINV INV (FNC NOFNC, CS NOCS)
	412 ページの『JAVAIOP』	NOJAVAIOP	JIOP NOJIOP
	415 ページの『LP』	LP (32)	None
	419 ページの『MAXPCF』	MAXPCF (100000)	なし
	422 ページの『NUMCHECK』	NONUMCHECK	NC NONC
	426 ページの『NUMPROC』	NUMPROC (NOPFD)	なし
	429 ページの『OPTIMIZE』	OPTIMIZE (0)	OPT (n)
	430 ページの『OUTDD』	OUTDD (SYSOUT)	OUT
	431 ページの『PARMCHECK』	NOPARMCHECK	PC NOPC
	454 ページの『TRUNC』	TRUNC (STD)	なし
	456 ページの『TUNE』	ARCH が指定されていない場合は TUNE (10)。ARCH が指定されている場合、デフォルトの TUNE レベルは ARCH レベルと一致している必要があります。	なし
	458 ページの『VLR』	VLR (STD)	VLR (C S)
	464 ページの『ZONECHECK』	NOZONECHECK	NOZC ZC (MSG) ZC (ABD)
	465 ページの『ZONEDATA』	ZONEDATA (PFD)	ZD (PFD) ZD (MIG) ZD (NOPFD)
	468 ページの『ZWB』	ZWB	なし

表 47. 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
仮想記憶域の使用量	384 ページの『BUFSIZE』	4096	BUF
	391 ページの『データ』	DATA(31)	なし
	398 ページの『DYNAM』	NODYNAM	DYN NODYN
	435 ページの『RENT』	RENT	なし
	436 ページの『RMODE』	AUTO	なし
	447 ページの『STGOPT』	NOSTGOPT	SO NOSO
デバッグと診断	394 ページの『DIAGTRUNC』	NODIAGTRUNC	DTR NODTR
	398 ページの『DUMP』	NODUMP	DU NODU
	402 ページの『FLAG』	FLAG(I,I)	F NOF
	403 ページの『FLAGSTD』	NOFLAGSTD	なし
	405 ページの『INITCHECK』	NOINITCHECK	IC NOIC
	416 ページの『LSACHECK』	NOLSACHECK	LSAC NOLSAC
	437 ページの『RULES』	NORULES	RULES(ENDP,EVENP,LXPRF,SLCKB,OOM) RULES(NOENDP,NOEVENP,NOLXPRF,NOSLCKB,NOOOM,NOUNRA NOUNRS)
	446 ページの『SSRANGE』	NOSSRANGE	SSR(ZLEN NOZLEN,MSG ABD) NOSSR
	448 ページの『TEST』	NOTEST	なし
その他	378 ページの『ADATA』	NOADATA	なし
	389 ページの『COPYLOC』	NOCOPYLOC	CPLC
	399 ページの『出口』	NOEXIT	NOEX EX(INX NOINX,LIBX NOLIBX,PRTX NOPRTX,ADX NOADX,MSGX NOMSGX)
	419 ページの『MDECK』	NOMDECK	NOMD MD MD(C NOC)
	428 ページの『OPTFILE』	なし	なし
	440 ページの『SMARTBIN』	LP(32)が有効な場合は SMARTBIN。LP(64)が有効な場合、SMARTBIN はサポートされません。	なし
	453 ページの『THREAD』	NOTHREAD	なし
	461 ページの『VSAMOPENFS』	VSAMOPENFS(COMPAT)	VS(C S)

インストール先デフォルト: コンパイラーがインストールされたときにセットアップされたデフォルト・コンパイラー・オプションは、そのオプションをオーバーライドしない限り、プログラムでは有効になります。(インストール先によっては、特定のコンパイラー・オプションは、オーバーライドできないように固

定されます。デフォルト・オプションについて問題がある場合には、システム管理者に連絡してください。) デフォルト・オプションを判別するには、コンパイラー・オプションを指定せずにテスト・コンパイルを実行します。出力リストに、そのサイトで有効なデフォルト・オプションが示されます。

オーバーライド不可能なオプション: インストール先によっては、特定のコンパイラー・オプションは、オーバーライドできないように固定されます。そうしたオプションについて問題がある場合には、システム管理者に連絡してください。

オプション指定: コンパイラー・オプションおよびサブオプションは大/小文字を区別しません。

パフォーマンスの考慮事項。 AFP、ARCH、ARITH、AWO、BLOCK0、DYNAM、FASTSRT、HGPR、MAXPCF、NUMCHECK、NUMPROC、OPTIMIZE、PARMCHCK、RENT、SQLCCSID、SSRANGE、STGOPT、TEST、THREAD、TRUNC、TUNE、ZONECHECK、および INVDATA コンパイラー・オプションは、ランタイム・パフォーマンスに影響を与える可能性があります。

関連タスク

[279 ページの『第 15 章 z/OS のもとでのコンパイル』](#)

[287 ページの『TSO のもとでのコンパイル』](#)

[311 ページの『第 16 章 z/OS ユニックスのもとでのコンパイル』](#)

[837 ページの『第 50 章 プログラムのチューニング』](#)

関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

[481 ページの『第 21 章 コンパイラー指示ステートメント』](#)

[375 ページの『85 COBOL 標準に準拠するオプション設定』](#)

[845 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』](#)

85 COBOL 標準に準拠するオプション設定

85 COBOL 標準に準拠するためには、コンパイラー・オプションとランタイム・オプションが必要です。

以下のコンパイラー・オプションが必要です。

- ADV
- DYNAM
- NAME (ALIAS) または NAME (NOALIAS)
- NOBLOCK0
- NOCICS
- NODLL
- NOEXPORTALL
- NOFASTSRT
- NOTHREAD
- NOWORD
- NUMPROC (NOPFD)
- PGMNAME (COMPAT) または PGMNAME (LONGUPPER)
- QUALIFY (COMPAT)
- QUOTE
- TRUNC (STD)
- VLR (STANDARD)
- VSAMOPENFS (SUCC)
- ZWB

あなたは FLAGSTD コンパイラオプションは、次のような不適合要素をフラグ付けします。IBM 拡張機能。

以下のランタイム・オプションが必要です。

- AIXBLD
- CBLQDA(ON)
- TRAP(ON)

関連参照

言語環境プログラミング・リファレンス

矛盾するコンパイラー・オプション

Enterprise COBOL 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です。

競合するオプションが階層内の同じレベルで指定されている場合 (PROCESS または CBL ステートメントで、DECK と NODECK の両方を指定する場合など)、最後に指定されたオプションが有効になります。

相互に排他的なコンパイラー・オプションを同じレベルに指定すると、コンパイラーはエラー・メッセージを生成し、オプションの一方を対立しない値に強制します。例えば、PROCESS ステートメントで、OFFSET と LIST の両方を任意の順序で指定すると、OFFSET が有効になり、LIST は無視されます。

しかし、高レベルの優先順位でコーディング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、JCL ステートメントに OFFSET をコーディングし、PROCESS ステートメントに LIST をコーディングした場合、LIST が有効になります。これは、PROCESS ステートメントにコーディングされたオプションと、PROCESS ステートメントにコーディングされたオプションによって強制的にオンにされたオプションの方が優先順位が高いためです。

表 48. 相互に排他的なコンパイラー・オプション		
指定済み	無視される ¹	強制的にオンにされる ¹
AFP(VOLATILE)	LP(64)	LP(32)
CICS	DYNAM	NODYNAM
	LP(64)	LP(32)
	NORENT	RENT
DATA(24)	LP(64)	LP(32)
DLL	DYNAM	NODYNAM
	NORENT	RENT
EXPORTALL	NODLL	DLL
	DYNAM	NODYNAM
	NORENT	RENT
HGPR(NOPRESERVE)	LP(64)	LP(32)

表 48. 相互に排他的なコンパイラ・オプション (続き)

指定済み	無視される ¹	強制的にオンにされる ¹
LP (64)	AFP (VOLATILE)	AFP (NOVOLATILE)
	CICS	NOCICS
	DATA (24)	DATA (31)
	HGPR (NOPRESERVE)	HGPR (PRESERVE)
	NODLL	DLL
	NORENT	RENT
	RMODE (24)	RMODE (ANY)
	SMARTBIN	NOSMARTBIN
	SQLIMS	NOSQLIMS
	THREAD	NOTHREAD
NORENT	RMODE (ANY)	RMODE (24)
	LP (64)	LP (32)
NSYMBOL (NATIONAL)	NODBCS	DBCS
NUMCHECK (ZON (LAX))	NUMPROC (PFD)	NUMPROC (NOPFD)
NUMPROC (PFD)	NUMCHECK (ZON (LAX))	NUMCHECK (ZON (STRICT))
OBJECT	DECK	NODECK
OFFSET	LIST	NOLIST
PGMNAME (LM LU)	NAME	NONAME
RMODE (24)	LP (64)	LP (32)
SMARTBIN	LP (64)	LP (32)
SOURCE (HEX)	NUMBER	NONUMBER
	SEQUENCE	NOSEQUENCE
SQLIMS	LP (64)	LP (32)
TEST	NNOBJECT と NODECK	OBJECT と NODECK
THREAD	INITIAL	NOINITIAL
	NORENT	RENT
	LP (64)	LP (32)
WORD	FLAGSTD	NOFLAGSTD
1. 固定されているインストール先デフォルト・オプションと対立する場合を除きます。		

関連タスク

299 ページの『[z/OS のもとでのコンパイラ・オプションの指定](#)』

304 ページの『[バッチ・コンパイルでのコンパイラ・オプションの指定](#)』

312 ページの『[z/OS ユニックス のもとでのコンパイラ・オプションの指定](#)』

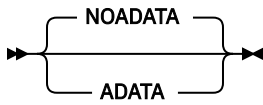
関連参照

428 ページの『[OPTFILE](#)』

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



デフォルトは、NOADATA

省略形: なし

z/OS では、SYSADATA ファイルは DD 名 SYSADATA に書き込まれます。

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。

- JCL の PARM パラメーター
- として cob2 コマンドオプション
- インストール先デフォルト値として
- COBOPT 環境変数

関連参照

[311 ページの『z/OS ユニックスのもとでの環境変数の設定』](#)

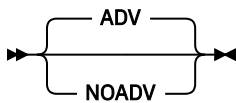
[315 ページの『cob2 の構文およびオプション』](#)

[909 ページの『COBOL SYSADATA ファイルの内容』](#)

ADV

ADV が意味を持つのは、ソース・コードで WRITE . . . ADVANCING を使用する場合があります。ADV が有効になっていると、コンパイラーは、印刷制御文字を入れるための 1 バイトをレコード長に追加します。

ADV オプションの構文



デフォルトは、ADV

省略形: なし

既に印刷制御文字のための 1 バイトを組み込むようにレコード長を調整している場合は、NOADV を使用してください。

AFP

AFP オプションは、IBM z/Architecture プロセッサに装備されている追加浮動小数点 (AFP) レジスターをコンパイラーで使用方法を制御します。

AFP オプションの構文

➡ AFP(NOVOLATILE) ➡

デフォルトは、AFP(NOVOLATILE)

省略形: なし

Enterprise COBOL コンパイラーは、IBM z/Architecture プロセッサに装備されている 16 個の浮動小数点レジスター (FPR) の完全な補数を使用するコードを生成します。これらの FPR には以下があります。

- オリジナル FPR。0、2、4、および 6 の番号が付けられています。
- AFP レジスター。1、3、5、7、および 8 から 15 の番号が付けられています。

AFP(VOLATILE)

AFP(VOLATILE) を指定すると、AFP レジスター 8 から 15 は揮発性に見なされます。これは、呼び出されたサブプログラムによって変更される可能性があることを意味します。このため、COBOL コンパイラーはこれらのレジスターの値を保護するために、追加コードを生成します。

AFP(NOVOLATILE)

AFP(NOVOLATILE) を指定すると、AFP レジスター 8 から 15 は不揮発性に見なされます。これは、呼び出されたどのサブプログラムも、これらのレジスターを変更しないこと、つまりレジスターが保持されることが既知であることを意味します。このため、コンパイラーは浮動小数点演算が含まれるプログラムに対して、より効率的なコード・シーケンスを生成できます。これは通常の z/OS アーキテクチャーの規則です。

AMODE 64 の考慮事項: とき LP(64) コンパイラオプションが有効になっている場合、AFP(VOLATILE) オプションはサポートされていません。AFP(VOLATILE) オプションを明示的に指定すると、警告メッセージが発行され、設定は破棄されます。

APOST/QUOTE

APOST は、形象定数 [ALL] QUOTE または [ALL] QUOTES で 1 つ以上のアポストロフィ (') 文字を表したい場合に使用します。QUOTE は、形象定数 [ALL] QUOTE または [ALL] QUOTES で 1 つ以上の引用符 (") 文字を表したい場合に使用します。

APOST/QUOTE オプションの構文

➡ { QUOTE
APOST } ➡

デフォルトは、QUOTE

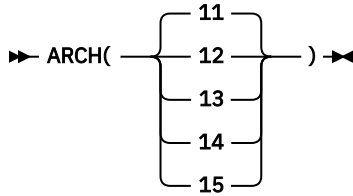
略語は次のとおりです。Q (のために QUOTE) | なし (のために APOST)

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、引用符 (") またはアポストロフィ (') 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。

ARCH

ARCH オプションは、実行可能プログラム命令が生成されるマシン・アーキテクチャーを指定します。

ARCH オプションの構文



デフォルトは、ARCH(11)

省略形: なし

より高い ARCH レベルを指定すると、コンパイラはより新しいより高速な命令を使用するコードを生成します。アプリケーションは、ARCH オプションで指定したレベルより低いアーキテクチャー・レベルのプロセッサで実行すると、異常終了する可能性があります。災害復旧 (DR) マシンを含む、アプリケーションが実行される最も低いマシン・アーキテクチャーに一致する ARCH レベルを使用します。

現在サポートされているアーキテクチャー・レベルおよびモデルのグループは以下のとおりです。

11

IBM z/Architecture モードの 2964-xxx (IBM z13[®]) モデルおよび 2965-xxx (IBM z13s[®]) モデルで使用可能な命令を使用するコードを生成します。

具体的には、これらの ARCH(11) マシンとその後継マシンは、以下の機能によってサポートされる命令を追加します。

- パック 10 進数データ項目と 10 進浮動小数点中間結果データ項目間のより効率的な変換を可能にする、拡張 10 進浮動小数点機能 (周囲条件が最適で、最適化レベルが 0 より大きい場合)。
- 一部の INSPECT REPLACING および INSPECT TALLYING ステートメントでのベクトル拡張機能 (SIMD) 命令の活用。

ベクトル拡張機能 (SIMD) 命令を使用するには、z/OS 2.2 または z/OS 2.1 (APAR 用 PTF 適用済み) OA43803 および PI12412 がインストールされているマシン上でコードを実行する必要があります。

12

IBM z/Architecture モードの 3906-xxx (IBM z14) モデルおよび 3907-xxx (IBM z14 ZR1) モデルで使用可能な命令を使用するコードを生成します。

具体的には、これらの ARCH(12) マシンとその後継マシンは、ベクトル・パック 10 進数機能でサポートされる命令を追加します。この機能は、中間結果をメモリーではなくベクトル・レジスターに保管することで、パック 10 進数およびゾーン 10 進数の計算を高速化します。

※13

IBM z/Architecture モードの 8561-xxx (IBM z15) モデルおよび 8562-xxx (IBM z15 T02) モデルで使用可能な命令を使用するコードを生成します。

具体的には、これらの ARCH(13) マシンとその後継マシンは、以下の機能によってサポートされる命令を追加します。

- ベクトル・パック 10 進数拡張機能
- ベクトル拡張機能 2
- その他の命令拡張機能 3
- 調整済みベクトル・ロード/保管のヒント

14

3931-xxx() および 3932-xxx() で使用可能な命令を使用するコードを生成します。IBMz16 と 3932-xxx (IBMz16 A02) モデルで IBM z/Architecture モード。

具体的には、この ARCH(14) マシンとその後継マシンは、新しいベクトル・パック 10 進数拡張機能 2 でサポートされる命令を追加します。この新しい機能により、以下のタイプのステートメントを 1 つ以上含む COBOL プログラムのパフォーマンスが向上します。

- 指数が 1 桁以上の小数で宣言されているパック 10 進数またはゾーン 10 進数のデータ項目に対する指数演算
- 10 進数データ項目と浮動小数点データ項目が混在する算術ステートメント
- 数字編集データ項目を使用するステートメント

15

9175-xxxx モデル (IBMz17) モデルの IBM z/Architecture モード。

具体的には、この ARCH(15) マシンとその後継機は、新しいベクター・エンハンスメント・ファシリティ 3 とベクター・パックド・デシマル・エンハンスメント・ファシリティ 3 でサポートされる命令を追加する。ベクトル拡張機能 3 は、COMP/COMP-5、64 ビット以上を必要とするバイナリ・データ型を含む数値計算を含む COBOL プログラムのパフォーマンスを向上させる。Vector Packed Decimal Enhancement Facility 3 は、NUMCHECK オプションでコンパイルされた COBOL プログラムのパフォーマンスを向上させる。

注：上位の ARCH レベルには、下位の ARCH レベルの機能が含まれています。例えば、ARCH(15)には、下位 ARCH レベルのすべての機能が含まれます。

これらの機能について詳しくは、「*IBM z/Architecture Principles of Operation*」を参照してください。

関連参照

456 ページの『TUNE』

ARITH

ARITH は、数値項目についてコーディングできる桁の最大数、および固定小数点の中間結果で使用する桁の数に影響します。

ARITH オプションの構文

➡ ARITH(COMPAT
┌───┐
EXTEND) ➡

デフォルトは、ARITH(COMPAT)

省略形: AR(C | E)

ARITH(EXTEND) を指定すると、以下のようになります。

- 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最大桁数が 18 から 31 に引き上げられます。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、NUMVAL-C、および NUMVAL-F 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、以下のようになります。

- 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。
- 引数の最大桁数 NUMVAL、NUMVAL-C そして NUMVAL-F18 歳です。

注: NUMVAL は長精度浮動小数点計算を使用して、引数内の桁を数値出力に変換します。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)。したがって、NUMVAL、NUMVAL-C、または NUMVAL-F の引数が 15 桁を超える場合には、ARITH(EXTEND) コンパイラ・オプションを指定して、結果が入力引数の値を正確に表現できるようにすることをお勧めします。

- 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。

ARITH(EXTEND)例

この例では、ARITH(EXTEND) :

```
CBL ARITH(EXTEND)
IDENTIFICATION DIVISION.
PROGRAM-ID.    SAMPLE
DATE-COMPILED.
ENVIRONMENT DIVISION.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.

01 WS-CHAR          PIC X(30).
01 WS-NUM           PIC 9(30).
*> Note this will get a syntax error with ARITH(COMPAT)

PROCEDURE DIVISION.

    MOVE '651310406401912040024657817701' TO WS-CHAR
    MOVE WS-CHAR TO WS-NUM

    DISPLAY 'THE CHAR VALUE: ' WS-CHAR
    DISPLAY 'THE NUM  VALUE: ' WS-NUM

    GOBACK.
```

この例の出力は次のとおりです。

```
THE CHAR VALUE: 651310406401912040024657817701
THE NUM  VALUE: 651310406401912040024657817701
```

関連概念

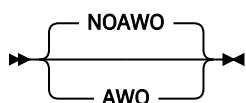
[859 ページの『中間結果および算術精度』](#)

[58 ページの『変換および精度』](#)

AWO

指定する場合 AWO、暗黙の APPLY WRITE-ONLY 条項は全員に有効です。サンプログラム内のファイルブロック化可変長レコードが必要です。

AWO オプションの構文



デフォルトは、NOAWO

省略形: なし

関連タスク

11 ページの『バッファーおよび装置スペースの最適化』

関連参照

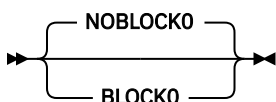
383 ページの『BLOCK0』

APPLY WRITE-ONLY 節 (*Enterprise COBOL for z/OS* 言語解説書)

BLOCK0

使用 BLOCK0QSAM ファイルのコンパイラのデフォルトを非ブロックからブロックに変更します (BLOCK CONTAINS 0 指定された場合)、出力ファイルに対してシステムによって決定されるブロック化の利点を得られます。

BLOCK0 オプションの構文



デフォルトは、NOBLOCK0

省略形: なし

BLOCK0 を指定すると、プログラム内で次の 3 つの基準を満たす各ファイルに対して、暗黙の BLOCK CONTAINS 0 節をアクティブにします。

- の FILE-CONTROL 段落は、ORGANIZATION SEQUENTIAL または省略する ORGANIZATION 句。
- の FD エントリは指定されていません RECORDING MODE U。
- の FD エントリは指定しません BLOCK CONTAINS 句。

結果として BLOCK CONTAINS 0 節が有効になるファイルのブロック化因数は、データ定義またはデータ・セット特性から実行時に判別されます。

APPLY WRITE-ONLY 節および AWO コンパイラ・オプションと BLOCK0 の相互作用:

- もし NOBLOCK0 が有効であり、上記の 3 つの基準を満たすファイルのファイル記述は、APPLY WRITE-ONLY コンパイラはエラーメッセージを発行します。APPLY WRITE-ONLY ブロックされたファイルにのみ適用されます。しかし、もし BLOCK0 が有効になっている場合、ファイルはブロックされ、APPLY WRITE-ONLY したがって、この条項は受け入れられます。
- AWO は、ブロック化可変長レコードが入った任意の QSAM ファイルに適用されます。BLOCK0 が有効な場合、結果として、NOBLOCK0 が有効な場合より多くのファイルがブロック化される可能性があるため、AWO が適用されるファイルが多くなることがあります。

既存のプログラムに BLOCK0 を指定すると、動作が変わり、INPUT としてオープンされたファイルに関して望まない結果が生成されることがあります。例:

- ブロック・サイズを判別できないファイルについて、OPEN INPUT ステートメントが失敗します。
- INPUT としてオープンされたファイルのゼロ以外の FILE STATUS コードを処理した後に続行したプログラムが、その後、そのファイルに入出力ステートメントを実行したときに、異常終了する場合があります。

これらの理由のため、BLOCK0 でコンパイルした後は、プログラムでの影響を調べてテストしてください。

ブロッキングに関する推奨事項については、*Enterprise COBOL for z/OS* 移行ガイド (移行に関する情報では CMPR2 に NOCMR2)。

関連タスク

11 ページの『バッファーおよび装置スペースの最適化』

184 ページの『ブロック・サイズの設定』

関連参照

382 ページの『AWO』

APPLY WRITE-ONLY 節 (*Enterprise COBOL for z/OS* 言語解説書)

BLOCK CONTAINS 節 (*Enterprise COBOL for z/OS* 言語解説書)

Enterprise COBOL for z/OS 移行ガイド

(JCL の DCB= パラメーターの推奨)

BUFSIZE

BUFSIZE を使用して、各コンパイラー作業データ・セットのバッファーに、ある量の主ストレージを割り振ります。通常、バッファー・サイズを大きくすると、コンパイラーのパフォーマンスが向上します。

BUFSIZE オプションの構文

➡ BUFSIZE(nnnnn nnnK) ➡

デフォルト: 4096

省略形: BUF

nnnnn は 10 進数で、少なくとも 256 でなければなりません。

nnnK は、1 KB の増分で 10 進数を指定します。ここで、1 KB = 1024 バイトです。

BUFSIZE は、使用される装置のトラック容量を超えることはできません。また、データ管理サービスによって許可される最大容量を超えることもできません。

注: BUFSIZE コンパイラー・オプションは、一連のプログラムの最初のプログラムに対してのみ指定できます (バッチ・コンパイル)。シーケンス内の最初のプログラム以外のプログラムに対して CBL または PROCESS ステートメントで BUFSIZE を指定すると、コンパイラー診断メッセージを受け取ります。

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。使用する必要があります CICS COBOL ソースプログラムに EXEC CICS または EXEC DLI ステートメントとプログラムは別の CICS 翻訳者。

CICS オプションの構文

➡ { NOCICS | CICS ("CICS-suboption-string") } ➡

デフォルトは、NOCICS

省略形: なし

使用 CICS コンパイルのみのオプション CICS プログラム。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。

指定する場合は NOCICS オプション、任意 CICS ソース プログラム内で見つかったステートメントは診断され、破棄されます。

引用符またはアポストロフィ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。

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。CICS サブオプションは出現順に連結されます。

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL='CICS("string1")'
//COBOL.SYSIN DD *
    CBL CICS('string2')
    CBL CICS("string3")
    IDENTIFICATION DIVISION.
    PROGRAM-ID. DRIVER1.
    . . .
```

コンパイラーは、以下のサブオプション・ストリングを組み込みの 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・ストリングのサイズは 4KB に制限されます。

AMODE 64 の考慮事項: とき LP(64) コンパイラオプションが有効になっている場合、CICS オプションはサポートされていません。CICS オプションを明示的に指定すると、警告メッセージが発行され、設定は破棄されます。

注: CICS コンパイラー・オプションは、一連のプログラムの最初のプログラムに対してのみ指定できます (バッチ・コンパイル)。シーケンス内の最初のプログラム以外のプログラムに対して CBL または PROCESS ステートメントで CICS を指定すると、コンパイラー診断メッセージを受け取ります。

関連概念

[551 ページの『組み込みの CICS 変換プログラム』](#)

関連タスク

[549 ページの『CICS オプションを使用したコンパイル』](#)

[550 ページの『CICS サブオプションの分離』](#)

[翻訳者オプションの定義 \(CICS アプリケーションの開発\)](#)

関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

CODEPAGE

CODEPAGE は、文字エンコードに依存する COBOL 操作をコンパイル時および実行時に処理するために、EBCDIC コード・ページのコード化文字セット ID (CCSID) を指定する場合に使用します。

CODEPAGE オプションの構文

➡ CODEPAGE(*ccsid*) ➡

デフォルトは、CODEPAGE(1140)

省略形: CP(*ccsid*)

ccsid は、EBCDIC コード・ページに対する有効な CCSID を表す整数でなければなりません。

デフォルトの CCSID 1140 は CCSID 37 (COM EUROPE EBCDIC) と同等ですが、さらにユーロ記号を含んでいます。

ccsid では次のエンコードを指定します。

- 英数字、国別、UTF-8、COBOL ソースプログラム内の DBCS リテラル
- 実行時の英数字および DBCS データ項目のコンテンツのデフォルト・エンコード

- DBCS ユーザー定義語のエンコードは、XML GENERATE XML 要素と属性名を作成するためのステートメント
- XML 文書のデフォルトのエンコーディングは、XML GENERATE 文書の受信データ項目が英数字の場合のステートメント
- 文書が XML PARSE ステートメントによって処理される際に、英数字データ項目の XML 文書で想定されるデフォルト・エンコード

CODEPAGE *ccsid* は、コード・ページに依存する操作をコンパイル時または実行時に実行し、デフォルト・コード・ページをオーバーライドする明示的な CCSID が指定されていない場合に使用されます。このような操作には次のものがあります。

- リテラル値の Unicode への変換
- 英数字データと国別 (Unicode) データ間の変換そして UTF-8 (Unicode) データ移動操作、比較、または組み込み関数の一部として DISPLAY-OF そして NATIONAL-OF
- INVOKE ステートメントなどのオブジェクト指向言語、またはクラス定義およびメソッド定義
- XML 構文解析
- XML 生成
- 実行時の XML 生成の一部としての DBCS 名の処理
- SQLCCSID オプションが有効である場合の SQL スtring・ホスト変数の処理
- EXEC SQL ステートメントのソース・コードの処理
- EXEC SQLIMS ステートメントのソース・コードの処理

ただし、COBOL ソース・プログラム内の次の項目のエンコードは、CODEPAGE コンパイラー・オプションの影響を受けません。

- USAGE NATIONAL を持つデータ項目

このような項目は常に、ビッグ・エンディアン形式の UTF-16、CCSID 1200 でエンコードされます。

- USAGE UTF-8 を持つデータ項目

これらの項目は常に UTF-8 形式、CCSID 1208。

- 基本 COBOL 文字セットの文字 (文字に関する下記の関連参照にある該当文字の表を参照)

基本 COBOL 文字、デフォルト通貨記号 (\$)、引用符 (")、および小文字ローマ字のエンコードは、EBCDIC コード・ページによって異なりますが、コンパイラーは常に、EBCDIC コード・ページ 1140 エンコードを使用してこれらの文字を解釈します。特に、デフォルトの通貨記号は常に値を持つ文字です X'5B' (変更されない限り CURRENCY コンパイラオプションまたは CURRENCY SIGN 条項の SPECIAL-NAMES 段落) であり、引用符は常に値を持つ文字です X'7F'。

例えば次に示すように、一部の COBOL 操作では、明示的なエンコードの指定を使用することによって、CODEPAGE *ccsid* をオーバーライドすることができます。

- DISPLAY-OF そして NATIONAL-OF2 番目の引数としてコードページを指定する組み込み関数
- XML PARSE 指定する文 WITH ENCODING フレーズ
- XML GENERATE 指定する文 WITH ENCODING フレーズ

さらに、CURRENCY コンパイラー・オプション、または SPECIAL-NAMES 段落の CURRENCY SIGN 節を使用して次のものをオーバーライドすることができます。

- ソース・プログラム内の数字編集データ項目の PICTURE 文字ストリングで使用されるデフォルト通貨記号
- 実行時に数字編集データ項目のコンテンツで使用される通貨記号

DBCS コード・ページ:

プログラムに次のいずれかの項目が含まれている場合には、CODEPAGE オプションを使用して、ccsid を下表に示されている EBCDIC マルチバイト文字セット (MBCS) CCSID のいずれかに設定し、COBOL プログラムをコンパイルします。

- DBCS 文字から成るユーザー定義語
- DBCS (USAGE DISPLAY-1) データ項目
- DBCS リテラル

下表の CCSID はすべて、SBCS と DBCS コード化文字セットの組み合わせを示す混合コード・ページを指定します。また、これらは Db2 が混合データでサポートする CCSID です。

表 49. EBCDIC マルチバイト・コード化文字セット ID			
各国語	MBCS CCSID	SBCS CCSID コンポーネント	DBCS CCSID コンポーネント
日本語 (カタカナ-漢字)	930	290	300
日本語 (カタカナ-漢字とユーロ)	1390	8482	16684
日本語 (カタカナ-漢字)	5026	290	4396
日本語 (ローマ字-漢字)	939	1027	300
日本語 (ローマ字-漢字とユーロ)	1399	5123	16684
日本語 (ローマ字-漢字)	5035	1027	4396
韓国語	933	833	834
韓国語	1364	13121	4930
中国語 (簡体字)	935	836	837
中国語 (簡体字)	1388	13124	4933
中国語 (繁体字)	937	28709	835

注: TEST オプションを指定する場合は、CODEPAGE オプションを、COBOL ソース・プログラムで使用する CCSID に設定する必要があります。特に、DBCS リテラルまたは DBCS ユーザー定義語で日本語文字を使用するプログラムの場合、CODEPAGE オプションを日本語コード・ページ CCSID に設定してコンパイルしてください。

Db2 ユーザーに関する注意: IBM では、Db2 DSNHDECP 値および/またはプリコンパイラー CCSID オプション値と同じ COBOL CCSID 値を使用することをお勧めしています。

関連概念

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[70 ページの『通貨記号の使用』](#)

[687 ページの『第 38 章 XML 入力の処理』](#)

[727 ページの『第 39 章 XML 出力の生成』](#)

関連参照

[390 ページの『CURRENCY』](#)

[444 ページの『SQLCCSID』](#)

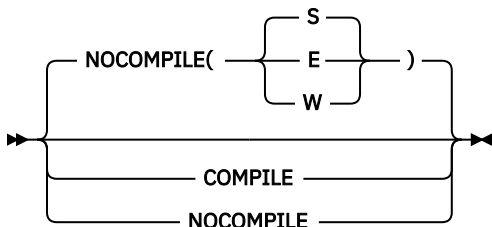
[448 ページの『TEST』](#)

[704 ページの『XML 文書のエンコード方式』](#)
文字 (Enterprise COBOL for z/OS 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。

COMPILE オプションの構文



デフォルトは、NOCOMPILE(S)

省略形は C | NOC

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・オプション (DECK、LIST、OBJECT、OFFSET、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

490 ページの『コーディング・エラーの検出』

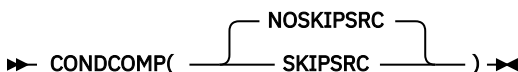
関連参照

307 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』

コンディション

CONDCOMP は条件付きコンパイル ディレクティブの動作に影響し、条件付きコードがリストにどのように表示されるか制御します。

CONDCOMP オプション構文



デフォルトは、CONDCOMP(NOSKIP SRC)

略語は次のとおりです。CC(SKIP | NOSKIP)

条件付き COMP(NOSKIP SRC)

もし CONDCOMP(NOSKIP SRC) が有効な場合、条件付きコンパイル ディレクティブによって区切られたすべてのソース行とコメントがリストに表示されます。偽のブランチのソース行 IF そして EVALUATE ディレクティブはコメントとして表示されます。

条件付き COMP(SKIPSRC)

もし CONDCOMP(SKIPSRC)有効になっている場合、条件付きコンパイルディレクティブの偽の分岐内のソース行とコメント、IF そして EVALUATE ディレクティブはリストに表示されません。さらに、コンパイルされたソース行とコメントも MDECK 出力ファイルと SYSMDECK から省略されます。

関連参照

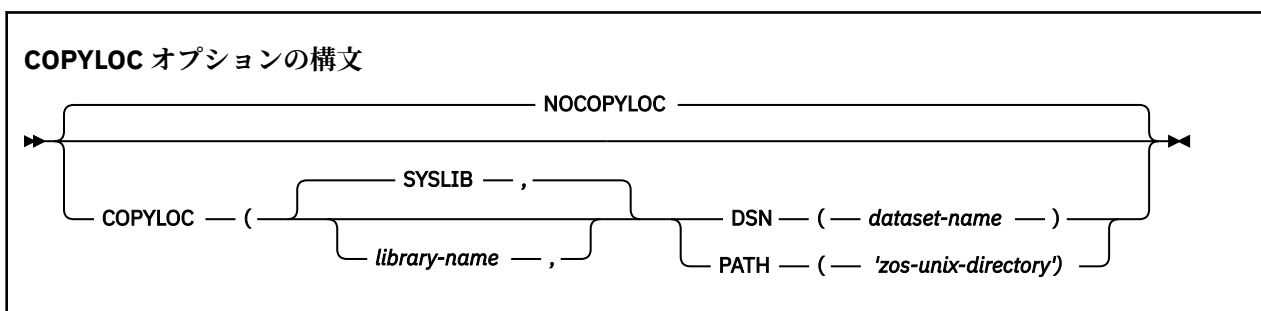
例: 条件付きコンパイル出力

条件付きコンパイル (Enterprise COBOL for z/OS 言語解説書)

COPYLOC

COPYLOC コンパイラー・オプションを使用して、PDSE (または PDS) データ・セットまたは z/OS ユニックスディレクトリーを、ライブラリー・フェーズ中にコピー・メンバーを検索する追加ロケーションとして追加します。COPYLOC オプションで指定されたロケーションは、コピー・メンバーを検索するために、ロケーションの順序の末尾に追加されます。詳しくは、「Enterprise COBOL for z/OS 言語解説書」の『COPY メンバー探索順序』を参照してください。

COPYLOC は、APAR PI91584 の PTF がインストールされた Enterprise COBOL V6.2 で導入されました。



デフォルトは、NOCOPYLOC

省略形は CPLC | NOCPLC

library-name

コピー場所が関連付けられるライブラリー名。ライブラリー名が指定されていない場合、デフォルトは SYSLIB です。これは、明示的なライブラリー名を含まない COPY ステートメントで想定されるライブラリー名です。

dataset-name

ライブラリーライブラリー名を参照する COPY ステートメントを処理するときに、コンパイラーがコピー・メンバーを検索する必要がある PDS または PDSE データ・セットの名前。

zos-unix ディレクトリ

ライブラリーライブラリー名を参照する COPY ステートメントを処理するときに、コンパイラーがコピー・メンバーを検索する必要がある z/OS ユニックスディレクトリーの名前。指定するパスは 64 文字を超えてはなりません。z/OS ユニックスの標準である小文字パスを指定するには、パスを引用符で囲む必要があります。囲まないと、パス名は大文字に変換されます。

COPYLOC オプションの複数インスタンスがサポートされます。指定可能な z/OS ユニックスディレクトリー数には制限はありませんが、探索に指定可能なデータ・セットは 256 に制限されています。コピー・ロケーションは、COPYLOC オプション経由で指定された順序で検索されます。これにより、ユーザーは、検索で PDSE (または PDS) ロケーションと z/OS UNIX ディレクトリーを混在させることができます。

NOCOPYLOC オプションが指定されている場合、COPYLOC オプションの以前のインスタンスはすべて無視されます。

COPYLOC オプションを使用して、コピー・メンバー・ロケーションの検索を排他的に制御するには、コピーブック・ロケーションを示すための既存の方法、JCL での ddname へのデータ・セットの割り振り、cob2 コマンドの -I オプションの指定など、を使用しないようにする必要があります。コンパイラーが cob2 か

ら呼び出される場合は、コピー・メンバーを現行ディレクトリーに保持しないようにする必要があります。これは、COPYLOC ロケーションが検索される前に、常に現行ディレクトリーが検索されるためです。

ヒント: 特にコンパイラーが cob2 コマンドから呼び出される場合は、COPYLOC オプションを使用して、コピー・メンバーの検索を排他的に制御すると便利な場合があります。

COPYLOC オプションが CBL ステートメントで指定されている場合、バッチ・プログラムの最初のプログラムでのみ使用できます。したがって、1つのファイルに複数の COBOL プログラムが含まれている場合、最初のプログラムの前に COPYLOC オプションを指定した CBL ステートメントを置くことはできますが、他のプログラムを指定することはできません。最初のプログラムに指定された COPYLOC オプション (および z/OS UNIX の下で cob2 コマンド・オプションとして指定された JCL の PARM に指定された COPYLOC オプションまたは COPYLOC オプション) は、ファイル内のすべてのプログラムに適用されます。CBL カードにある COPYLOC オプションで指定されたコピー・ロケーションは、呼び出しパラメーターとして指定された COPYLOC オプションにあるコピー・ロケーションの後で検索されます。

例

```
COPYLOC(MYLIB,DSN(USERID.COBOL.COPYLIB1))
COPYLOC(MYLIB,PATH('/home/userid/copylib1'))
COPYLOC(MYLIB,DSN(USERID.COBOL.COPYLIB2))
```

ライブラリー名 MYLIB を明示的に参照する COPY ステートメントの場合、上記のオプションがコンパイラーの 1 回の呼び出しで指定され、JCL で指定された場所 (または cob2 コンパイル用に指定された場所) にコピー・メンバーが見つからなかった場合、コピー・メンバーの追加検索は以下の場所で順次行われます。

1. USERID.COBOL.COPYLIB1 データ・セット
2. z/OS ユニックス ディレクトリー /home/userid/copylib1
3. USERID.COBOL.COPYLIB2 データ・セット

関連参照

COPY ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

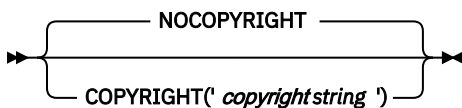
COPY メンバー探索順序 (*Enterprise COBOL for z/OS 言語解説書*)

ALLOWCOPYLOC (*Enterprise COBOL カスタマイズ・ガイド*)

COPYRIGHT

COPYRIGHT は、オブジェクト・モジュールが生成された場合に、オブジェクト・モジュール内にストリングを配置します。オブジェクトがプログラム・オブジェクトにリンクされている場合、ストリングはこのプログラム・オブジェクトとともにメモリーにロードされます。

COPYRIGHT オプションの構文



デフォルトは、NOCOPYRIGHT

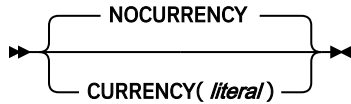
省略形は CPYR | NOCPYR

copyright string の長さは 64 文字に制限されています。

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



デフォルトは、NOCURRENCY

省略形は CURR | NOCURR

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
- 大文字のアルファベット文字 A、B、C、D、E、G、N、P、R、S、あなた、V、X、Z、またはそれらの小文字
- 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 種類の通貨のみを処理する場合は、CURRENCY オプションの代わりとして CURRENCY SIGN 使用する通貨記号を示す句 PICTURE プログラムの句。プログラムが複数の通貨タイプを処理する場合は、CURRENCY SIGN 条項と WITH PICTURE SYMBOL さまざまな通貨記号の種類を指定するためのフレーズ。

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 사용할 ことができます。

とき NOCURRENCY オプションが有効で、CURRENCY SIGN 句では、ドル記号 (\$) が PICTURE 通貨記号の記号。

区切り文字: CURRENCY オプション・リテラルは、APOST|QUOTE コンパイラ・オプション設定に関係なく、引用符またはアポストロフィで区切ることができます。

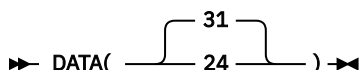
関連タスク

[70 ページの『通貨記号の使用』](#)

データ

DATA オプションは、動的データ域のストレージおよび他の動的ランタイム・ストレージが 16 MB 境界より上から取得されるのか下から取得されるのかに影響を与えます。

DATA オプションの構文



デフォルトは、DATA(31)

省略形: なし

再入可能プログラムの場合、DATA コンパイラオプションと HEAP ランタイムオプションは、動的データ領域 (WORKING-STORAGE そして FD レコード領域は 16MB ラインより下から取得されます (DATA(24)) または無制限のストレージ (DATA(31))。DATA 場所に影響を与えない LOCAL-STORAGE データ;STACK ランタイムオプションは、代わりにその場所を制御し、AMODE プログラムの。

31 ビット・アドレッシング・モードで実行されているプログラムで、24 ビット・アドレッシング・モードのプログラムにデータ引数を渡す場合は、DATA(24) を指定します。そうすると、データは必ず呼び出されたプログラムからアドレス可能になります。

外部データおよび QSAM バッファ: DATA オプションは、ストレージおよびそのアドレス可能性に影響を与える他のコンパイラ・オプションおよびランタイム・オプションと相互作用します。詳細については、関連情報を参照してください。

DATA コンパイラ・オプション設定は、ALLOCATE でストレージを取得する方法に影響します。

- もし DATA(24) が有効であり、LOC 31 のフレーズ ALLOCATE 文が指定されていない、ALLOCATE 16 MB ラインより下からストレージを取得します。
- もし DATA(31) が有効であり、LOC 24 のフレーズ ALLOCATE 文が指定されていない、ALLOCATE 16 MB ラインより上のストレージを取得しようとします。

AMODE 64 の考慮事項: DATA(24) コンパイラオプションはサポートされていません LP(64) が有効です。DATA(24) オプションを明示的に指定すると、警告メッセージが発行され、設定は破棄されます。

LOCAL-STORAGE セクションは、言語環境で管理されるスタック・ストレージから割り振られます。LE は 64 ビット・エンクレープの 2 GB 境界より上でスタック・ストレージを割り振ります。

関連概念

[40 ページの『ストレージとそのアドレス可能性』](#)

関連タスク

[620 ページの『プログラムを再入可能にする』](#)

言語環境プログラミングガイド (ランタイム・オプションの使用)

関連参照

[197 ページの『QSAM ファイル用のバッファの割り振り』](#)

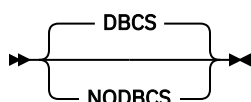
[435 ページの『RENT』](#)

ALLOCATE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

DBCS

DBCS を使用すると、コンパイラは、X'0E' (SO) および X'0F' (SI) を英数字リテラルの 2 バイト部分のシフト・コードとして認識するようになります。

DBCS オプションの構文



デフォルトは、DBCS

省略形: なし

DBCS が有効であると、リテラルはカテゴリ英数字のままで、リテラルの 2 バイト部分が構文検査されます。

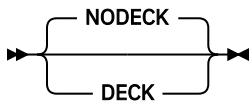
関連参照

[376 ページの『矛盾するコンパイラ・オプション』](#)

DECK

DECK は、オブジェクト・コードを 80 桁のレコード形式で作成する場合に使用します。DECK オプションを使用する場合は、コンパイル用の JCL で SYSPUNCH を必ず定義するようにしてください。

DECK オプションの構文



デフォルトは、NODECK

省略形は D | NOD

関連タスク

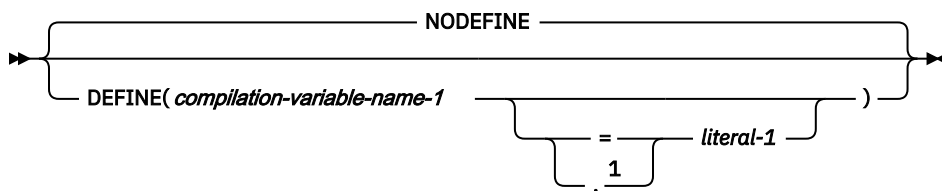
297 ページの『オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)』

DEFINE

DEFINE コンパイラ・オプションは、PARAMETER 句を持つ DEFINE ディレクティブを使用してプログラムで定義されたコンパイル変数にリテラル値を代入する場合に使用します。DEFINE オプションでコンパイル変数に対して指定されるリテラル値は、対応するコンパイル変数のパラメータ値と呼ばれることがあります。コンパイル変数は、任意の条件付きコンパイル・ディレクティブ (DEFINE、EVALUATE、IF など) 内で使用できます。条件付きコンパイル・ディレクティブに含まれている条件付きコンパイル変数は、現在示しているリテラル値に対するシンボル参照として扱われます。

DEFINE コンパイラ・オプションを使用すれば、プログラム・ソースの外側からコンパイル変数に値を代入できます。それが不要な場合は、プログラム・ソース内で DEFINE ディレクティブを使用してコンパイル変数を定義すれば十分です。

DEFINE オプションの構文



注:

¹ cob2 コマンドで z/OS UNIX シェルから COBOL コンパイラを呼び出す場合、「=」は使用できません。

デフォルトは、NODEFINE

省略形は DEF | NODEF

compilation-variable-name-1

プログラムにおいて条件付きコンパイル・ディレクティブで参照されるコンパイル変数の名前。対応する DEFINE ディレクティブ (PARAMETER 句あり) がプログラム内の *compilation-variable-name-1* に対して存在しない場合は、そのコンパイル変数に対して指定された DEFINE コンパイラ・オプションのインスタンスはいずれも無視されます。 *compilation-variable-name-1* はデータ名のユーザー定義語の規則に従って構成されます。ただし、DBCS 文字は名前には使用できません。詳細については、ユーザー定義の単語の中に *Enterprise COBOL for z/OS* 言語解説書。

literal-1

プログラムにおいて条件付きコンパイル関連ディレクティブで *compilation-variable-name-1* がシンボリックに表すリテラル値。 *literal-1* は以下のいずれかの項目でなければなりません。

- 通常の英数字リテラル ('abcd') または 16 進数リテラル (x'F1F2F3') として指定可能な英数字リテラル。 国語のリテラル、UTF-8 リテラル、DBCS リテラル、および null で終了する英数字リテラル (Z リテラル) はサポートされていません。
- 整数リテラル。
- ブール・リテラル (B'0' および B'1' のみサポート)

literal-1 が指定されない場合は、値 B'1' がコンパイル変数に代入されます。例えば、次のように指定したとします。

```
DEFINE(foo)
```

foo には値 B'1' が割り当てられます。

DEFINE オプションのインスタンスを複数指定して、異なる複数のコンパイル変数に対して値を定義できます。単一の条件付きコンパイル変数が複数定義されている場合は、最後の変数定義が、対応する条件付きコンパイル変数の値として使用されます。NODEFINE が、前の DEFINE オプション・インスタンスの後に指定されている場合は、すべての条件付きコンパイル変数に対する定義は取り消されます。

CBL ステートメントで指定される DEFINE オプションはバッチ・プログラムの最初のプログラムでのみ使用できます。そのため、ファイルに複数の COBOL プログラムが含まれている場合は、最初のプログラムの前にのみ DEFINE オプションを指定した CBL ステートメントを使用できます。最初のプログラムに対して指定された DEFINE オプション (および JCL の PARM で指定された DEFINE オプションや、z/OS UNIX の下で cob2 コマンド・オプションとして指定された DEFINE オプション) は、ファイル内のすべてのプログラムに適用されます。

関連参照

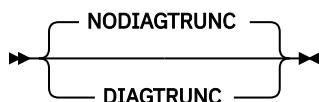
条件付きコンパイル (*Enterprise COBOL for z/OS* 言語解説書)

DEFINE (*Enterprise COBOL for z/OS* 言語解説書)

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



デフォルトは、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、診断メッセージが出されます。

TRUNC(BIN) オプションを指定した場合、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、診断メッセージはありません。

関連概念

[53 ページの『数値データの形式』](#)

[118 ページの『参照修飾子』](#)

関連参照

[454 ページの『TRUNC』](#)

DISPSIGN

DISPSIGN オプションは、符号付き数値項目の DISPLAY の出力形式設定を制御します。

DISPSIGN オプションの構文

DISPSIGN(COMPAT
SEP)

デフォルトは、DISPSIGN(COMPAT)

省略形: DS(C | S)

DISPSIGN(COMPAT)

DISPSIGN(COMPAT) を指定すると、符号付き数値項目の表示値は、旧バージョンの Enterprise COBOL と互換性がある形式になります。オーバープンチ符号が生成される場合もあります。

DISPSIGN(SEP)

DISPSIGN(SEP) を指定すると、符号付き 2 進数、符号付きパック 10 進数、またはオーバープンチ符号付きゾーン 10 進数の各項目の表示値は常に、先頭に分離符号が付いた形式になります。

以下の例は、DISPSIGN(COMPAT) オプションまたは DISPSIGN(SEP) オプションを指定した DISPLAY 出力を示しています。

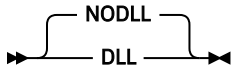
表 50. DISPSIGN(COMPAT) オプションまたは DISPSIGN(SEP) を指定した DISPLAY 出力		
データ項目	DISPSIGN(COMPAT) オプションを指定した DISPLAY 出力	DISPSIGN(SEP) オプションを指定した DISPLAY 出力
符号なし 2 進数	111	111
正の 2 進数	111	+111
負の 2 進数	11J	-111
符号なしパック 10 進数	222	222
正のパック 10 進数	222	+222
負のパック 10 進数	22K	-222
符号なしゾーン 10 進数	333	333
末尾が正のゾーン 10 進数	33C	+333
末尾が負のゾーン 10 進数	33L	-333
先頭が正のゾーン 10 進数	C33	+333
先頭が負のゾーン 10 進数	L33	-333

DLL

DLL は、ダイナミック・リンク・ライブラリー (DLL) サポートで使用可能なオブジェクト・モジュールを生成するようコンパイラーに指示する場合に使用します。DLL を使用可能にする必要があるのは、プログラムが DLL の一部である場合、プログラムが DLL を参照する場合、あるいはプログラムに INVOKE ステートメントやクラス定義などのオブジェクト指向 COBOL 構文が含まれている場合です。

注: 特定の CALL ステートメントでは、CALLINTERFACE 指示を使用して DLL オプションをオーバーライドできます。

DLL オプションの構文



デフォルトは、NODLL

省略形: なし

リンク・エディットの考慮事項: DLL オプションを指定してコンパイルされた COBOL プログラムは、RENT および AMODE 31 リンク・エディット・オプションを使用してリンク・エディットする必要があります。

NODLL は、DLL として使用できないオブジェクト・モジュールを生成するようにコンパイラーに指示します。

AMODE 64 の考慮事項: とき LP(64) コンパイラオプションが有効になっている場合、DLL そして NODLL オプションは無視され、デフォルト NODLL 生成されたオブジェクトファイルは LP(64) 常に DLL が有効になります。このようなオブジェクト・ファイルは DLL としても非 DLL としてもリンクできます。

の EXPORTALL オプションがサポートされています。このオプションは、DLL の作成時にプログラムからシンボルをエクスポートするために使用します。

例

この例では、2 つの COBOL プログラム、PROG1 と SUB1 を、z/OS UNIX 上でビルド、リンク、実行する方法を示す。プログラム SUB1 は DLL プログラムとしてビルドされ、カレント z/OS UNIX ディレクトリに存在する mylibdll.so という DLL としてリンクされる。プログラム PROG1 は基本的に DLL mylibdll.so のクライアントであり、DLL 内のプログラム SUB1 を呼び出す。

注: 実際には、次のような DLL は mylibdll.so 通常、DLL にリンクされた多数の COBOL プログラムで構成されます。

この例は次のファイルで構成されています。

buildrun.sh

z/OS UNIX サンプル・プログラムをビルドして実行するシェル・スクリプト

prog1.cbl

COBOL プログラム PROG1 DLL プログラムを呼び出す

sub1.cbl

COBOL プログラム SUB1 DLL の一部となる mylibdll.so

の buildrun.sh スクリプトは次のとおりです。

```
#!/bin/sh

# Remove existing artifact files
rm -f *.o *.lst *.so *.x prog1

# Build SUB1 as a DLL program and link it into a DLL called mylibdll.so,
# which will be written to the current directory
cob2 sub1.cbl -o libmydll.so "-qdll,exportall,rent,list" -bdll
```

```
# Build PROG1 with the DLL compiler option as it will be making a call
# to program SUB1 which resides in DLL mylibdll.so, but PROG1 itself
# is not linked as a DLL -- it is linked as a non-DLL program.
# Note that PROG1 must be linked with the side deck mylibdll.x produced
# in the previous step in order for the symbol SUB1 to be resolved
cob2 prog1.cbl -o prog1 libmydll.x "-qdll,list"

# Set LIBPATH to include the current directory so that it will be
# found at runtime when PROG1 is executed
export LIBPATH=.:$LIBPATH

echo ""

# Run PROG1
./prog1
```

実行中 `buildrun.sh` スクリプトは次の出力を生成します。

```
PP 5655-EC6 IBM Enterprise COBOL for z/OS 6.5.0 in progress ...
End of compilation 1, program SUB1, no statements flagged.
PP 5655-EC6 IBM Enterprise COBOL for z/OS 6.5.0 in progress ...
End of compilation 1, program PROG1, no statements flagged.

Entered PROG1
Calling DLL routine SUB1 from PROG1
Entered DLL routine SUB1
Exited DLL routine SUB1
Exited PROG1
```

PROG1、SUB1 がコンパイルされリンクされると、カレント z/OS UNIX ディレクトリに以下のファイルが存在する：

<code>buildrun.sh</code>	<code>libmydll.x</code>	<code>prog1.cbl</code>	<code>prog1.o</code>	<code>sub1.lst</code>
<code>libmydll.so</code>	<code>prog1</code>	<code>prog1.lst</code>	<code>sub1.cbl</code>	<code>sub1.o</code>

`prog1.cbl`:

```
identification division.
program-id. prog1.
data division.
working-storage section.
procedure division.
MainProgram.
    display "Entered PROG1"
    display "Calling DLL routine SUB1 from PROG1"
    call 'SUB1'
    display "Exited PROG1"
    goback.
end program prog1.
```

`sub1.cbl`:

```
identification division.
program-id. sub1.
data division.
working-storage section.
procedure division.
MainProgram.
    display 'Entered DLL routine SUB1'
    display 'Exited DLL routine SUB1'
    goback.
end program sub1.
```

関連タスク

[607 ページの『動的呼び出しの作成』](#)

関連参照

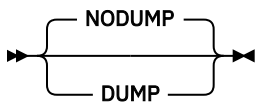
[376 ページの『矛盾するコンパイラ・オプション』](#)

コールインターフェイス (*Enterprise COBOL for z/OS* 言語解説書)

DUMP

DUMP を使用して、コンパイル時に内部コンパイラー・エラーに関するシステム・ダンプを作成します。

DUMP オプションの構文



デフォルトは、NODUMP

省略形は DU | NODU

通常は不使用: DUMP オプションは、IBM 担当員から依頼があったときにのみ使用してください。

ダンプは、コンパイラーのレジスターのリストとストレージ・ダンプで構成され、主として診断を行う担当者がコンパイラーのエラーを判別するために使用するものです。

DUMP オプションを使用する場合は、コンパイル時に DD ステートメントを組み込んで、SYSABEND、SYSUDUMP、または SYSMDUMP を定義してください。

DUMP を指定すると、コンパイラーは、異常終了処理の前に診断メッセージを出しません。その代わりに、ユーザー異常終了コードは IGYppnnnn で出されます。一般に、メッセージ IGYppnnnn はコンパイル時のユーザー異常終了コード nnnn に対応しています。ただし、IGYpp5nnn メッセージと IGYpp1nnn メッセージはどちらも、ユーザー異常終了 1nnn を生成します。NODUMP オプションを使用して再コンパイルすれば、そのメッセージが実際は 5nnn と 1nnn のどちらなのかを区別することができます。

次のものを含め、正常な終了処理を行いたい場合には、NODUMP を使用してください。

- コンパイル中でそれまでに作成された診断メッセージ。
- エラーの記述。
- 現在実行中のコンパイラー・フェーズの名前。
- エラー検出時に処理されていた COBOL ステートメントの行番号。(OPTIMIZE(1|2) を指定してコンパイルした場合、行番号は常に正しいとは限りません。一部のエラーについては、プログラムの最後の行となります。)
- 汎用レジスターの内容。

DUMP および OPTIMIZE(1|2) コンパイラー・オプションを一緒に使用することにより、コンパイラーは、以下のオブティマイザー・メッセージの代わりにシステム・ダンプを作成します。

```
"IGYOP3124-W This statement may cause a program exception at
                execution time."
```

この状況はコンパイラー・エラーではありません。NODUMP オプションを使用すると、コンパイラーはメッセージ IGYOP3124-W を出して、処理を継続することができます。

関連タスク

言語環境デバッグガイド (異常終了コードについて)

関連参照

376 ページの『[矛盾するコンパイラー・オプション](#)』

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。

注：特定の CALL ステートメントでは、CALLINTERFACE 指示を使用して DYNAM オプションをオーバーライドできます。

CALL *identifier* ステートメントの場合、ターゲット・プログラムは常に実行時にロードされ、このオプションの影響を受けません。

DYNAM オプションの構文



デフォルトは、NODYNAM

省略形は DYN | NODYN

制約事項：以下の場合に DYNAM コンパイラー・オプションを使用してはなりません。

- COBOL プログラムは、CICS 翻訳者または CICS コンパイラオプション
- EXEC SQL ステートメントが含まれており、CICS または Db2 呼び出し接続機能 (CAF) のもとで実行される COBOL プログラム

COBOL プログラムがダイナミック・リンク・ライブラリー (DLL) としてリンクされているプログラムを呼び出す場合は、DYNAM オプションを使用してはなりません。代わりに、NODYNAM および DLL オプションを使用してそのプログラムをコンパイルするようにしてください。

関連タスク

[611 ページの『静的呼び出しと動的呼び出しの両方の作成』](#)

[567 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』](#)

関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

コールインターフェイス (Enterprise COBOL for z/OS 言語解説書)

出口

EXIT オプションを使用して、各種コンパイラー関数に代わるユーザー提供モジュールを指定します。

コンパイラー入力には、INEXIT と LIBEXIT のサブオプションを使用して、それぞれ SYSIN と SYSLIB (またはコピー・ライブラリー) に代わるモジュールを指定することができます。

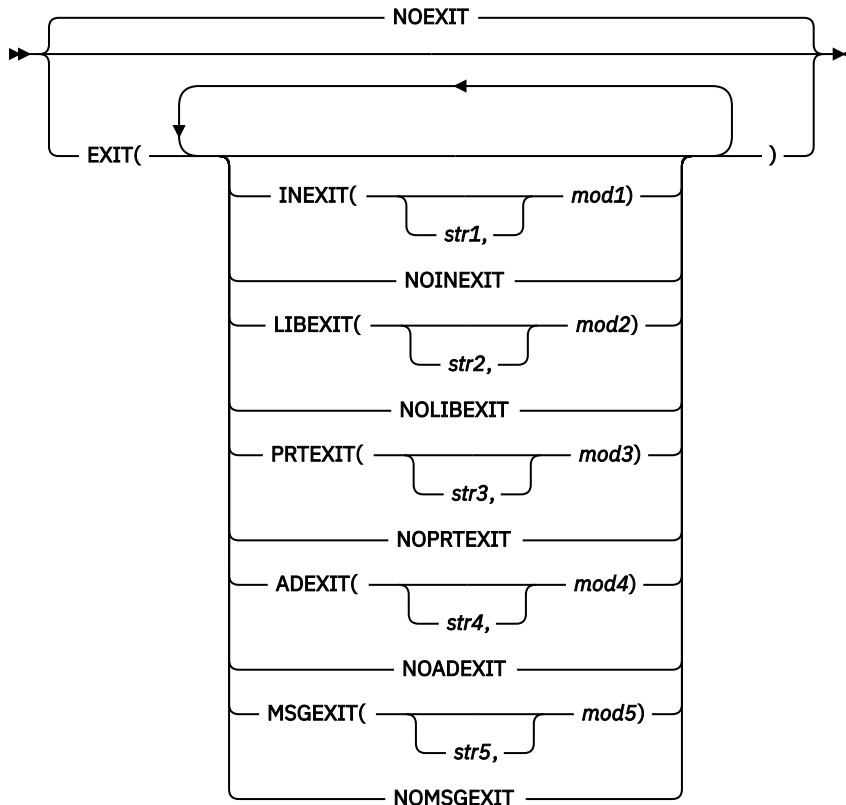
注：もし INEXIT または LIBEXIT を使用すると、デバッグセッションの「ソース」ビューは利用できなくなります。Debug for z/OS で「Switch View」の「Show Source」機能を選択しても、ソースファイルがないため動作しません。

コンパイラー出力には、PRTEXTIT サブオプションを使用して、SYSPRINT に代わるモジュールを指定します。

SYSADATA 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るモジュールを指定するには、ADEXIT サブオプションを使用します。

コンパイラー・メッセージをカスタマイズする (重大度を割り当てる診断メッセージへの FIPS (FLAGSTD) メッセージの変換など、コンパイラー・メッセージの重大度を変更したり、それを抑制したりする) には、MSGEXIT サブオプションを使用します。メッセージのカスタマイズに指定したモジュールは、コンパイラーが診断メッセージまたは FIPS メッセージを出すたびに呼び出されます。

EXIT オプションの構文



デフォルトは、NOEXIT

略語は以下の通り： NOEX|EX(INX|NOINX LIBX|NOLIBX, PRTX|NOPRTX, ADX|NOADX, MSGX|NOMSGX)

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できます。サブオプションの肯定形式と否定形式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。

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と、NOEXIT が有効になります。

EXIT オプションは、呼び出し時に JCL PARM フィールドで (TSO/E のもとではコマンド引数で)、またはインストール時にのみ指定できます。EXIT オプションを PROCESS (CBL) ステートメントに指定してはなりません。

INEXIT(['str1'],mod1)

コンパイラーは、SYSIN ではなく、ユーザー提供のプログラム・オブジェクト (*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、ユーザー提供のプログラム・オブジェクト (*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。

PRTEXIT(['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、ユーザー提供のプログラム・オブジェクト (*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、ユーザー提供のプログラム・オブジェクト (*mod4* はモジュール名) に渡します。

MSGEXIT(['str5'],mod5)

コンパイラーは、メッセージ番号を渡し、コンパイラー診断メッセージのデフォルトの重大度、または FIPS コンパイラー・メッセージのカテゴリ (数字コード) をユーザー提供プログラム・オブジェクトに渡します (ここで、mod5 はモジュール名です)。

名前 mod1、mod2、mod3、mod4、および mod5 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、str4、および str5 は、プログラム・オブジェクト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。これらの長さは最大 64 文字とすることができ、1 対のアポストロフィ (') で囲む必要があります。ストリング内の任意の文字を使用することはできますが、アポストロフィを含める場合は、アポストロフィを 2 つ (") にする必要があります。小文字は大文字に変換されます。

str1、str2、str3、str4、または str5 のいずれかが指定されると、そのストリングは次のフォーマット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 (ハーフワード境界に位置する) ハーフワードです。

LL string

898 ページの『例: MSGEXIT ユーザー出口』

EXIT オプションで指定されたコンパイラー出口モジュールは、アセンブラー言語、または高水準プログラミング言語 (COBOL など) のいずれかで実装できます。ただし、出口が Language Environment 準拠のプログラミング言語または Language Environment 準拠のアセンブラー言語で書かれている場合、この出口は再入可能でなければなりません。また、コンパイラー出口は、LP (64) コンパイラー・オプションを指定してコンパイルしてはいけません。

Enterprise COBOL コンパイラーは自動的に、事前に初期化された Language Environment をコンパイル時に管理し、この環境内でコンパイラー出口を呼び出します。このため、以下の規則が適用されます。

- コンパイラー出口は、メインプログラムではなく、サブプログラムとして実行されます。
- コンパイラー出口には、Language Environment を明示的に初期化または終了するロジックが含まれてはいけません。具体的には、出口では、環境管理のための RTEREUS ランタイム・オプション、IGZERRE 呼び出し可能サービス、または CEEPIPI 呼び出し可能サービスを使用してはなりません。
- コンパイラー出口では、STOP RUN ステートメントを使用してはいけません。

関連参照

376 ページの『矛盾するコンパイラー・オプション』

403 ページの『FLAGSTD』

885 ページの『EXIT コンパイラー・オプション』

EXPORTALL

EXPORTALL を使用すると、オブジェクト・デックをリンク・エディットして DLL を形成するときに、PROGRAM-ID 名および各代替入り口点名をそれぞれのプログラム定義から自動的にエクスポートするようコンパイラーに指示することができます。

EXPORTALL オプションの構文



デフォルトは、NOEXPORTALL

省略形は EXP | NOEXP

これらのシンボルが DLL からエクスポートされた場合、エクスポートされたプログラム名とエンタリー・ポイント名は、ルート・プログラム・オブジェクト内のプログラム、アプリケーション内の他の DLL プログラム・オブジェクト内のプログラム、およびその DLL にリンクされたプログラムから呼び出すことができます。

EXPORTALL オプションを指定する場合は、RENT リンカー・オプションも併せて指定する必要があります。

NOEXPORTALL は、記号をエクスポートしないようにコンパイラーに指示します。この場合、プログラムには、COBOL プログラム定義と同じプログラム・オブジェクトにリンク・エディットされた他のルーチンからのみアクセス可能です。

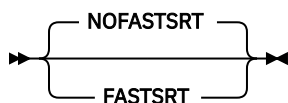
関連参照

376 ページの『[矛盾するコンパイラー・オプション](#)』

FASTSORT

FASTSORT を使用して、Enterprise COBOL の代わりに、IBM DFSORT または同等の製品でソート入出力を実行することができます。形式 1 SORT (ファイル SORT) ステートメントを使用したファイルのソートにのみ適用されます。

FASTSORT オプションの構文



デフォルトは、NOFASTSORT

省略形は FSRT | NOFSRT

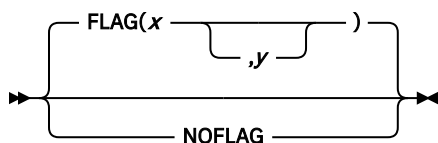
関連タスク

255 ページの『[FASTSORT によるソート・パフォーマンスの向上](#)』

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



デフォルトは、FLAG(I,I)

省略形は F | NOF

x そして y どちらもよい I、W、E、S、または U。

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ならないけません。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。

例:

```
000413      MOVE CORR WS-DATE TO HEADER-DATE
==000413==>  IGYPS2121-S      " WS-DATE " was not defined as a data-name. . . .
```

FLAG(*x*,*y*) が有効な場合、深刻度 *y* 以上のメッセージのほとんどは、そのメッセージの原因となった行の後のリストに埋め込まれる。IGYCB プレフィックスのあるメッセージがソース中に組み込まれることは決してありません。(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・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・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・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、コンパイラ出力の先頭に表示されます。
- IGYCB プレフィックスのある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、FLAG オプションの設定に関係なく、コンパイラ出力の末尾に表示されます。
- *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

307 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』

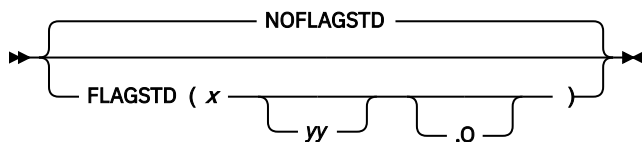
FLAGSTD

使用 FLAGSTD レベルまたはサブセットを指定する 85 COBOL 標準適合しているとみなされ、その情報メッセージを受け取る 85 COBOL 標準プログラムに含まれる要素。

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
- 廃止される言語エレメント
- 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、「非標準拠非標準」として識別された場合です)

FLAGSTD オプションの構文



デフォルトは、NOFLAGSTD

省略形: なし

x は、準拠していると見なされる 85 COBOL 標準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。その後、フラグ付き項目の総数と、その項目のタイプの要約があります。

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	
3		1	1	1	

EXIT コンパイラー・オプションの MSGEXIT サブオプションを使用することによって、FIPS 通知メッセージを診断メッセージに変換したり、FIPS メッセージを抑制したりできます。詳細については、MSGEXIT の処理に関する関連参照と、関連タスクを参照してください。

関連タスク

895 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

関連参照

376 ページの『矛盾するコンパイラー・オプション』

893 ページの『MSGEXIT の処理』

HGPR

HGPR オプションは、IBM z/Architecture プロセッサに装備されている 64 ビット・レジスターをコンパイラーで使用方法を制御します。

HGPR オプションの構文

➡ HGPR(PRESERVENOPRESERVE) ➡

デフォルトは、HGPR(PRESERVE)

省略形: なし

Enterprise COBOL コンパイラーは、IBM z/Architecture 汎用レジスター (GPR) の 64 ビットの幅を使用します。HGPR は「64 ビット GPR の高位半分」を表し、ネイティブ 64 ビット命令を使用することを意味します。

HGPR(PRESERVE)

HGPR(PRESERVE) を指定すると、コンパイラーは、プログラムが使用している 64 ビット GPR の上位半分を、関数のプロローグで保管してエピローグで復元するという方法によって保存します。

PRESERVE サブオプションは、プログラムの呼び出し元が Enterprise COBOL、Enterprise PL/I、z/OS XL C/C++ コンパイラー生成コードのいずれでもない場合にのみ必要です。

HGPR(NOPRESERVE)

HGPR(NOPRESERVE) を指定すると、コンパイラーは、プログラムが使用している 64 ビット GPR の高位半分を保持しないため、パフォーマンスが向上します。

関連タスク

262 ページの『DFSORT での COBOL ユーザー出口ルーチンの使用』

INITCHECK

INITCHECK オプションを使用すると、初期化されていないデータ項目がないかコンパイラーに検査させ、初期化されずに使用されている場合に警告メッセージを発行させることができます。

INITCHECK オプションの構文

➡ NOINITCHECK
INITCHECK (LAXSTRICT) ➡

デフォルトは、NOINITCHECK

サブオプション・デフォルト: INITCHECK がサブオプションなしで指定されている場合は INITCHECK(LAX)

省略形: IC | NOIC

NOINITCHECK

NOINITCHECK が有効になっている場合、コンパイラーは、初期化されていないデータ項目について警告メッセージをいっさい発行しません。

INITCHECK(LAX | STRICT)

INITCHECK または INITCHECK(LAX) が有効になっている場合、コンパイラーは、初期化されていないデータ項目について検査を行い、データ項目が初期化されずに使用されていると警告メッセージを発行します。ただし、ステートメントまでの少なくとも 1 つの論理パスでデータ項目が初期化される場合は、警告メッセージは発行されません。

INITCHECK(STRICT) が有効な場合にも、コンパイラーは、初期化されていないデータ項目がないか検査し、データ項目が初期化されずに使用されていると警告メッセージを発行します。ただし、INITCHECK(LAX) とは異なり、INITCHECK(STRICT) は、ステートメントまでのすべての論理パスでデータ項目が初期化されていない限り、ステートメントで使用されているデータ項目のデータが初期化されていないことについての警告メッセージを発行します。

INITCHECK(LAX) を指定した場合と INITCHECK(STRICT) を指定した場合の動作の違いを示すサンプル・プログラムを以下に示します。Y と Z は、VALUE 節のないデータ項目を表しています。

```
PROCEDURE DIVISION.  
  IF Y > 5  
    MOVE 2 TO Z  
  END-IF  
  DISPLAY Z
```

Z は DISPLAY ステートメントまでの一方のパスで初期化されていますが、もう一方のパスでは初期化されていません。したがって、INITCHECK(LAX) が有効な場合は、Y についての警告メッセージのみが発行されますが、INITCHECK(STRICT) の場合は Z についての警告メッセージも発行されます。

制約事項:

- INITCHECK オプションは、WORKING-STORAGE SECTION および LOCAL-STORAGE SECTION 内のデータ項目のみを分析します。具体的には、LINKAGE SECTION や FILE SECTION 内のデータ項目は分析しません。
- INITCHECK 分析では、外部データ項目やグローバル・データ項目は追跡されません。
- INITCHECK 分析では、テーブル内の個々のエレメントが別々に追跡されることはありません。代わりに、テーブルの 1 つのエレメントが初期化されると、そのテーブルの対応するすべてのエレメントはすべて初期化されるとみなされます。これは固定長テーブルにも可変長テーブルにも適用されます。
- INITCHECK 分析では、ポインターを介して行われる項目の初期化は追跡されません。例えば、初期化されていないデータ項目へのポインターが ADDRESS-OF を使用して作成され、そのデータ項目がそのポインターを介して初期化されている場合でも、INITCHECK 分析から警告メッセージが発行される可能性があります。
- 初期化されていないデータ項目が BY REFERENCE を使用して渡されている場合、警告メッセージは発行されません。一方、初期化されていないデータ項目が BY CONTENT および BY VALUE を使用して渡されている場合は、INITCHECK 分析から警告が発行されます。
- INITCHECK オプションでは、参照変更データ項目の個別バイトは正確には追跡されません。代わりに、データ項目が参照変更を使用してアクセスされた場合、そのデータ項目は初期化されるとみなされます。
- データ項目が、例えば SQL ホスト変数の結果としてアドレスが取られた他の項目を持つグループ内にある場合、そのデータ項目もアドレスが取られたと見なされ、アドレスが取られたすべてのデータ項目のセットは、常に外部関数の呼び出しによって設定されると見なされます。

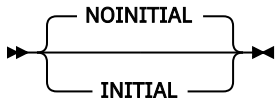
注:

- すべての INITCHECK 分析は、コンパイル時にのみ行われます。
- INITCHECK オプションは、コンパイル後のプログラムの動作やパフォーマンスには影響しません。
- INITCHECK オプションを使用すると、コンパイル時間とメモリー消費量が増加する可能性があります。
- INITCHECK オプションは、グループ内の最初の未初期化データ項目のみを報告し、出力します。それ以降のデータ項目は、初期化されていなくても出力されません。
- INITCHECK は、OPT(1) または OPT(2) と一緒に使用するとより正確になりますが、OPT(0) と一緒に使用する場合にも役立ちます。

INITIAL

INITIAL コンパイラー・オプションによって、あるプログラムとそのネスト・プログラムが、IS INITIAL 節が PROGRAM-ID 段落に指定されたかのように動作します。

INITIAL オプションの構文



デフォルトは、NOINITIAL

省略形: なし

INITIAL

INITIAL によって、プログラムとそのすべてのネスト・プログラムは、IS INITIAL 節が PROGRAM-ID 段落に指定されたかのように動作します。

注: INITIAL および IS INITIAL 節は、VALUE 節を持たないデータ項目には効果がありません。

NOINITIAL

NOINITIAL は、そのソースにおいて、既に IS INITIAL を PROGRAM-ID 段落に持つプログラムには効果がありません。

関連タスク [関連タスク](#)

[5 ページの『プログラムを初期状態に設定する』](#)

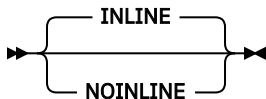
関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

INLINE

INLINE コンパイラー・オプションは、ソース・プログラムの PERFORM ステートメントによって参照されるプロシージャ (段落またはセクション) のインライン化を許可するかどうかを制御します。

INLINE オプションの構文



デフォルトは、INLINE

省略形: INL | NOINL

INLINE

INLINE は、パフォーマンスを向上させる可能性のあるオプションです。

INLINE を指定すると、コンパイラーは、OPTIMIZE(1)またはOPTIMIZE(2)が有効な場合に、ソース・プログラムの PERFORM ステートメントによって参照されるプロシージャのインライン化を許可します。特定の PERFORM ブロック内のプロシージャをインライン化するかどうかは、コンパイラーによって決定されます。

インライン化¹ 特定の段落でこれを防ぐには、>>INLINE ON そして>>INLINE OFF メモリ使用量が懸念され、実行可能ファイルの全体的なサイズを縮小したい場合、例えば、CICS 地域。

注:

1. ここで述べているインライン化という単語は、コンパイラーがプロシージャ (段落またはセクション) の PERFORM をそのプロシージャのコードのコピーで置き換えることを選択する可能性があるこ

とを暗示しています。コンパイラーは、PERFORM の位置にプロシージャー・コードを挿入することで、ロジックをプロシージャーに/プロシージャーから分岐するオーバーヘッドを省きます。

NOINLINE

NOINLINE を指定すると、有効な最適化レベルに関係なく、PERFORM ステートメントによって参照されるプロシージャーのインライン化は行われません。NOINLINE は、CICS 領域などでメモリー使用量が懸念されるため、実行可能ファイルの全体的なサイズを削減したい場合に指定できます。NOINLINE は、>>INLINE ON ディレクティブを使用してオーバーライドすることはできません。

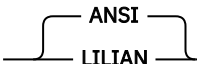
関連参照

INLINE ディレクティブ (*Enterprise COBOL for z/OS 言語解説書*)

INTDATE

INTDATE(ANSI) は、コンパイラーに、日付組み込み関数で使用する整数日付形式に 85 COBOL 標準の開始日を使用するように指示します。日付 1 は、1601 年 1 月 1 日です。INTDATE(LILIAN) は、コンパイラーに、日付組み込み関数で使用する整数日付形式に言語環境のリリアン開始日を使用するように指示します。日付 1 は、1582 年 10 月 15 日です。

INTDATE オプションの構文

► INTDATE() ◄

デフォルトは、INTDATE(ANSI)

省略形: なし

と INTDATE(LILIAN)、日付組み込み関数は、言語環境日付指定可能なサービス。

使用上の注意: INTDATE(LILIAN) が有効なときは、CEECBLDY は使用不能になります。これは、組み込み関数または呼び出し可能サービスを使用して ANSI 整数を意味のある日付に変換する方法がないためです。INTDATE(LILIAN) が有効になっている呼び出しのターゲットとして、CEECBLDY を指定した CALL literal ステートメントをコーディングすると、コンパイラーはこれを診断し、呼び出しターゲットを CEEDAYS に変換します。

関連タスク

[66 ページの『データ呼び出し可能サービスの使用』](#)

INVDATA

INVDATA オプションは、USAGE DISPLAY および PACKED-DECIMAL データ項目内のデータが有効であるかどうかをコンパイラーに指示し、有効でない場合は、コンパイラーの動作を指示します。

ほとんどのユーザーは、USAGE DISPLAY および USAGE PACKED-DECIMAL データ項目に有効なデータを持っているため、NUMPROC(NOPFD)を使用する場合でも、NOINVDATAを使用する必要があります。プログラムが NUMCHECK コンパイラー・オプションを使用してランタイムに無効なデータを処理していることが判明した場合でも、無効なデータを処理しないようにプログラムを変更して、NOINVDATAを使用する必要があります。

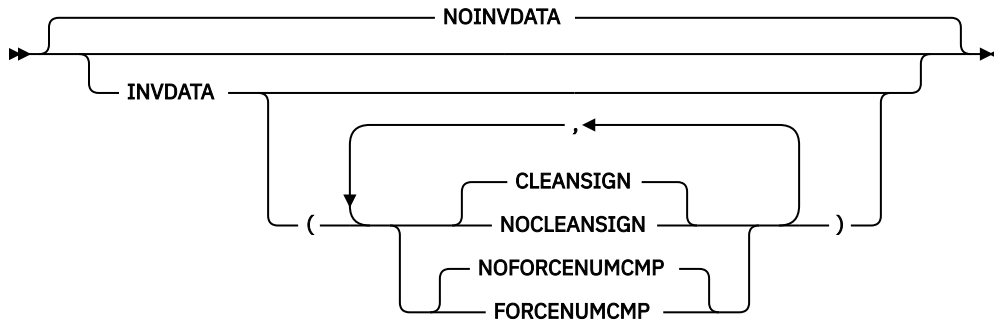
注: INVDATA オプションの目的は、無効な数値データの場合に、COBOL 4 以前のバージョンでコンパイルされたプログラムの動作と可能な限り互換性のある動作を提供することです。矛盾が見つかった場合、このオプションは、COBOL 4 以前のバージョンの動作とまったく一致した動作になるように更新されます。

INVDATA オプションが有効な場合、ゾーン 10 進数データ項目またはパック 10 進数データ項目に無効な数字または無効な符号コードが含まれている場合、またはゾーン 10 進数データ項目に無効なゾーン・ビットが含まれている場合に、コンパイラーは、COBOL 4 以前のバージョンとは異なる結果を生成する可能性がある既知の最適化を実行しません。

以下の表は、以前のバージョンの COBOL で使用されていた NUMPROC オプションのデフォルト値、および無効なデータがあるかどうかに応じて、を以前のバージョンの COBOL から COBOL 6.2 以降のバージョンにマイグレーションする際に INVDATA オプションおよび NUMPROC オプションを設定する方法についてのクイック・リファレンスを示しています。

表 51. 以前の COBOL バージョンからマイグレーションする場合の INVDATA オプションと NUMPROC オプションの設定			
COBOL のバージョン	無効なデータが存在しますか?	COBOL 6.1 以前のバージョンで使用される NUMPROC/ZONEDATA	COBOL 6.2 以降のバージョンでの INVDATA および NUMPROC の設定
COBOL 5 より前	いいえ	NUMPROC (MIG)	NOINVDATA, NUMPROC (NOPFD)
COBOL 5 より前	いいえ	NUMPROC (NOPFD)	NOINVDATA, NUMPROC (NOPFD)
COBOL 5 より前	いいえ	NUMPROC (PFD)	NOINVDATA, NUMPROC (PFD)
COBOL 5 より前	はい	NUMPROC (MIG)	INVDATA (FORCENUMCMP, NOCLEANSIGN), NUMPROC (NOPFD)
COBOL 5 より前	はい	NUMPROC (NOPFD)	INVDATA (NOFORCENUMCMP, CLEAN SIGN)、NUMPROC (NOPFD) または INVDATA、NUMPROC (NOPFD)
COBOL 5 より前	はい	NUMPROC (PFD)	INVDATA (NOFORCENUMCMP, CLEAN SIGN)、NUMPROC (PFD) または INVDATA、NUMPROC (PFD)
COBOL 5 以降	いいえ	ZONEDATA (PFD)	NOINVDATA
COBOL 5 以降	はい	ZONEDATA (NOPFD)	INVDATA (NOFORCENUMCMP, CLEAN SIGN) または単に INVDATA
COBOL 5 以降	はい	ZONEDATA (MIG)	INVDATA (FORCENUMCMP, CLEAN SIGN) ¹
1. INVDATA (FORCENUMCMP, NOCLEANSIGN) は、無効なデータが存在する場合の INVDATA (FORCENUMCMP, CLEAN SIGN) よりも、COBOL 5 より前の NUMPROC (MIG) 動作をより詳しく表したものです。無効なデータが存在する場合の COBOL 5 以降のバージョンでの ZONEDATA (MIG) の動作に満足できない場合は、INVDATA (FORCENUMCMP, NOCLEANSIGN) を使用して、無効なデータが存在する場合の COBOL 5 より前の NUMPROC (MIG) の動作をより厳密に模倣することを検討してください。			

INVDATA オプションの構文



デフォルトは、NOINVDATA

INVDATA がサブオプションなしで指定されている場合、デフォルトは INVDATA(NOFORCENUMCMP,CLEANSIGN) です。

省略形:

- INVD | NOINVD
- CS | NOCS
- FNC | NOFNC

NOINVDATA

NOINVDATA オプションが有効な場合、コンパイラーは、USAGE DISPLAY および PACKED-DECIMAL データ項目内のすべてのデータが有効であると想定し、可能な限り最も効率的なコードを生成します。例えば、コンパイラーは数値変換を避けるために、string 比較を生成する場合があります。

INVDATA(NOFORCENUMCMP | FORCENUMCMP)

INVDATA が指定され、FORCENUMCMP も NOFORCENUMCMP も指定されていない場合、NOFORCENUMCMP がデフォルトです。

INVDATA(FORCENUMCMP) オプションが有効な場合、コンパイラーは、各桁のゾーン・ビットを無視するゾーン 10 進数データ項目の比較を行う命令を生成します。例えば、ゾーン 10 進数値は比較前に、PACK 命令によってパック 10 進数に変換されます。

INVDATA(NOFORCENUMCMP) オプションが有効な場合、コンパイラーは、COBOL 4 またはそれ以前のバージョン NUMPROC(NOPFD | PFD)を使用する場合に COBOL 4 またはそれ以前のバージョンが実行するのと同じ方法で、数値比較またはゾーン 10 進数データの英数字比較のための命令を生成します。

- COBOL 4 (またはそれ以前のバージョン) でゾーン・ビットの対象となっていた場合、コンパイラーは英数字比較を生成し、英数字比較でもゾーン 10 進数データ項目内の各桁のゾーン・ビットが数値比較の対象となります。ゾーン 10 進数値はゾーン 10 進数のままです。
- COBOL 4 以前のバージョンでゾーン・ビットが無視された場合、ゾーン 10 進数データ項目内の各桁のゾーン・ビットが無視される数値比較をコンパイラーが生成します。ゾーン 10 進数値は比較前に、PACK 命令によってパック 10 進数に変換されます。

ゾーン 10 進数データの比較を生成するときに、コンパイラーが COBOL 4 以前のバージョンと同じ方法でゾーン・ビットを処理できるようにするには、COBOL 6 で使用される NUMPROC サブオプションが、COBOL 4 以前のバージョンで使用される NUMPROC サブオプションと一致する必要があります。

- COBOL 6 で COBOL 4 以前のバージョンの NUMPROC(NOPFD)動作を取得するには、COBOL 6 で INVDATA(NOFORCENUMCMP) (または単に INVDATA) および NUMPROC(NOPFD)を使用します。
- COBOL 6 で COBOL 4 以前のバージョンの NUMPROC(PFD)動作を取得するには、COBOL 6 で INVDATA(NOFORCENUMCMP) (または単に INVDATA) および NUMPROC(PFD)を使用します。

注: 符号コードは、NUMPROC オプション設定に従った有効な符号コードでなければなりません。さらに、下位バイトには、SIGN IS LEADING または SIGN IS SEPARATE のいずれかが指定された有効なゾーン (x'F') がなければなりません。

INVDATA(CLEANSIGN | NOCLEANSIGN)

INVDATA が指定され、CLEANSIGN も NOCLEANSIGN も指定されていない場合、CLEANSIGN がデフォルトです。

INVDATA(CLEANSIGN) オプションが有効な場合、コンパイラーは、比較演算、加算演算、減算演算、乗算演算、および除算演算の入力時に USAGE DISPLAY および USAGE PACKED-DECIMAL データ項目の符号ニブルを消去するコードを生成します。

注: CLEANSIGN は、SIGN IS SEPARATE で定義された USAGE DISPLAY 項目には適用されません。

INVDATA(NOCLEANSIGN) オプションが有効な場合、コンパイラーは、比較、加算、減算、乗算、および除算操作の入力で USAGE DISPLAY および USAGE PACKED-DECIMAL データ項目の符号ニブルを消去するコードの生成を回避します。これにより、演算のいずれかのオペランドに無効な符号ニブルが含まれている場合に SOC7 が異常終了する可能性が高くなります。

注: INVDATA オプションは、無効な数字、無効な符号コード、または無効なゾーン・ビットを含む可能性がある USAGE DISPLAY または PACKED-DECIMAL データ項目の MOVE ステートメント、比較、および計算の動作に影響を与えます。

次の例では、REDEFINES によって強制投入された、データ項目 VALUE1 の中央のゾーン・ビットに無効なゾーン・ビット 4 を持つデータ項目を確認できます。

```
77 VALUE0    PIC X(4) VALUE '00 0'.          *>  x'F0F040F0'
77 VALUE1    REDEFINES VALUE0 PIC 9(4).
PROCEDURE DIVISION.
  IF VALUE1 = ZERO
    DISPLAY 'INVDATA(FORCENUMCMP) is in effect ' VALUE1
  ELSE
    DISPLAY 'INVDATA(NOFORCENUMCMP) is in effect ' VALUE1
  END-IF
```

この例では、次のようになっています。

- COBOL 4 以前のバージョンでは、NUMPROC(MIG) オプションが使用されている場合はテストが true になり、NUMPROC(NOPFD | PFD) の場合は false になります。
- 6.2 以降のバージョンの場合:
 - NOINVDATA を使用する場合、テストは OPT(0) では true、OPT(1 | 2) では false になります。
 - INVDATA(NOFORCENUMCMP) を使用すると、どの OPT 設定でもテストは false になります。

COBOL 6 へのマイグレーションを容易にするには、以下のようにします。

- 数字、符号コード、およびゾーン・ビットが有効な場合は、NOINVDATA を使用します。COBOL 4 以前のバージョンで NUMPROC(PFD) または NUMPROC(NOPFD) を使用した場合は、COBOL 6 の使用時に同じ NUMPROC 設定を使用します。COBOL 4 以前のバージョンで NUMPROC(MIG) を使用した場合は、COBOL 6 の使用時に NUMPROC(NOPFD) を使用します。
- データに無効な数字、無効な符号コード、または無効なゾーン・ビットがある場合は、プログラムが実行時に数値データ項目に無効なデータを持たないように、プログラムまたはシステムを変更してください。プログラムまたはシステムを訂正した後、優先 NOINVDATA オプションを使用することができます。この作業を含めることができず、引き続き無効なデータを使用して実行する必要がある場合にのみ、INVDATA に対して以下の選択項目を検討してください。
 - COBOL 4 以前のバージョンからマイグレーションし、NUMPROC(MIG) を COBOL 4 で使用した場合は、INVDATA(FORCENUMCMP, NOCLEANSIGN) および NUMPROC(NOPFD) を COBOL 6.2 以降のバージョンで使用して、無効データが存在する場合の以前の NUMPROC(MIG) の動作を最も厳密に模倣します。
 - COBOL 4 以前のバージョンからマイグレーションし、NUMPROC(NOPFD) を COBOL 4 で使用した場合は、INVDATA(NOFORCENUMCMP, CLEANSIGN) (または単に INVDATA) および NUMPROC(NOPFD) を COBOL 6 で使用します。
 - COBOL 4 以前のバージョンからマイグレーションし、NUMPROC(PFD) を COBOL 4 で使用した場合は、INVDATA(NOFORCENUMCMP, CLEANSIGN) (または単に INVDATA) および NUMPROC(PFD) を COBOL 6 で使用します。

注:

- COBOL 4 以前のバージョンから COBOL 5 または 6 へのマイグレーションを過去に完了し、COBOL 5 または 6 で非推奨の ZONEDATA(MIG) オプションを使用しており、動作に満足している場合は、ZONEDATA(MIG)ではなく INVDATA(FORCENUMCMP,CLEANSIGN)を使用してください。
- 明らかに無効なデータが検出された場合、これらのオプションを使用したとしても、古いコンパイラの動作を完全に一致させることは常に可能というわけではありません。例えば、比較の場合でも、INVDATA(NOFORCENUMCMP)はすべてのケースで COBOL 4 と同じ結果になるわけではありません。

パフォーマンスの考慮事項、NOINVDATA は、INVDATA よりもランタイム・パフォーマンスが優れています。INVDATA は、NUMPROC(PFD)が提供できる最適化の一部を使用不可にします。

関連タスク

60 ページの『[非互換データの検査 \(数値のクラス・テスト\)](#)』

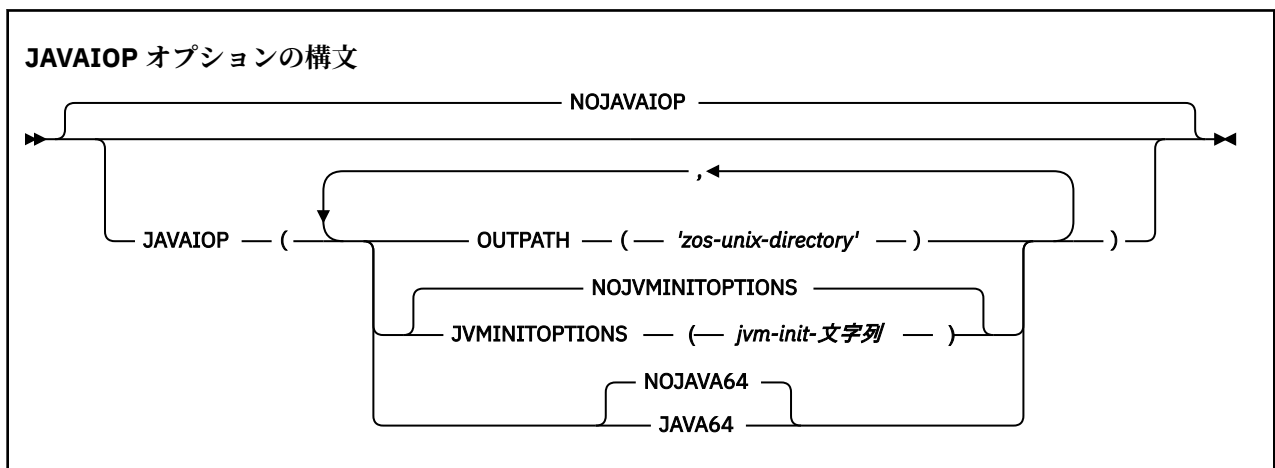
関連参照

422 ページの『[NUMCHECK](#)』

426 ページの『[NUMPROC](#)』

JAVAIOP

JAVAIOP オプションは、JAVA-CALLABLE ディレクティブまたは JAVA-SHAREABLE ディレクティブを介して、または CALL ステートメントを使用して Java 静的メソッドを呼び出すことによって、Java と相互運用する COBOL プログラムの動作を制御します。



デフォルトは、NOJAVAIOP

省略形:

- JIOP = ジャヴァイオプ
- ノジオプ = ノジャヴァイオプ
- OP = アウトパス
- jvmi = javainitoptions
- nojvmi = nojavainitoptions

OUTPATH('zos-unix-directory')

デフォルトは、コンパイラーが実行されているユーザー ID の現行作業ディレクトリーです。コンパイラーが z/OS UNIX から呼び出されない場合、現行作業ディレクトリーは、コンパイラーが実行されているユーザー ID のホーム・ディレクトリーです。'zos-unix-directory'は引用符で囲んで指定し、コンパイラを実行する前に存在していなければなりません。

注: OUTPATH オプションを使用する場合は「zos-unix-directory」を指定する必要があり、「JAVAIOP(OUTPATH)ではサポートされない。それ以外の場合は、コンパイラ診断メッセージが表示されます。

OUTPATH サブオプションは、以下のいずれかを含む COBOL プログラムによって成果物ファイルが生成される z/OS UNIX ディレクトリーを指定するために使用されます。

- JAVA-CALLABLE ディレクティブ
- JAVA_SHAREABLE ディレクティブ
- 形式 '*Java.java-class-name.java-static-method-name*' のリテラル呼び出しターゲットを指定した CALL ステートメント。

生成される成果物ファイルには、以下が含まれます。

- JAVA-CALLABLE ディレクティブを含むプログラムの COBOL ネイティブ・メソッド・スタブ
- JAVA-SHAREABLE ディレクティブを使用するプログラム用に生成される WORKING-STORAGE 初期設定スタブ
- COBOL アプリケーションに、'*Java.java-class-name.java-static-method-name*' という形式のリテラル呼び出しターゲットを持つ CALL ステートメントが含まれている場合に、コンパイラーによって生成される Java 呼び出しスタブ・ファイル

サポートされる zos-unix-ディレクトリーの最長は 1023 文字です。

NOJAVAINITOPTIONS | JAVAINITOPTIONS(*jvm-init-string*)

jvm-init-文字列初期化に使用する初期化文字列を指定します。ジャワ仮想マシン (JVM)。これは、アプリケーションが COBOL 側で開始され、次に Java を呼び出すシナリオの場合にのみ関係します。これにより、アプリケーションの COBOL 側が JVM を開始する必要があります。

デフォルトは `-Djava.library.path=.` です。これは、COBOL ネイティブ・メソッド・スタブを含む DLL および作業用ストレージ初期化スタブが、Java アプリケーションが実行されている現行ディレクトリーに配置されていることを示します。

jvm-init-string の最長は 256 文字です。

NOJAVA64 | JAVA64

デフォルトは NOJAVA64 です。LP(32) が有効な場合に JAVA64 サブオプションを指定すると、COBOL プログラムが 64 ビット Java と相互運用することを示します。結果として、Java オブジェクト参照は、4 バイトではなく 8 バイトとして扱われます。

注: JAVAIOP コンパイラー・オプションは、一連のプログラムの最初のプログラムに対してのみ指定できます (バッチ・コンパイル)。シーケンス内の最初のプログラム以外のプログラムに対して CBL または PROCESS ステートメントで JAVAIOP を指定すると、コンパイラー診断メッセージを受け取ります。

関連タスク

CALL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

JAVA-CALLABLE (*Enterprise COBOL for z/OS* 言語解説書)

JAVA-SHAREABLE (*Enterprise COBOL for z/OS* 言語解説書)

LANGUAGE

LANGUAGE オプションは、コンパイラー出力の印刷に使用する言語を選択する場合に使用します。選択された言語で印刷される情報には次のものがあります。診断メッセージ、ソース・リストのページ・ヘッダーとスケール・ヘッダー、FIPS メッセージ・ヘッダー、メッセージ要約ヘッダー、コンパイル要約、および特定のコンパイラー・オプション (MAP、XREF、VBREF、および FLAGSTD) を選択した結果として生じるヘッダーと表記が含まれます。

LANGUAGE オプションの構文

►► LANGUAGE(*名前*) ◄◄

デフォルトは、LANGUAGE(ENGLISH)

省略形: LANG(EN|UE|JA|JP)

name は、コンパイラー出力メッセージに使用する言語を指定します。LANGUAGE オプションに使用できる可能な値を、以下の表に示します。

表 52. LANGUAGE コンパイラー・オプションの値		
名前	省略形 ¹	出力言語
ENGLISH	EN	英大/小文字混合 (デフォルト)
JAPANESE ³	JA, JP	日本語 (日本語文字セットを使用)
UENGLISH ^{2, 3}	UE	英大文字

1. インストール先のシステム・プログラマーが、ここに説明されている以外の言語を提供した場合は、その言語名の少なくとも最初の 2 文字を指定しなければなりません。

2. UENGLISH 以外の言語を指定するには、該当する言語の機能をインストールしなければなりません。

3. コンパイラー・メッセージを大文字の英語に変更したり日本語に変更したりするには、LANGUAGE コンパイラー・オプションを使用するだけでなく、コンパイル時に言語環境・ランタイム・オプション NATLANG も設定する必要があります。使用をお勧めします CEEOPTS DD コンパイル JCL 内。

例えば、メッセージを日本語に変更するには、LANGUAGE (JA) コンパイラー・オプションを使用し、コンパイル時には NATLANG LE ランタイム・オプションも指定します。

```
//CEEOPTS DD *  
           NATLANG(JPN)  
/*
```

コンパイル時に (CBL または PROCESS ステートメントを使用して) LANGUAGE オプションが変更された場合は、一部の初期テキストは、コンパイラーの開始時に有効であった言語を使用して印刷されます。

NATLANG: NATLANG ランタイム・オプションを使用すると、エラー・メッセージ、月名、および曜日名を含む、ランタイム環境で使用される各国語を制御することができます。LANGUAGE コンパイラー・オプションと NATLANG ランタイム・オプションは互いに独立して働きます。それらのどちらかに優先権を与えずに同時に使用することができます。

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*) ◄◄

デフォルトは、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。

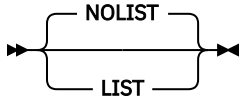
LINECOUNT(0) を指定すると、コンパイル・リストではページ替えが行われません。

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



デフォルトは、NOLIST

省略形: なし

次の項目も出力リストに書き込まれます。

- 定数域
- プログラム・プロログ域 (PPA1、PPA2、PPA3、PPA4)
- タイム・スタンプ、コンパイラー・バージョン、およびビルド・レベル情報
- コンパイラー・オプションおよびプログラム情報
- ベース・ロケーター・テーブル
- 外部シンボル辞書
- 初期ヒープ・ストレージ・マップ
- スタック・ストレージ・マップ

出力が生成されるのは、以下の場合です。

- COMPILE オプションを指定している、または NOCOMPILE(x) オプションが有効であり、エラー・レベル x 以上が発生していない。
- OFFSET オプションを指定していない。

アセンブラー・リスト出力を制限したい場合は、PROCEDURE DIVISION で *CONTROL (または *CBL) LIST か、NOLIST ステートメントを使用します。*CONTROL NOLIST 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 LIST ステートメントによって出力が通常の LIST 形式に戻されない限り、リストには含まれません。

関連タスク

[500 ページの『リストの入手』](#)

関連参照

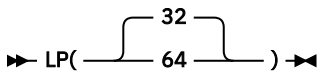
[376 ページの『矛盾するコンパイラー・オプション』](#)

*CONTROL (*CBL) ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

LP

LP コンパイラー・オプションを使用して、AMODE 31 (31 ビット) または AMODE 64 (64 ビット) プログラムを、関連言語機能を使用可能にして生成するかどうかを指示します。

LP オプションの構文



デフォルトは、LP(32)

省略形: なし

LP(32)

関連する言語機能を有効にして AMODE 31 (31 ビット) プログラムを生成することを指定します。

LP(64)

関連する言語機能を有効にして AMODE 64 (64 ビット) プログラムを生成することを指定します。

オプション指定。 LP オプションは、他のコンパイラー・オプションを指定する方法で指定できます。ただし、CBL または PROCESS ステートメントでこのオプションを指定する場合は、一連のプログラム (バッチ・コンパイル) の最初のプログラムに対してのみ LP オプションを指定できます。バッチでは、後続のプログラムに対してオプションの値を変更することはできません。

実行時の考慮事項: 現在、言語環境は、同じアプリケーションに AMODE 64 プログラムと AMODE 31 プログラムを混在させることをサポートしていません。1つのプログラムを LP(64)でコンパイルする場合は、アプリケーション内のすべてのプログラムも LP(64)でコンパイルする必要があります。静的呼び出しでは、外部名の解決時にバインダーがアドレッシング・モードの混在を検出するとメッセージが発行されます。動的 CALL の場合、混合 AMODE サポートなしで、一方が AMODE 64 でもう一方が AMODE 31 または 24 である場合、プログラム間の CALL に対して実行時エラーが発生します。混合 AMODE での動的呼び出しサポートの詳細については、31/AMODE64 環境については、[596 ページの『AMODE 31 および AMODE 64 の混合プログラムを使用する COBOL アプリケーション』](#)。

LP(64)コンパイラー・オプションを使用すると、コンパイル・プロセスに POSIX (ON) モードで実行されるコンポーネントが組み込まれます。これは、このオプションを使用してコンパイラーを実行するユーザーごとに、ラクトールに OMVS セグメントが設定されている (または ラクトール 代替として同等) 必要があることを意味します。

プログラムに関する制限事項 LP(64):

- コンパイルされたプログラム LP(64) 含むことができない XML GENERATE または XML PARSE 声明、JSON GENERATE または JSON PARSE ステートメント、オブジェクト指向 COBOL ステートメント、ALTER 声明、GO TO. 声明、および DISPLAY ... UPON SYSPUNCH 声明。
- AMODE 64 プログラムは CICS や IMS でも実行できません。
- コンパイラー出口 (EXIT コンパイラー・オプションを使用) は、LP(64)コンパイラー・オプションを指定してコンパイルしてはなりません。
- いつ LP(64)が有効になっている場合、DLL そして NODLL オプションは警告なしに無視され、NODLL リストには常に表示されます。

注: と LP(64)一部のコンパイラ オプションは適用されません。詳しくは、[588 ページの『コンパイラー・オプションを使用した、AMODE 64 プログラムのコンパイル』](#)を参照してください。

関連参照

[299 ページの『z/OS のもとでのコンパイラー・オプションの指定』](#)

[300 ページの『PROCESS \(CBL\) ステートメントでのコンパイラー・オプションの指定』](#)

[302 ページの『複数プログラムのコンパイル \(バッチ・コンパイル\)』](#)

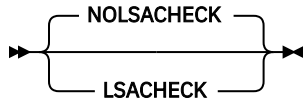
LSACHECK

LSACHECK オプション (LSACHECK は Linkage Section Addressability Check の略) を使用して、アドレス指定を確立する前に LINKAGE-SECTION データ項目が不用意に使用されるのを防ぎます。

LSACHECK コンパイラー・オプションは、BLL セル (LINKAGE-SECTION データ・アイテムのアドレス) を初期化するコードを、以前のデフォルトである 'x'00000000' 「x'7FFFF000'」に生成するようコンパイラーに指示する。NOLSACHECK を使っても、BLL セルは 'x'00000000' に初期化される。

この特定のアドレスは、z/OS システム内のページに対するもので、ストレージのバックアップがないため、このページを参照すると OC4ABEND となる。LSACHECK を使用すると、アドレスがプログラムまたは入力パラメータで明示的に設定されていない参照データ項目は、OC4ABEND になります。

LSACHECK オプションの構文



デフォルトは、NOLSACHECK

省略形: LSAC | NOLSAC

アドレサビリティを持たない LINKAGE SECTION データ項目を持つ最も一般的な方法は、この例のように PROCEDURE DIVISION USING 文で参照されない場合です：

```
LINKAGE SECTION.  
  01 LINKAGE-ITEM PIC C(100).  
  
PROCEDURE DIVISION.          *> LINKAGE-ITEM will not get any  
                              *> address from a CALLing program
```

プログラムがリンクセクションのデータ項目のアドレス指定可能性を設定するには、いくつかの方法がある。ひとつは PROCEDURE DIVISION USING のフレーズに含める方法だが、ここでは他の方法もいくつか紹介しよう：

```
LINKAGE SECTION.  
  01 LINKAGE-ITEM PIC C(100).  
  
PROCEDURE DIVISION USING LINKAGE-ITEM.  *> LINKAGE-ITEM will  
                                         *> get the address from the USING  
                                         *> phrase of the CALL statement  
                                         *> of the CALLing program  
  
*> NOTE: If a program is called with no USING phrase, the  
*>         address will be changed and will not have the LSACHECK value.  
*>         Our testing showed a different ABEND from the LSACHECK  
*>         ABEND.  
  
*> In the following example, the BLL for LINKAGE-ITEM will  
*> be set to the value in Pointer-1  
  
    SET ADDRESS OF LINKAGE-ITEM To Pointer-1  
  
*> In the following example, the BLL for LINKAGE-ITEM will  
*> be set to the address of Working-Storage-Item  
  
    SET ADDRESS OF LINKAGE-ITEM To  
        ADDRESS OF Working-Storage-item  
  
*> In the following example, the BLL for LINKAGE-ITEM will  
*> be set to the address of the storage acquired by the  
*> ALLOCATE statement  
    ALLOCATE LINKAGE-ITEM
```

パフォーマンスとの関係

LSACHECK を使用してコンパイルされたプログラムには余分な処理がないため、LSACHECK が結果のプログラム・パフォーマンスに影響を与えることはありません。

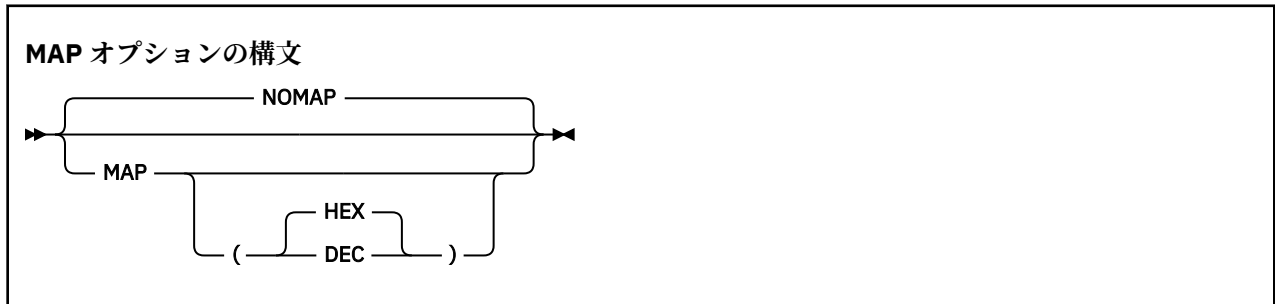
AMODE 64 インプリケーション

LSACHECK と LP(64) でコンパイルされたプログラムでは、コンパイラは BLL を 'x'7FFFF000' に設定した LP(32) プログラムと同様の動作を得るために、BLL を 'x'000000007FFFF000' に初期化するコードを生成します。

注：ABEND やアプリケーションの停止なしにゼロアドレスの使用レポートを取得する方法はすでにあるので（z/OS 「ゼロアドレス検出」機能）、他のいくつかのコンパイラオプションのように MSG サブオプションでその機能を複製する必要はない。

MAP

使用 MAP リストを作成するオプション DATA DIVISION 項目および暗黙的に宣言されたすべての項目。16 進数または 10 進数のオフセットを表示するかどうかも指定できます。MAP リストに出力されます。



デフォルトは、NOMAP

サブオプション・デフォルト: MAP がサブオプションなしで指定されている場合は MAP (HEX)

省略形: なし

HEX

MAP (HEX) を指定した場合、グループ内のデータ項目オフセットは 16 進表記になります。

DEC

MAP (DEC) を指定した場合、グループ内のデータ項目オフセットは 10 進表記になります。

出力には、以下の項目が含まれます。

- DATA DIVISION のマップ
- 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 と LOCAL-STORAGE のサイズ、およびプログラムが NORENT オプションを使用してコンパイルされた場合には、オブジェクト・コード内でのその位置

制限したい場合は MAP 出力、使用*CONTROL MAP または NOMAP 声明文 DATA DIVISION。続くソースステートメント*CONTROL NOMAP リストには含まれません。*CONTROL MAP ステートメントは出力を通常に戻す MAP フォーマット。例:

*CONTROL NOMAP	*CBL NOMAP
01 A	01 A
02 B	02 B
*CONTROL MAP	*CBL MAP

MAP (HEX | DEC) オプションが有効になっている場合、ソース・コード・リストに組み込まれた MAP レポートも取得します。凝縮された MAP 情報は、データ名定義の右側に表示されます。WORKING-STORAGE SECTION、FILE SECTION、LOCAL-STORAGE SECTION、そして LINKAGE SECTION の DATA DIVISION。XREF データと組み込み MAP 要約の両方が同じ行にある場合は、組み込み MAP 要約が先にリストされます。

505 ページの『例: MAP 出力』

関連概念

485 ページの『第 22 章 デバッグ』

関連タスク

500 ページの『リストの入手』

関連参照

*CONTROL (*CBL) ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

MAXPCF

MAXPCF オプションは、最大プログラム複雑度係数値を指定するために使用します。プログラム複雑度係数 (PCF) はコンパイラによって計算され、その値はリスト・ファイルに収容されます。プログラムの PCF が最大値を超えると、コンパイラは自動的に最適化レベルを下げて、コンパイルを高速化し、ストレージ使用量を削減します。そのため、一組のプログラムをコンパイルする際に、プログラムごとに OPTIMIZE オプション値を指定する必要はありません。

MAXPCF オプションの構文

➡ MAXPCF(*n*) ⬅

デフォルトは、MAXPCF(1000000)

省略形: なし

n は整数 0 - 999999 でなければなりません。

複雑度係数の計算時には、以下のようなプログラムの要素が考慮に入れます。

- COBOL 文の数は PROCEDURE DIVISION、生成されたステートメントを含む CICS、SQL または SQLIMS オプションと拡大 COPY そして REPLACE 声明
- value 節のある WORKING-STORAGE データ項目または LOCAL-STORAGE データ項目の初期化操作
- 可変長グループまたはサブグループに対する操作 DATA DIVISION 実行時にサイズを計算する

注: PCF は、プログラムの複雑さを示す尺度ではありません。これは単に、COBOL 項目数が多い場合に最適化の問題の原因となり得る、COBOL 項目の数です。プログラムの複雑さを計測するには、IBM 開発者 z/OS に備わっている [測定基準機能](#) のようなものを使用する必要があります。

巨大で複雑なプログラムの場合、コンパイラが最適化しようとするプログラムの複雑度に対してしきい値を MAXPCF オプションにより設定できます。MAXPCF 値を下げると最適化のレベルが下がるため、コンパイラに必要なメモリおよびコンパイル時間は少なくなります。コンパイル時間が長くなるのを承知の上でプログラムの最適化を試みるのであれば、MAXPCF 値を上げてください。

MAXPCF(0) を指定した場合、プログラムの複雑度には制限が課せられず、MAXPCF オプションは無効になります。

MAXPCF(*n*) を指定し、*n* がゼロでない場合、プログラムの複雑度係数が *n* を超えると、OPTIMIZE(1) または OPTIMIZE(2) の指定が OPTIMIZE(0) にリセットされ、警告メッセージが生成されます。

COBOL ソース・ファイルに一連のソース・プログラムが含まれる場合 (バッチ・コンパイル)、MAXPCF 制限はプログラムごとに適用されます。

注:

- OPTIMIZE(1) オプションまたは OPTIMIZE(2) オプションをインストール時に、固定された指定変更不可オプションとして設定すると、ゼロ以外の *n* を持つ MAXPCF(*n*) は矛盾するオプションになります。この場合、OPTIMIZE オプションが優先され、MAXPCF(0) オプションが強制されます。
- MAXPCF の値を *n* (*n* はデフォルトより大きい値) に上げるにより、または MAXPCF(0) を指定することにより、デフォルトのしきい値よりプログラムの最適化を向上させようとする場合は、コンパイラがコンパイルするのに過度の時間がかかったり、メモリ不足によりコンパイルに失敗したりする可能性があります。

関連参照

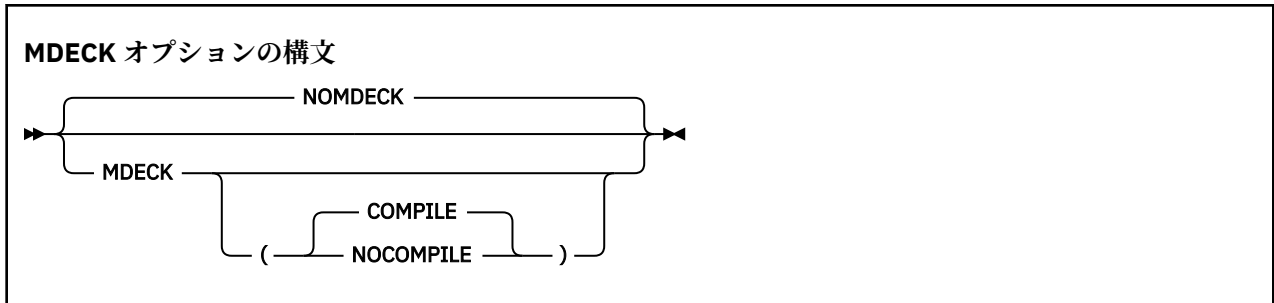
[429 ページの『OPTIMIZE』](#)

MDECK

の MDECK コンパイラオプションは、更新された入力ソースのコピーライブラリ処理 (つまり、結果の COPY、BASIS、REPLACE、EXEC SQL INCLUDE、EXEC SQLIMS INCLUDE、および条件付きコンパイルディレクティブステートメント) がファイルに書き込まれます。

Enterprise COBOL が z/OS ユニックス のもとで稼働している場合、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。のために Enterprise COBOLTSO またはバッチで実行する場合、MDECK 出力は、SYSDMDECK DD 割り当ては、MVS データセットには RECFM F または FB と LRECL80 バイト。

注：以下の条件でコンパイルする場合 z/OSTSO またはバッチの場合、COBOL コンパイラは SYSDMDECKMDECK オプションまたは NOMDECK オプションのどちらかを指定したかに関係なく、すべてのコンパイルのデータセット割り当てが行われます。の SYSDMDECK DD 割り当ては、一時データセットまたは永続的なカタログ化/非カタログ化データセットのいずれかとして指定できます。



デフォルトは、NOMDECK

省略形: NOMD | MD | MD(C | NOC)

オプションの指定:

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。

- OPTFILE (OPTFILE が PROCESS ステートメントや CBL ステートメントで指定されていない場合)
- JCL の PARM パラメーター
- として cob2 コマンドオプション
- インストール先デフォルト値として
- COBOPT 環境変数で

サブオプション:

- いつ MDECK(COMPILE)が有効な場合、ライブラリ処理と生成の後にコンパイルが正常に続行されます。MDECK 出力ファイルが完了しました。の設定 COMPILE|NOCOMPILE、DECK|NODECK、そして OBJECT|NOOBJECT コンパイラオプション。
- MDECK(NOCOMPILE) 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。コンパイラは、COMPILE、DECK、および OBJECT コンパイラ・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、EXEC CICS、EXEC SQL、または EXEC SQLIMS ステートメントを含むプログラムとともに使用する場合、これらの EXEC ステートメントは MDECK 出力にそのまま組み込まれます。ただし、SQL または SQLIMS オプション、対応する EXEC SQL INCLUDE または EXEC SQLIMS INCLUDE ステートメントは、MDECK 出力。

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・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。

関連タスク

[289 ページの『アセンブラー・プログラムからコンパイラーを開始する』](#)

[298 ページの『ライブラリー処理出力ファイル \(SYSMDECK\) の定義』](#)

関連参照

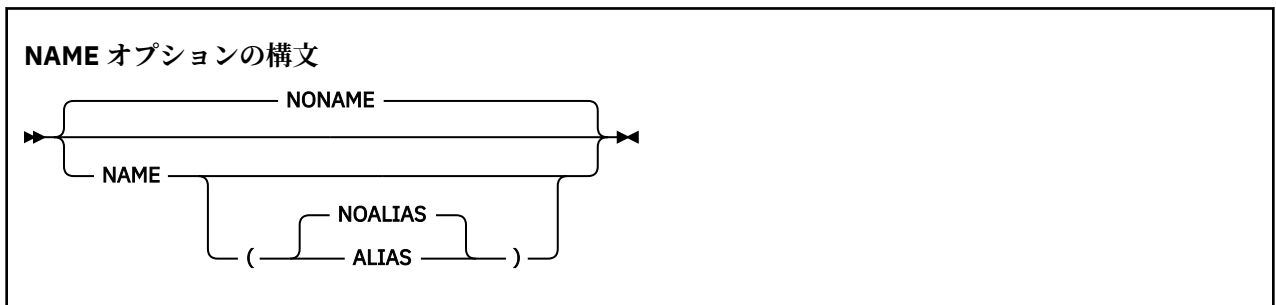
[376 ページの『矛盾するコンパイラー・オプション』](#)

[481 ページの『第 21 章 コンパイラー指示ステートメント』](#)

名前

NAME は、各オブジェクト・モジュールについてのリンク・エディット NAME カードを生成する場合に使用します。NAME は、バッチ・コンパイルを行う際に、各プログラム・オブジェクトの名前を生成する目的でも使用できます。

NAME を指定すると、作成される各オブジェクト・モジュールに NAME カードが追加されます。プログラム・オブジェクト名は、PROGRAM-ID ステートメントからモジュール名を形成する際の規則を使用して作成されます。



デフォルト: NONAME、または NAME (NOALIAS) (NAME だけが指定された場合)

省略形: なし

NAME (ALIAS) を指定し、プログラムに ENTRY ステートメントが含まれている場合には、ENTRY ステートメントごとにリンク・エディット ALIAS カードが生成されます。

関連参照

PROGRAM-ID 段落 (*Enterprise COBOL for z/OS* 言語解説書)

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。



デフォルトは、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。
- 形式のリテラル N" . . . "または N' . . . ' 国語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。
- 形式のリテラル N". . ."または N'. . .' DBCS リテラルとして扱われます。

の NSYMBOL (DBCS) オプションは、以前のリリースとの互換性を提供します IBM COBOL、そして NSYMBOL (NATIONAL) オプションは、上記の言語要素の取り扱いを 2002 COBOL 標準この点について。

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。

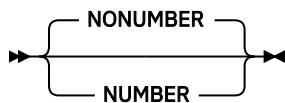
関連参照

376 ページの『矛盾するコンパイラー・オプション』

番号

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



デフォルトは、NONNUMBER

省略形は NUM | NONUM

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。

バッチコンパイルを行っている場合、NUMBER は実際には、バッチコンパイル内のすべてのプログラムは単一の入力ファイルとして扱われます。入力ファイル全体のシーケンス番号は昇順でなければなりません。

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。

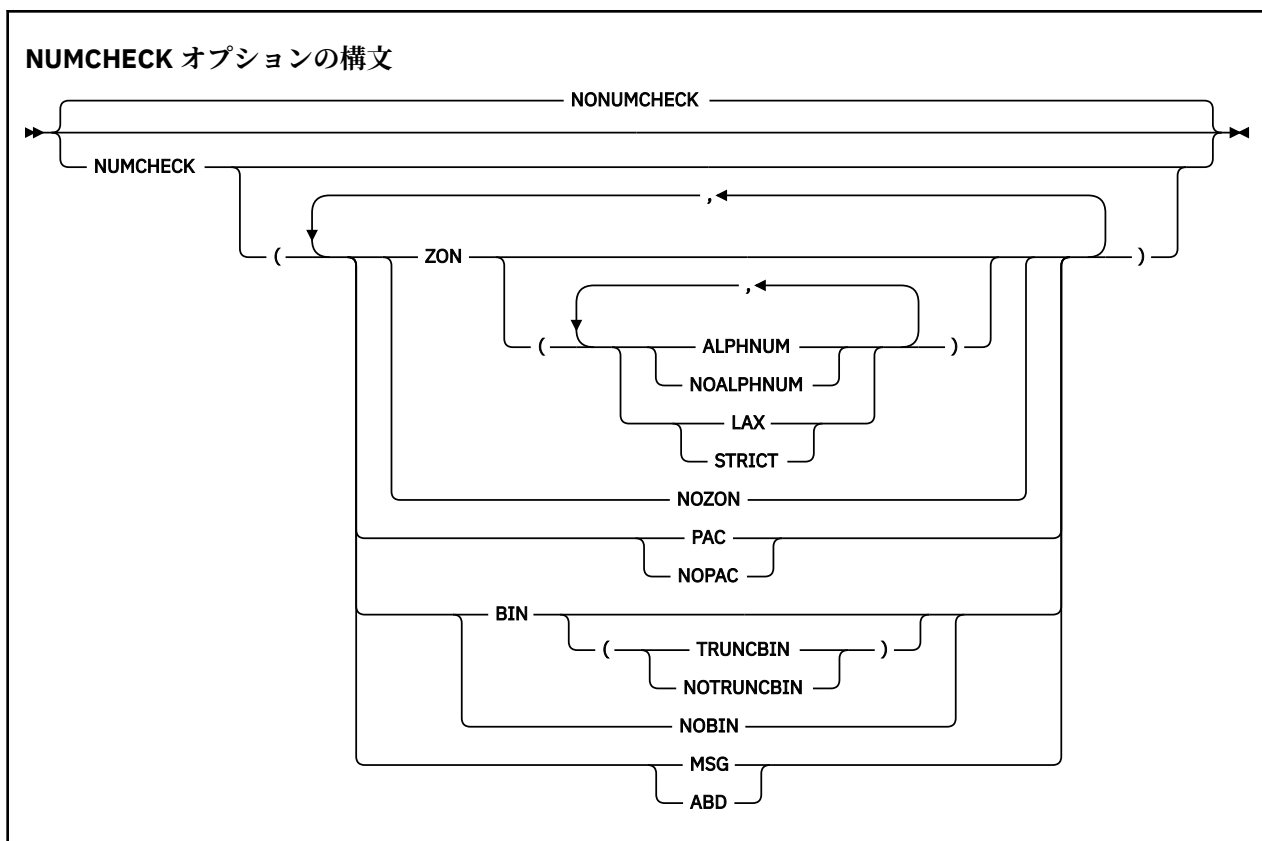
NUMCHECK

NUMCHECK コンパイラー・オプションは、データ項目が送信データ項目として使用されるときに、データ項目を検証するための追加コードを生成するかどうかをコンパイラーに指示します。ゾーン 10 進数 (数値 USAGE DISPLAY) およびパック 10 進数 (COMP-3) データ項目の場合、コンパイラーは送り出しフィールドごとに暗黙の数値クラス・テストを生成します。内容が数値受信側に移動される英数字送信側の場合、コンパイラーは送信側を数値整数として扱うため、NUMCHECK は各英数字送信側に対して暗黙的な数値クラス・テストを生成します。バイナリー・データ項目の場合、コンパイラーは、データ項目の桁数が PICTURE 節で許可されている桁数より多いかどうかを確認するために、SIZE ERROR 検査を生成します。

NUMCHECK オプションが更新され、無効データの冗長検査が削除されました。これにより、ランタイム・パフォーマンスが向上します。ランタイム・メッセージは以前より減っている可能性があります。

冗長な検査を除去するために行われる分析は、OPT(1|2)ではOPT(0)よりも関係があります。OPT(0)は、コンパイル時間をできるだけ短くするために、より簡単な形式の分析を行います。より高いOPTレベルでは、メッセージの数が少なくなる可能性があります。

検査によって常に無効データが見つかり、コンパイラがコンパイル時に判断できる場合は、コンパイル時メッセージが生成され、実行時検査は取り除かれます。(下記のMSG|ABDを参照してください。)



デフォルトは、NONUMCHECK

サブオプションのデフォルトは以下のとおりです。

- サブオプションを指定しない場合、デフォルトは ZON(ALPHNUM,STRICT)、PAC、BIN(TRUNCBIN)、および MSG です。例えば、NUMCHECK は NUMCHECK(ZON(ALPHNUM,STRICT),PAC,BIN(TRUNCBIN),MSG) と同じ効果があります。
- データ・タイプ・サブオプションが指定されていない場合、デフォルトのデータ・タイプ・サブオプションは ZON(ALPHNUM,STRICT)、PAC、および BIN(TRUNCBIN) です。例えば、NUMCHECK(ABD) は NUMCHECK(ZON(ALPHNUM,STRICT),PAC,BIN(TRUNCBIN),ABD) と同じ効果があります。
- データ・タイプ・サブオプションが1つしか指定されていない場合、デフォルトは NOZON、NOPAC、NOBIN、および MSG です。例えば、NUMCHECK(BIN) は NUMCHECK(NOZON,NOPAC,BIN(TRUNCBIN),MSG) と同じ効果があります。
- すべてのデータ・タイプ・サブオプションが NO で指定されている場合、リストには NONUMCHECK が表示されます。例えば、NUMCHECK(NOZON,NOPAC,NOBIN) は NONUMCHECK と同じ効果があります。

省略形は NONC | NC

ZON(ALPHNUM|NOALPHNUM,LAX|STRICT) | NOZON

ZON がサブオプションなしで指定されている場合、デフォルトは ZON(ALPHNUM,STRICT) です。

ZON(ALPHNUM)を指定すると、コンパイラは、COBOL ステートメントで送信データ項目として使用されるゾーン 10 進数 (数値 USAGE DISPLAY) データ項目の暗黙数値クラス・テスト用のコードを生成します。

ZON(NOALPHNUM)を指定すると、コンパイラーは、COBOL ステートメントで送信データ項目として使用されるゾーン 10 進数 (数値 USAGE DISPLAY) データ項目の暗黙的な数値クラス・テスト用のコードを生成します。ただし、英数字データ項目、英数字リテラル、または英数字形象定数との比較で 사용되는場合は除きます。

以下のサンプル・ステートメント内のデータ項目 B のように、受信側が送信側と受信側の両方を兼ねていない限り、受信側はチェックされません。

```
ADD A TO B
```

```
DIVIDE A INTO B
```

```
COMPUTE B = A + B
```

```
INITIALIZE B REPLACING ALPHANUMERIC BY B
```

このチェックは、各ステートメント内でデータが使用される前に実行されます。

- データが NOT NUMERIC の場合、NUMCHECK(ZON,MSG)の警告メッセージまたは NUMCHECK(ZON,ABD)の終了メッセージのいずれかが発行されます。
- データが NUMERIC の場合、ステートメントの外部動作は NUMCHECK(NOZON)と同じですが、速度が遅くなる点が異なります。

ZON(LAX)を指定すると、コンパイラーはゾーン 10 進数データ項目内の無効データに対してより寛容になります。コンパイラーで考慮される 3 つのケースは、以下のとおりです。

- 符号なしゾーン 10 進数データ項目で、末尾オーバーパンチ符号付きのゾーン 10 進数データ項目を再定義していて、符号なし項目の最後のバイトが符号付き項目の最後のバイトとオーバーラップしている。この場合、符号なし再定義項目は、NUMCHECK 検査のために符号付きゾーン 10 進数項目として扱われます。

注：

- 再定義される符号付きゾーン 10 進数項目は、レベル 01 項目またはレベル 77 項目である必要があります。符号なしゾーン 10 進数項目には、レベル 01 項目またはレベル 77 項目、あるいはグループ内の従属項目を指定できます。
- 符号なしゾーン 10 進数項目が、符号付きゾーン 10 進数項目全体とオーバーラップする必要はありません。各項目の最後のバイトさえオーバーラップしていれば十分です。例:

```
01 NUM1 PIC S9(8).  
01 NUM2 REDEFINES NUM1.  
    03 NUM2-PART1 PIC 9(4).  
    03 NUM2-PART2 PIC 9(2).  
    03 NUM2-PART3 PIC 9(2).
```

この場合、データ項目 NUM2-PART3 は NUMCHECK によって符号付きゾーン 10 進数データ項目として扱われます。これは、符号付き末尾オーバーパンチ・ゾーン 10 進数項目であるその最後のバイトが NUM1 の最後のバイトとオーバーラップするためです。したがって、NUM2-PART3 の以下の値はすべて有効と見なされます。

- x'F1F2F3F4F5F6F7F8'
- x'F1F2F3F4F5F6F7C8'
- x'F1F2F3F4F5F6F7D8'

- ゾーン 10 進数データ項目は、数字編集項目の PICTURE スtring の Z 記号で示されるように、先行スペースを含むことができる数字編集データ項目を再定義します。ゾーン 10 進数データ項目の先行バイトは、数字編集項目の先行バイトの一部またはすべてをオーバーラップします。この場合、NUMCHECK は、スペースを許可する数字編集項目のバイトとオーバーラップする、ゾーン 10 進数データ項目の先行バイトのスペースを許容します。

注:

- 再定義される数字編集項目は、レベル 01 項目またはレベル 77 項目である必要があります。ゾーン 10 進数項目には、レベル 01 項目またはレベル 77 項目、あるいはグループ内の従属項目を指定できます。
- ゾーン 10 進数項目に符号を付ける場合は、末尾オーバーパンチ符号にする必要があります。
- この処理の対象と見なされるには、ゾーン 10 進数項目の最初のバイトが数字編集項目の最初のバイトとオーバーラップする必要があります。ゾーン 10 進数項目が数字編集項目全体とオーバーラップする必要はありません。例:

```
01 NUMED PIC ZZ99.99.  
01 NUM REDEFINES NUMED.  
   03 INTVAL PIC 9(4).  
   03 FILLER PIC X.  
   03 DECVAL PIC 9(2).
```

この場合、NUMCHECK は INTVAL の最初の 2 バイトのスペースを許容します。これは、PICTURE スtringで Z 記号を使用して定義されている NUMED の最初の 2 バイトとオーバーラップするためです。したがって、INTVAL の以下の値はすべて有効と見なされます。

- x'F1F2F3F4'
- x'40F1F2F3'
- x'4040F1F2'

パフォーマンス上の理由から、先頭バイトにスペースと非スペースが混在することは許容されるので、x'F140F1F2' も有効と見なされることに注意してください。

- ゾーン 10 進数データ項目はゾーン 10 進数に移動されますまたは英数字データ項目。この場合、NUMCHECK は移動の送信側を検査しません。ただし、その後、送信側が数値コンテキストで使用される場合は、チェック・マークが付けられます。

注: もし ZON(LAX)指定され、NUMPROC(PFD)指定できない場合があります。

ZON(STRICT)が指定されている場合、NUMCHECK は、ゾーン 10 進数データ項目が再定義する可能性があるデータ項目を考慮せず、ゾーン 10 進数データの厳密な検査は通常どおり実行されます。

注: ZON(LAXREDEF|STRICTREDEF)オプションは非推奨ですが、互換性のために許容されており、ZON(LAX|STRICT)オプションに置き換えられています。

PAC | NOPAC

PAC を指定すると、コンパイラーは、COBOL ステートメントで送信データ項目として使用されるパック 10 進数 (COMP-3) データ項目の暗黙数値クラス・テスト用のコードを生成します。偶数桁を持つパック 10 進数データ項目に対して、未使用のビットで 1 の有無が検査されます。

制約事項: CALL ステートメントの場合、NUMCHECK(ZON)および NUMCHECK(PAC)は、ゾーン 10 進数またはパック 10 進数のデータ項目を検査しますが、BY REFERENCE パラメーターは検査しません。BY CONTENT (BY VALUE 句には、ゾーン 10 進数データ項目もパック 10 進数データ項目も指定できません。)

BIN(TRUNCBIN|NOTRUNCBIN) | NOBIN

BIN を指定すると、コンパイラーは ON SIZE ERROR のようなコードを生成して、バイナリー・データ項目の内容が PICTURE 節より大きいかどうかをテストします。この追加コードは、送信データ項目として使用されるバイナリー・データ項目に対してのみ生成され、COMP-5 データ項目はこの ON SIZE ERROR コードを生成しません。

BIN(TRUNCBIN)が有効な場合は、TRUNC(BIN)コンパイラー・オプションが有効な場合でも、バイナリー・データ項目の検査コードが生成されます。BIN のサブオプションが指定されていない場合は、BIN(TRUNCBIN)がデフォルトであることに注意してください。

BIN(NOTRUNCBIN)が有効になっている場合、TRUNC(BIN)コンパイラー・オプションが有効になっていると、バイナリー項目の検査コードは生成されません。

注: BIN(NOTRUNCBIN)は、NUMCHECK(..., BIN, ...)をデフォルト・オプションで固定オプションにしたいが、有効な TRUNC(BIN)オプションを指定してコンパイルされたモジュールに対しては検査を行わない場合に役立ちます。

MSG | ABD

これは、無効データに対して発行されるメッセージが警告レベル(処理は続行される)のメッセージなのか強制終了レベル(異常終了が引き起こされる)のメッセージなのかを判別します。

- MSG が有効になっている場合、行番号、データ項目名、データ項目内容、およびプログラム名を含むランタイム警告メッセージが発行されます。また、影響を受けるステートメントは引き続き実行されます。
- ABD が有効になっている場合は、異常終了の原因となる終了メッセージが発行されます。

コンパイル時に、チェックが常に無効なデータを検出することをコンパイラーが判別できる場合は、MSG または ABD が有効であるかどうかに関係なく、コンパイル時のエラー・レベル・メッセージが生成され、チェックは除去されます。

パフォーマンスの考慮事項。 NUMCHECK は、COBOL プログラムで使用されるゾーン 10 進数(数値 USAGE DISPLAY) データ項目、パック 10 進数(COMP-3) データ項目、および 2 進データ項目の数によっては、NONUMCHECK よりはるかに遅くなります。

サービスが適用された COBOL 6.2 以降、NUMCHECK のパフォーマンスが向上しました。ただし、NONUMCHECK を指定するとパフォーマンスが最高になり、より高い OPT レベルでパフォーマンスが向上します。

注: ZONECHECK は非推奨ですが、互換性のために許容されます。これは NUMCHECK(ZON(ALPHNUM)) に置き換えられます。

関連タスク

60 ページの『非互換データの検査(数値のクラス・テスト)』

関連参照

426 ページの『NUMPROC』

454 ページの『TRUNC』

464 ページの『ZONECHECK』

408 ページの『INVDATA』

NUMPROC

内部 10 進数およびゾーン 10 進数データで非優先符号を使用することがある場合には常に NUMPROC(NOPFD) を使用してください。

NUMPROC オプションの構文

➡ NUMPROC(^{NOPFD} _{PFD}) ➡

デフォルトは、NUMPROC(NOPFD)

省略形: なし

コンパイラーは、任意の有効符号構成(X'A'、X'B'、X'C'、X'D'、X'E'、またはX'F')を受け入れます。ほとんどの場合の推奨オプションは NUMPROC(NOPFD) です。

注: もし NUMPROC(PFD)指定され、NUMCHECK(ZON(LAX))指定できない場合があります。

パフォーマンスに関する考慮事項: NUMPROC(PFD)内部 10 進数およびゾーン 10 進数データの処理パフォーマンスが向上します。ただし、このオプションは、数値データが以下の IBM システム標準に正確に従っている場合にのみ使用してください。

- **ゾーン 10 進数、符号なし:** 符号バイトの高位 4 ビットに X'F' が入ります。

- ・ **ゾーン 10 進数、符号付きオーバーパンチ:** 符号バイトの上位 4 ビットには、数値が正または 0 の場合は X'C' が、それ以外の場合は X'D' が入ります。
- ・ **ゾーン 10 進数、分離符号:** 分離符号は、文字「+」(数値が正または 0 の場合)、および「-」(数値がそれ以外の場合)を含みます。
- ・ **内部 10 進数、符号なし:** 下位バイトの下位 4 ビットには、X'F' が入ります。
- ・ **内部 10 進数、符号付き:** 下位バイトの下位 4 ビットには、その数値が正または 0 であれば X'C' が入り、そうでなければ X'D' が入ります。

COBOL 算術ステートメントによって作成されるデータは、上記の IBM システム標準に適合します。しかし、REDEFINES を使用したり、グループ移動を行うと、データが変更されて、この標準に適合しなくなることがあります。NUMPROC (PFD) を使用する場合は、グループ移動を行うのではなく、INITIALIZE ステートメントを使用してデータ・フィールドを初期化しなければなりません。

NUMPROC (PFD) を使用すると、数値データのクラス・テストに影響を与えることがあります。COBOL プログラムが PL/I または FORTRAN で書かれたプログラムを呼び出す場合は、NUMPROC (NOPFD) を使用する。

符号表現は、NUMPROC オプションだけでなく、NUMCLS インストール・オプションの影響も受けます。

関連タスク

60 ページの『[非互換データの検査 \(数値のクラス・テスト\)](#)』

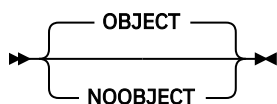
関連参照

59 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](#)』

OBJECT

OBJECT は、生成されたオブジェクト・コードを、バインダーへの入力として使用するファイルに書き込む場合に使用します。

OBJECT オプションの構文



デフォルトは、OBJECT

省略形は OBJ | NOOBJ

OBJECT を指定する場合は、コンパイル用の JCL の中に SYSLIN DD ステートメントを組み込んでください。

DECK と OBJECT の相違点は、出力の経路指定先となるデータ・セットのみです。

- ・ DECK 出力は、SYSPUNCH の DD 名に関連するデータ・セットに送られます。
- ・ OBJECT 出力は、SYSLIN の DD 名に関連するデータ・セットに送られます。

ご使用のシステムの指針に従ってオプションを使用してください。

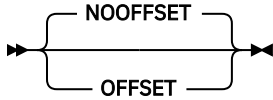
関連参照

376 ページの『[矛盾するコンパイラ・オプション](#)』

OFFSET

使用 OFFSET 凝縮された PROCEDURE DIVISION リスト。

OFFSET オプションの構文



デフォルトは、NOOFFSET

省略形は OFF | NOOFF

と OFFSET、凝縮された PROCEDURE DIVISION リストには、行番号、ステートメント参照、およびステートメントに対して生成された各命令ブロックの場所が含まれます。最適化プログラムは、段落をインライン化したりコードを移動したりすることがあります。実際に、最適化プログラムは、使用頻度が低いコード (エラー・メッセージ書式設定コードなど) をプログラム本体の後に置きます。その結果、特定のステートメントの OFFSET テーブルに複数の項目が存在する可能性があります。

次の項目も出力リストに書き込まれます。

- 定数域
- プログラム・プロログ域 (PPA1、PPA2、PPA3、PPA4)
- タイム・スタンプおよびコンパイラー・バージョン情報
- コンパイラー・オプションおよびプログラム情報
- ベース・ロケーター・テーブル
- 外部シンボル辞書
- 初期ヒープ・ストレージ・マップ
- スタック・ストレージ・マップ

注: エラーメッセージのフォーマットに使用されている行外コードのため、言語環境生成されたオフセットは、「コンパイル単位から{name}エン트리ポイントで{name}コンパイル単位オフセット{offset} ...」、プログラムのオフセット範囲外である可能性があります。このような場合は、COBOL メッセージのステートメント番号 (IGZnnnns) を参照して、問題を特定してください。

関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

[536 ページの『例: OFFSET コンパイラー出力』](#)

OPTFILE

OPTFILE は、データ・セットでの COBOL コンパイラー・オプションの指定を有効化するために使用します。コンパイラー・オプション・データ・セットを使用することによって、JCL PARM スtring に指定されたオプションの 100 文字制限を回避します。

OPTFILE オプションの構文

➡ OPTFILE ➡

デフォルト: なし

省略形: なし

OPTFILE は、コンパイラー呼び出しオプションとして、または COBOL ソース・プログラムの PROCESS または CBL ステートメントで、指定できます。OPTFILE は、インストール先デフォルトとしては指定できません。

OPTFILE コンパイル時に無視される cob2 コマンドの z/OS ユニックス環境。(その環境では、COBOPT 環境変数が、OPTFILE に同等の機能を提供します。)

OPTFILE が有効化されている場合には、コンパイラー・オプションは、SYSOPTF DD ステートメントで指定したデータ・セットから読み取られます。SYSOPTF データ・セットには、RECFM F または FB、および 80 バイトの LRECL が存在する必要があります。SYSOPTF データ・セットのフォーマットの詳細については、コンパイラー・オプション・データ・セットの定義に関する下記の関連タスクを参照してください。

SYSOPTF データ・セットのオプションの優先順位は、OPTFILE オプションを指定した場所で決まります。例えば、呼び出し PARM スtring で OPTFILE を指定した場合、PARM スtring で後から指定したオプションによって、競合する SYSOPTF データ・セットで指定されたすべてのオプションが置き換えられます。

(概念的には、オプション指定の OPTFILE は、SYSOPTF データ・セットにあるオプションで置き換えられます。それから、コンパイラー・オプションおよび競合するコンパイラー・オプションの優先順位に関する規則が適用されます。)

アセンブラー・プログラム内から COBOL コンパイラーを始動した場合、SYSOPTF でコンパイラー・オプション・データ・セットを指定する代わりに、代替 DD 名リストを使用して、使用する DD 名を指定することができます。

関連タスク

[289 ページの『アセンブラー・プログラムからコンパイラーを開始する』](#)

[295 ページの『コンパイラー・オプション・データ・セットの定義 \(SYSOPTF\)』](#)

[299 ページの『z/OS のもとでのコンパイラー・オプションの指定』](#)

[311 ページの『第 16 章 z/OS ユニックス のもとでのコンパイル』](#)

関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

OPTIMIZE

OPTIMIZE を使用して、オブジェクト・プログラムのランタイムを削減します。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。

OPTIMIZE オプションの構文

➡ OPTIMIZE — ({ 0
1
2 }) ➡

デフォルトは、OPTIMIZE(0)

省略形は、OPT(0)、OPT(1)、または OPT(2)

最適化は、プログラムおよびデータが有効であり、コンパイラー・オプションが指定されているという想定のもとで行われます。例えば、USAGE DISPLAY を持つ外部 10 進データは、ZONEDATA(MIG | NOPFD) を使用して無効なゾーン・ビットを許可しない限り、有効でなければなりません。どのようなオプションが使用されるとしても、数字と符号コードは有効でなければなりません。プログラムまたはデータが無効である場合、プログラムの動作は、最適化のレベルごとに、または Enterprise COBOL のバージョンごとに異なる可能性があります。

- OPTIMIZE(0) 制限付きの最適化が指定されます。結果として、コンパイル時間が最も短くなります。TEST オプションを指定すると、完全なデバッグ機能が使用可能になります。
- OPTIMIZE(1) アプリケーション・ランタイム・パフォーマンスの向上につながる最適化が指定されます。このレベルの最適化には、基本インライン化、強度の削減、複雑な演算から同等のより単純な演算への単純化、一部の到達不能コードの除去、ブロック再配置などがあります。また、OPTIMIZE(1) には、共通副次式の除去や値の伝搬など、いくつかのイントラブロック最適化も含まれています。TEST オプションを指定すると、ほとんどのデバッグ機能が使用可能になります。

- OPTIMIZE(2)さらなる最適化が指定されます。この最適化には、より積極的な単純化と命令スケジューリングが含まれます。また、グローバル値の伝搬およびループ不変コードの動作など、いくつかのブロック間最適化も含まれます。TEST オプションを指定すると、いくつかのデバッグ機能が使用可能になります。

OPTIMIZE(1)または OPTIMIZE(2)を TEST コンパイラー・オプションなしで使用する場合は、言語環境サービス CEEHDLR を介して登録されたユーザー作成条件ハンドラーに注意する必要があります。特に、条件ハンドラーが、条件ハンドラー・プログラム自体に対してローカルで定義されていないデータ項目 (例えば、アプリケーションで EXTERNAL として定義されているデータ項目) にアクセスする場合は、そのようなデータ項目を VOLATILE 節で定義して、ハンドラーがデータ項目の最新の値を使用するようにする必要があります。または、TEST コンパイラー・オプションを指定して条件処理プログラムをコンパイルすることもできます。VOLATILE 節の使用は、TEST オプションの使用よりも優先されます。これは、TEST オプションを使用すると、プログラム全体の最適化を削減できますが、VOLATILE は縮小された最適化をローカライズするためです。VOLATILE 節について詳しくは、『Enterprise COBOL for z/OS 言語解説書』の『VOLATILE 節』を参照してください。

注: 最良のパフォーマンスを得るために、IBM では OPT(2) コンパイラー・オプションを使用することを推奨します。

OPTIMIZE(1)または OPTIMIZE(2)が有効な場合、INLINE を指定すると、コンパイラーは PERFORM ステートメントのインライン化プロシーチャーを考慮します。詳細については、[407 ページの『INLINE』](#)を参照してください。

注: Enterprise COBOL 5 以降、NOOPTIMIZE、OPTIMIZE、OPTIMIZE(STD)、および OPTIMIZE(FULL) オプションは 削除済み ですが、互換性のために許容されています。これらのオプションの 1 つを指定した場合、そのオプションは以下のように新しいオプションにマップされます。

表 53. 削除されたオプションから新しいオプションへのマッピング	
削除されたオプション	新しいオプション
NOOPTIMIZE	OPTIMIZE(0)
OPTIMIZE	OPTIMIZE(2)
OPTIMIZE(STD)	OPTIMIZE(2)
OPTIMIZE(FULL)	OPTIMIZE(2)と STGOPT

関連概念

[843 ページの『最適化』](#)

関連タスク

[276 ページの『エラー処理用のルーチンの作成』](#)

関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

[407 ページの『INLINE』](#)

[419 ページの『MAXPCF』](#)

[448 ページの『TEST』](#)

[447 ページの『STGOPT』](#)

VOLATILE 文節 (Enterprise COBOL for z/OS 言語解説書)

OUTDD

OUTDD を使用して、システム論理出力装置に送られる DISPLAY 出力を特定の ddname に送りたいことを指定します。

OUTDD で指定された ddname を持つファイルを z/OS UNIX ファイルシステムで指定することができます。この DD 名が割り振られない場合の出力の送り先については、データの表示に関する関連タスクを参照してください。

OUTDD オプションの構文

➡ OUTDD(*ddname*) ⬅

デフォルトは、OUTDD(SYSOUT)

省略形: OUT

OUTDD コンパイラー・オプションと言語環境 MSGFILE ランタイム・オプションが同じ *ddname* (両方ともデフォルトは SYSOUT) を指定した場合、システム論理出力装置への DISPLAY 出力は、言語環境メッセージ機能を使用して書き込まれます。

注:

- CICS では、OUTDD オプションは無効です。
- OUTDD コンパイラー・オプションは、一連のプログラムの最初のプログラムに対してのみ指定できます (バッチ・コンパイル)。シーケンス内の最初のプログラム以外のプログラムに対して CBL または PROCESS ステートメントで OUTDD を指定すると、コンパイラー診断メッセージを受け取ります。

関連タスク

37 ページの『システム論理出力装置上でのデータの表示』

545 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

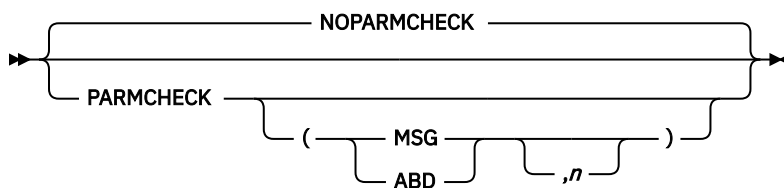
言語環境プログラミング・リファレンス (MSGFILE)

PARMCHECK

PARMCHECK オプションは、WORKING-STORAGE 内の最後の項目の後に追加のデータ項目を生成するようにコンパイラーに指示します。このバッファー・データ項目は、呼び出されたサブプログラムが WORKING-STORAGE の終わりを超えてデータを破壊したかどうかを検査するために、ランタイムに使用されます。

呼び出し側プログラムが PARMCHECK でコンパイルされると、コンパイラーは WORKING-STORAGE セクションの最後のデータ項目の後にバッファーを生成します。実行時に、各呼び出しの前にバッファーが ALL x'AA' に設定されます。各呼び出しの後には、バッファーが検査されて、バッファーが変更されたかどうかを確認されます。PARMCHECK オプションは、COBOL 4 以前のコンパイラーから COBOL 6 以降のコンパイラーへのマイグレーションに役立ちます。また、適切なプログラミング手法をクリーンアップして検査するためにも使用できます。

PARMCHECK オプションの構文



デフォルトは、NOPARMCHECK

省略形は NOPC | PC

サブオプションのデフォルトは以下のとおりです。

- サブオプションを指定しない場合、デフォルトは PARMCHECK(MSG,100) です。
- MSG または ABD のみが指定されている場合、デフォルトは PARMCHECK(MSG | ABD,100) です。例えば、PC(ABD) = PC(ABD,100) です。

- n のみが指定されている場合、デフォルトは `PARMCHECK(MSG, n)` です。例えば、`PC(5000) = PC(MSG, 5000)` です。

MSG | ABD

これは、サブプログラムによるデータ破壊に対して発行されるメッセージが警告レベル (処理は続行される) のメッセージなのか強制終了レベル (異常終了が引き起こされる) のメッセージなのかを判別します。

- MSG が有効になっている場合は、パラメーターの名前、CALL ステートメントの行番号、およびプログラム名を含むランタイム警告メッセージが発行されます。また、この検査は、影響を受ける CALL ステートメントが実行された後に行われます。
- ABD が有効になっている場合は、同様のメッセージが発行されますが、異常終了の原因となる終了レベルが示されます。

n

WORKING-STORAGE の最後の項目の後に追加されるバイト単位でのバッファのサイズです。1 から 9999 までの範囲にある整数でなければなりません。

パフォーマンスの考慮事項。 PARMCHECK を使用すると、コンパイラーは CALL ステートメントを使用するプログラムに対して低速コードを生成します。良好なパフォーマンスを得るには、NOPARMCHECK が有効になっている必要があります。

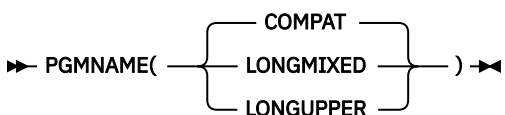
関連参照

CALL ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

PGMNAME

PGMNAME オプションは、プログラム名、エントリー・ポイント名、およびユーザー定義関数名の処理を制御します。

PGMNAME オプションの構文



デフォルトは、`PGMNAME (COMPAT)`

省略形は `PGMN (LM | LU | CO)`

LONGUPPER は、UPPER、LU、または U と省略することができます。LONGMIXED は、MIXED、LM、または M と省略することができます。

PGMNAME 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 ステートメント内のプログラム・エントリー・ポイント名。
- 以下におけるプログラム名参照:
 - ネストされたプログラム、静的にリンクされたプログラム、または DLL を参照する CALL ステートメント。
 - 静的にリンクされたプログラムまたは DLL を参照する SET プロシージャ・ポインターまたは関数ポインターステートメント。
 - ネストされたプログラムを参照する CANCEL ステートメント。
- AS 句が指定されていない場合は、FUNCTION-ID 段落で定義されたユーザー定義関数名。
- AS 句が指定されている場合は、ユーザー定義関数名の AS 句の *literal-1*。

注：ユーザー定義関数名が AS 句で指定されている場合、AS 句の *literal-1* は、ユーザー定義関数名自体ではなく、外部化された名前です。

PGMNAME(COMPAT)

PGMNAME(COMPAT)では、プログラム名およびユーザー定義関数名は旧バージョンの COBOL コンパイラと互換性のある方法で処理されます。

- 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、ハイフン、または下線でなければなりません。ただし、プログラム名がリテラルで、最外部のプログラムにある場合には、リテラルに拡張文字の @、#、および \$ を含めることが可能で、先頭文字を下線にすることができます。
- 少なくとも 1 文字は英字でなければなりません。
- 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。

外部プログラム名および外部ユーザー定義関数名は、コンパイラによって次のように処理されます。

- 大文字に変換される。
- 8 文字に切り捨てられる。
- ハイフンはゼロ (0) に変換される。
- 先頭文字が英字でなく、下線でない場合は、次のように変換される。
 - 1 から 9 は A から I に変換される。
 - その他はすべて J に変換される。

PGMNAME(LONGUPPER)

PGMNAME(LONGUPPER)では、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、または、FUNCTION-ID 段落で COBOL ユーザー定義語として指定されているユーザー定義関数名は、ユーザー定義語を形成するための通常の COBOL 規則に従う必要があります。

- 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、ハイフン、または下線でなければなりません。
- 少なくとも 1 文字は英字でなければなりません。
- 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。
- 下線を先頭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、ハイフン、または下線でなければなりません。
- 少なくとも 1 文字は英字でなければなりません。
- 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。
- 下線はどの位置でも使用できます。

外部プログラム名および外部ユーザー定義関数名は、コンパイラによって次のように処理されます。

- 大文字に変換される。
- ハイフンはゼロ (0) に変換される。
- 先頭文字が英字でなく、下線でない場合は、次のように変換される。
 - 1 から 9 は A から I に変換される。
 - その他はすべて J に変換される。

ネストされたプログラムの名前はコンパイラによって大文字に変換されますが、それ以外はそのまま処理され、切り捨てても変換も行われません。

PGMNAME(LONGMIXED)

PGMNAME(LONGMIXED)では、プログラム名は切り捨て、変換、または大文字への変換なしに、そのまま処理されます。

8文字より長い PROGRAM-ID を使用する場合は、単一引用符またはアポストロフィ (') で囲んで、PGMNAME(LONGMIXED)を使用します。

PGMNAME(LONGMIXED)では、すべてのプログラム名定義は、PROGRAM-ID 段落または ENTRY ステートメントでプログラム名のリテラル形式を使用して指定する必要があります。プログラム名のリテラル・ユーザーは、X'41'から X'FE'の範囲内の任意の文字を含むことができます。

PGMNAME(LONGMIXED)では、すべてのユーザー定義関数定義は、FUNCTION-ID 段落の AS 句を使用して指定する必要があります。AS 句の *literal-1* は、切り捨て、変換、または大文字への変換なしに、そのまま処理されます。

使用上の注意

- 以下のエレメントは、PGMNAME オプションの影響を受けません。
 - クラス名およびメソッド名。
 - システム名 (SELECT . . . ASSIGN の場合は割り当て名、COPY ステートメントの場合はテキスト名またはライブラリー名)。
 - 動的呼び出し。
動的呼び出しが解決される際、プログラム名は 8 文字に切り詰められ、大文字に変換されて、埋め込まれたハイフンや先行桁は変換されます。
 - ネストされていないプログラムの CANCEL。ネーム・レゾリューションは動的呼び出しのメカニズムと同じものを使用します。
- PGMNAME(LONGUPPER)または PGMNAME(LONGMIXED)オプションを指定してコンパイルされたリンク・エディットに関する考慮事項、COBOL プログラムは、AMODE 31 でリンク・エディットする必要があります。
- プログラム名が 8 バイト以下で、すべて大文字でない限り、PGMNAME(LONGMIXED)オプションまたは PGMNAME(LONGUPPER)オプションを使用してコンパイルされた COBOL プログラムに対して動的呼び出しを行うことはできません。さらに、プログラムの名前はそれが入っているモジュールの名前に対して固有でなければなりません。
- PGMNAME(LONGMIXED)でサポートされる拡張文字セットを使用する場合は、名前の解決に使用されるメカニズムに応じて、バインダー (リンケージ・エディター) または適用されるシステム規則に準拠した名前を使用するようにしてください。

コンマや括弧などの文字は使用しないでください。これらの文字は、バインダー (リンケージ・エディター) の制御ステートメントの構文で使用されます。

関連参照

PROGRAM-ID 段落 (*Enterprise COBOL for z/OS 言語解説書*)

FUNCTION-ID 段落 (*Enterprise COBOL for z/OS 言語解説書*)

QUALIFY

QUALIFY は、修飾の規則に作用し、COBOL 標準規則の下で参照できない一部のデータ項目を参照できるように修飾規則を拡張するかどうかを制御します。

QUALIFY オプションの構文

► QUALIFY() ◄

デフォルトは、QUALIFY(COMPAT)

省略形: QUA(C | E)

QUALIFY (COMPAT)

QUALIFY (COMPAT) が有効である場合、データ項目に対する参照は固有でなければなりません。

QUALIFY (EXTEND)

QUALIFY (EXTEND) が有効になっている場合は、COBOL 標準規則では固有でない一部の参照を固有にできるように、修飾規則が拡張されます。名前を含む階層内のレベルがすべて指定されている場合、修飾子のセットは修飾子の完全セットと呼ばれます。特定の修飾子の完全セットを持つデータ項目が 1 つしかない場合、参照はそのデータ項目に解決されます。これは、同じ修飾子のセットが、修飾子の不完全セットとして別の参照に一致する場合も同様です。

例

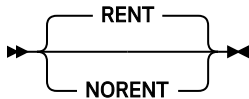
```
01 A.  
  02 B.  
    03 C PIC X.  
    03 A PIC X.  
  02 C PIC X.  
.  
.  
.  
  Move space to C of A      *> Refers to 02 level C (unique only with  
QUALIFY(EXTEND))  
  Move space to A::C        *> Refers to 02 level C (unique only with  
QUALIFY(EXTEND))  
  Move space to A           *> Refers to 01 level A (unique only with  
QUALIFY(EXTEND))  
  Move space to C of B of A *> Refers to 03 level C (unique by COBOL standard  
rules)  
  Move space to A::B::C     *> Refers to 03 level C (unique by COBOL standard  
rules)  
  Move space to C of B      *> Refers to 03 level C (unique by COBOL standard  
rules)  
  Move space to B::C        *> Refers to 03 level C (unique by COBOL standard  
rules)
```

RENT

RENT としてコンパイルされたプログラムは再入可能オブジェクト・プログラムとして生成されます。次のようにコンパイルされたプログラム NORENT 再入不可能なオブジェクト プログラムとして生成されます。

再入可能プログラムまたは再入不可プログラムは、メインプログラムまたはサブプログラムとして呼び出すことができます。

RENT オプションの構文



デフォルトは、RENT

省略形: なし

DATA および RMODE 設定: RENT オプションは、ストレージおよびアドレス可能度に影響を与える他のコンパイラ・オプションと相互作用します。使用 DATA (24 | 31) 動的データ領域を無制限のストレージに割り当てるか、16 MB 未満から取得したストレージに割り当てるかを制御する、再入可能プログラムのオプション。プログラムが 16 MB より上の仮想記憶域アドレスで実行される場合は、プログラムを RENT でコンパイルしてください。

16 MB より上での再入不可プログラムの実行はサポートされていません。NORENT でコンパイルされるプログラムは RMODE 24 でなければなりません。

の設定 DATA オプションは、コンパイルされたプログラムには影響しません。NORENT。

再入可能にする必要がある Enterprise COBOL プログラムについては、プログラムの再入可能化に関する関連タスクを参照してください。

リンク・エディットに関する考慮事項: プログラム・オブジェクト内のすべてのプログラムを RENT を指定してコンパイルする場合は、プログラム・オブジェクトを RENT バインダー (リンケージ・エディター) オプションを指定してリンク・エディットすることをお勧めします。プログラム・オブジェクトに、逐次でのみ再使用が可能な非 COBOL プログラムも含まれる場合は、代わりに REUS バインダー (リンケージ・エディター) オプションを使用してください。

もし、プログラムオブジェクト再入可能ではない、のプログラムオブジェクトリンク編集してはならない RENT または REUS リンク編集属性。CANCEL ステートメントが後続の CALL でプログラムの最新コピーを保証するようにするには、NOREUS バインダー (リンケージ・エディター) オプションが必要です。

AMODE 64 の考慮事項: LP (64) コンパイラオプションは RENT。ユーザーが NORENT を明示的に指定した場合は、通知メッセージが発行され、この設定は無視されます。

関連概念

[40 ページの『ストレージとそのアドレス可能性』](#)

関連タスク

[620 ページの『プログラムを再入可能にする』](#)

DB2 アプリケーション・プログラミングおよび SQL ガイド (再入可能コードを使用)

関連参照

[376 ページの『矛盾するコンパイラ・オプション』](#)

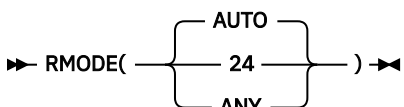
[391 ページの『データ』](#)

[436 ページの『RMODE』](#)

RMODE

RMODE の設定は、生成されたオブジェクト・プログラムの RMODE (常駐モード) にも影響を与えます。

RMODE オプションの構文



デフォルトは、AUTO

省略形: なし

RMODE (AUTO) オプションでコンパイルされたプログラムは、NORENT が指定された場合は RMODE 24 になり、RENT が指定された場合は RMODE ANY になります。RMODE AUTO は、VS COBOL II などの古いコンパイラと互換性があります。これらのコンパイラは、NORENT でコンパイルされたプログラムに対しては RMODE 24 を生成し、RENT でコンパイルされたプログラムに対しては RMODE ANY を生成していました。

コンパイルされたプログラムは RMODE (24) オプションには RMODE 24 かどうか NORENT または RENT が指定されます。

RMODE (ANY) オプションを使用してコンパイルするプログラムは、コンパイル時に RENT オプションも使用する必要があります。このプログラムには、RMODE ANY 属性。

NORENT オプションを指定した場合は、RMODE (24) または RMODE (AUTO) コンパイラ・オプションを指定する必要があります。バインダー・オプションまたは制御ステートメントによるモジュール RMODE のオーバーライドはサポートされていません。

DATA および RENT: RMODE オプションは、ストレージおよびアドレス可能度に影響を与える他のコンパイラー・オプションおよびランタイム・オプションにも相互作用します。モードの異なるプログラム間でのデータの受け渡しに関する情報については、ストレージとそのアドレス可能度に関する関連概念を参照してください。

リンク・エディットに関する考慮事項: COBOL が生成するオブジェクト・コードに属性 RMODE 24 がある場合には、k コードを RMODE 24 モジュールにリンク・エディットする必要があります。COBOL が生成するオブジェクトコードに次の属性がある場合 RMODE ANY コードをリンク編集して、RMODE ANY または RMODE 24 モジュール。

AMODE 64 の考慮事項: の LP(64) コンパイラオプションは RMODE(ANY)。この設定は、ユーザーが RMODE(24) を明示的に指定すると、通知メッセージが出されて無視されます。

関連概念

40 ページの『ストレージとそのアドレス可能度』

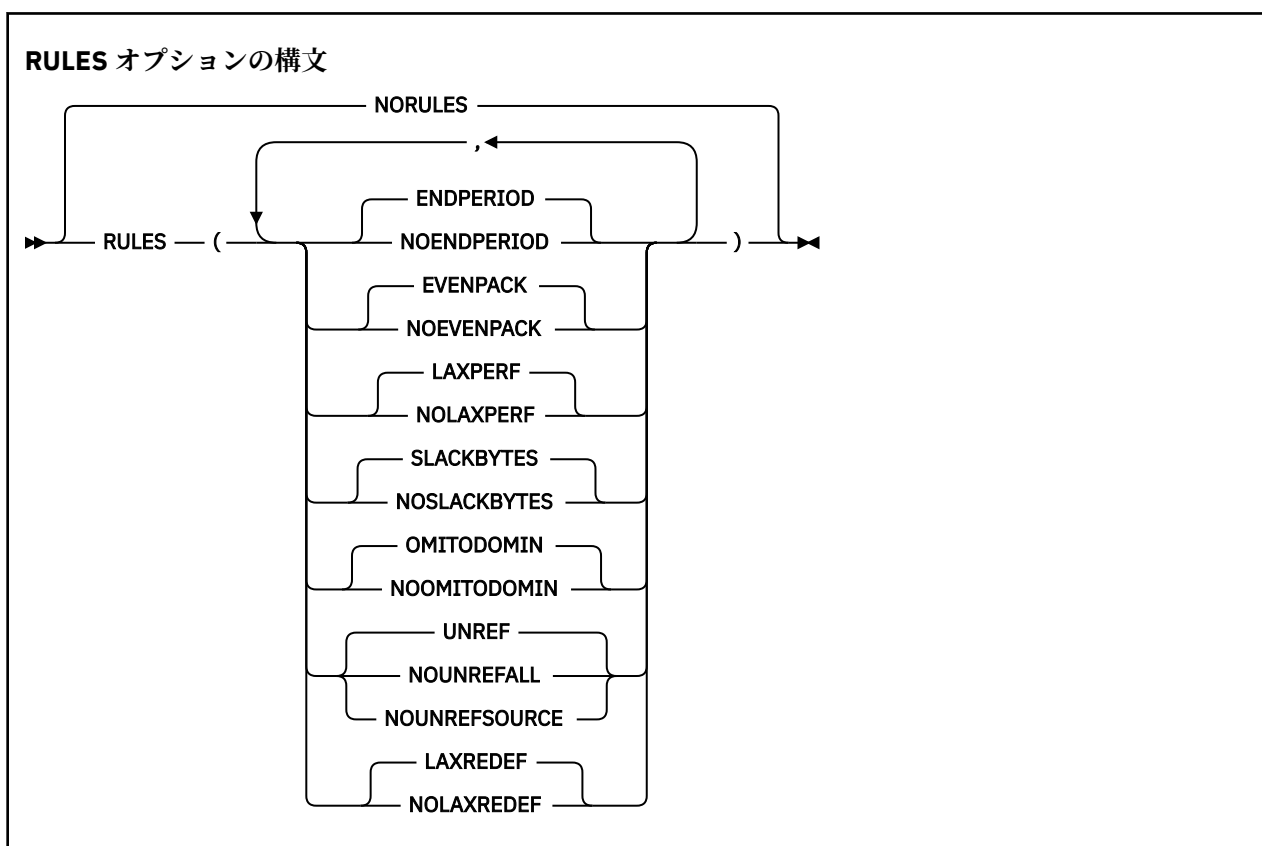
関連参照

197 ページの『QSAM ファイル用のバッファの割り振り』

376 ページの『矛盾するコンパイラー・オプション』

RULES

RULES オプションを使用すると、コンパイル時に特定タイプのソース・コードにフラグを立てることにより、プログラムを改善するためにプログラムに関する情報をコンパイラーから要求できます。



デフォルトは、NORULES

省略形:

- ENDP = ENDPERIOD
- EVENP = EVENPACK
- LXPRF = LAXPERF

- SLCKB = SLACKBYTES
- OOM = OMITODOMIN
- NOUNRA = NOUNREFALL
- NOUNRS = NOUNREFSOURCE
- LXRDF=LAXREDEF

RULES には、以下のサブオプションを指定できます。

ENDPERIOD | NOENDPERIOD

デフォルトは ENDPERIOD です。NOENDPERIOD を指定すると、条件ステートメントの範囲が明示的な範囲終了符号 END-★ではなくピリオドで終了した場合に、コンパイラーから警告メッセージが出されます。

EVENPACK | NOEVENPACK

デフォルトは EVENPACK です。NOEVENPACK を指定すると、コンパイラーでは、桁数が偶数であるすべての USAGE PACKED-DECIMAL (COMP-3) データ項目に対して警告メッセージが発行されます。これは、未使用ビットがゼロ以外のデータ項目が原因で、プログラムの動作が予想しないものになる可能性があるためです。

注：

- RULES (NOEVENPACK) は、未使用の追加スペースが予約されている USAGE PACKED-DECIMAL (COMP-3) データ項目を識別する際に役立ちます。ただし、これらのデータ項目を奇数桁になるように変更する必要はありません。これは単に、やや優れたプログラミング方法にすぎません。
- データ項目の名前が DFH、DSN、EYU、または SQL で始まる場合 (これらは CICS および Db2 用に、またはこれらによって生成されたデータ項目です)、コンパイラーは、偶数桁 PACKED-DECIMAL データ項目についてはメッセージを出しません。

LAXPERF | NOLAXPERF

デフォルトは LAXPERF です。NOLAXPERF サブオプションを指定すると、非効率的な COBOL 機能の使用に関する警告メッセージがコンパイラーから出されます。これらの機能には、算術ステートメントでの USAGE DISPLAY 数値データ項目、MOVE ステートメント内の大量のスペース埋め込み、非効率的なコンパイラー・オプションなどがあります。

SLACKBYTES | NOSLACKBYTES

デフォルトは SLACKBYTES です。NOSLACKBYTES を指定すると、コンパイラーによって遊びバイト (レコード内の遊びバイトまたはレコード間の遊びバイト) が追加される原因となるすべての SYNCHRONIZED データ項目に対して、コンパイラーから警告メッセージが出されます。遊びバイトの追加を発生させるデータ項目はそれぞれ、コンパイラー診断を受け取ります。

OMITODOMIN | NOOMITODOMIN

デフォルトは OMITODOMIN です。NOOMITODOMIN を指定すると、integer-1 (最小出現回数) なしで指定されたすべての OCCURS DEPENDING ON 節に対して、コンパイラーから警告メッセージが出されます。

OCCURS DEPENDING ON 節について詳しくは、「Enterprise COBOL for z/OS 言語解説書」の『可変長テーブル』を参照してください。

UNREF | NOUNREFALL | NOUNREFSOURCE

デフォルトは UNREF です。これは、参照されていないデータ項目のレポートは作成されないことを意味します。

NOUNREFALL を指定すると、データ項目の定義がユーザー・ソース・プログラムに直接表示されるか、COPY メンバーからプログラムに組み込まれたかに関係なく、FILE SECTION、WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION 内の参照されていない level-01 および level-77 のすべてのデータ項目 (項目がグループである場合に参照される従属項目を除いて) が報告されます。

NOUNREFSOURCE を指定すると、データ項目の定義がユーザー・ソース・プログラムに直接表示される場合にのみ、FILE SECTION、WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、および

LINKAGE SECTION 内の参照されていない level-01 および level-77 のすべてのデータ項目 (項目がグループであるときに参照される従属項目を除いて) が報告されます。

注:

- COBOL では、単一グループ項目の定義を、複数のファイルにわたって行うことができます。その際に、グループの level-01 データ項目の定義がメイン・ソース・ファイルにある場合、参照されていないそれらのデータ項目は、NOUNREFSOURCE が有効になっていれば報告されます。
- 名前接頭部 DFH、DSN、EYU、または SQL を持つデータ項目 (CICS および Db2 用に、またはこれらによって生成されたデータ項目) は、NOUNREFALL または NOUNREFSOURCE が有効になっていても報告されません。

LAXREDEF | NOLAXREDEF

デフォルトは LAXREDEF です。NOLAXREDEF を指定すると、データ項目がさらに小さい項目によって再定義される場合には、どのレベルであろうと警告メッセージがコンパイラから出されます。デフォルト・オプション LAXREDEF を使用すると、non-level-01 データ項目に対してのみ、そのような警告がコンパイラから出されます。

RULES オプションをサブオプションなしで指定すると、デフォルトは RULES(ENDPERIOD,EVENPACK,LAXPERF,SLACKBYTES,OMITODOMIN,UNREF,LAXREDEF) になります。

注:

- RULES のすべてのサブオプションを指定する必要はありません。サブオプションが指定されない場合は、そのサブオプションのデフォルト値が有効になります。例えば、RULES(NOENDP,NOSLCKB) と指定すると、RULES(NOENDP,EVENP,LXPRF,NOSLCKB,OMITODOMIN,UNREF,LXRDF) が有効になります。
- 1つのコンパイルに対して複数の RULES 指定がある場合、サブオプションは加算されます。これは、サブオプションが累積されることを意味します。例えば、RULES(ENDP) と指定すると、以前に指定した RULES サブオプションが何であれ、それだけではなく、RULES(ENDP) も有効になります。サブオプションの RULES 指定がない場合は、デフォルトが有効になります。後続の RULES 指定が以前に指定されたサブオプションを参照している場合は、最後の指定が有効になります。
- オプションで、RULES オプションを EXIT コンパイラ・オプションの MSGEXIT サブオプションとともに使用すると、ローカル・コーディング標準を適用することができます。例えば、プログラマーが条件ステートメントを区切る際に、明示的な範囲区切り文字の代わりにピリオドを使用できないようにしたい場合は、ENDPERIOD メッセージの重大度を警告レベル (RC=4) から重大レベル (RC=12) に変更することができます。このメッセージおよびその他の RULES メッセージの重大度を変更する方法の例については、IGYMSGXT という名前の、SIGYSAMP 内のサンプル MSGEXIT を参照してください。

関連参照

SYNCHRONIZED 文節 (Enterprise COBOL for z/OS 言語解説書)

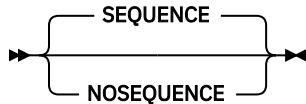
[447 ページの『STGOPT』](#)

SEQUENCE

SEQUENCE を使用すると、コンパイラは桁 1 から 6 を調べ、ソース・ステートメントが EBCDIC 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。

SEQUENCE オプションの構文



デフォルトは、SEQUENCE

省略形は SEQ | NOSEQ

SEQUENCE オプションを有効にして COPY ステートメントを使用する場合、ソース・プログラムのシーケンス・フィールドと、コピーブック・シーケンス・フィールドが対応している必要があります。

使用する場合、NUMBER そして SEQUENCE シーケンスは数値ではなく、EBCDIC、照合順序。

バッチ・コンパイルを行っており、SEQUENCE が有効である場合、バッチ・コンパイルのすべてのプログラムが1つの入力ファイルとして扱われます。入力ファイル全体のシーケンス番号は昇順でなければなりません。

この検査と診断メッセージを抑止する場合は、NOSEQUENCE を使用してください。

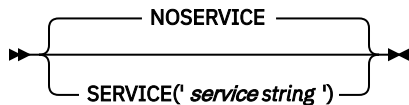
関連タスク

491 ページの『行シーケンス問題の検出』

サービス

SERVICE は、オブジェクト・モジュールが生成された場合に、オブジェクト・モジュール内にストリングを配置します。オブジェクト・モジュールがプログラム・オブジェクトにリンクされている場合、ストリングはこのプログラム・オブジェクトとともにメモリーにロードされます。Language Environment ダンプにトレースバックが含まれている場合は、このストリングがそのトレースバックに組み込まれます。

SERVICE オプションの構文



デフォルトは、NOSERVICE

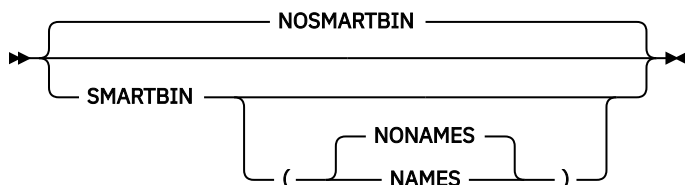
省略形は SERV | NOSERV

service string の長さは 64 文字に制限されています。

SMARTBIN

SMARTBIN を使用すると、以下の方法で最適化できるようにする追加のバイナリ・メタデータを含むモジュールを生成するよう、コンパイラに指示することができます。IBM 自動バイナリ・オプティマイザ (ABO) z/OS2.2 またはそれ以上に最適化されます。

SMARTBIN オプションの構文



デフォルトは、LP(32) が有効な場合は SMARTBIN、そうでない場合は NOSMARTBIN。

サブオプションなしで SMARTBIN が指定された場合、デフォルトは SMARTBIN(NONAMES)に変更される。

注：LP(64)が有効な場合、SMARTBIN はサポートされません。

注：NAMES サブオプションは、ABO 2.3 以上でサポートされている。

SMARTBIN が有効な場合、追加のバイナリ・メタデータが、モジュールの NOLOAD セグメントに配置されます。生成するには SMARTBIN メタデータのコンパイル時間は最大 21%増加する可能性がある。OPT(0) 2～3%OPT(1)そして OPT(2)に IBMz15[®]（またはそれ以降）のマシン zEnterprise データ圧縮 (zEDC) が有効（ハードウェア圧縮がオン）になります。これは、OPT(0) 10%で OPT(1)そして OPT(2)に IBMz15[™]（またはそれ以降の）マシン zEDC 有効（ハードウェア圧縮はオフ）。追加のメタデータによって、ディスク上のモジュールのサイズも増加するので、必要なロード・ライブラリーも大きくなります。ただし、プログラムの実行時は、ロードされないのメモリ・サイズが増加することはありません。ディスク上のサイズの増加は、元のバイナリのサイズの約 2 倍から 3 倍になります。

LP(32)が有効な場合は、SMARTBIN がデフォルトです。NOSMARTBIN ただし、バイナリ・メタデータを追加しないと、Enterprise COBOL 6.5 で構築された COBOL モジュールは ABO 最適化の対象外となり、ハードウェアの改善から最大限の利益を得るには、将来的にモジュールを再コンパイルしてテストする必要があります。IBM Z[®] ハードウェアの改善から最大限の利益を得るためには、将来的にモジュールを再コンパイルしてテストする必要があります。ABO を使用する場合、または将来使用する予定の場合は、SMARTBIN オプションをお勧めします。

SMARTBIN(NAMES) を指定すると、ユーザーシンボルの名前に関する情報が追加バイナリメタデータに追加される。SMARTBIN(NAMES) でコンパイルすると、*IBM watsonx Code Assistant for Z* コード最適化アドバイスの解析出力が向上します。SMARTBIN(NAMES) は ABO 最適化には必要ない。

注：

- IBM 自動バイナリ・オプティマイザ z/OS2.2 またはそれ以上の COBOL コンパイラで生成されたプログラム・モジュール内の CSECT を最適化できる：
 - Enterprise COBOL for z/OS 6
 - ABO 2.2 以降、新たに対象になりました。ABO 2.2 またはそれ以降では、SMARTBIN コンパイラ・オプションが有効な場合にのみ、COBOL 6.4 またはそれ以降のモジュールを最適化できる。ABO 2.2、またはそれ以上のバージョンでは、特別な COBOL コンパイラ・オプションを設定することなく、COBOL 6.1、6.2、6.3 モジュールを最適化することができる。
 - Enterprise COBOL for z/OS 5
 - ABO 2.2 以降、新たに対象になりました。ABO 2.2 またはそれ以上のバージョンでは、特別な COBOL コンパイラ・オプションを設定することなく、COBOL 5 モジュールを最適化することができる。
 - Enterprise COBOL for z/OS 4
 - Enterprise COBOL for z/OS 3
 - COBOL for OS/390 & VM 2
 - 多重仮想ストレージ & 仮想マシン M 1.2 のための COBOL
 - COBOL/370 1.1
 - VS COBOL II 1.4.0
 - VS COBOL II 1.3.x
- CA オプティマイザーによって処理された COBOL モジュールは、ABO では最適化できません。このようなタイプのモジュールの場合は、CA オプティマイザーで処理される前に、ABO を使用して、COBOL コンパイラで作成された元のモジュールを最適化することをお勧めします。

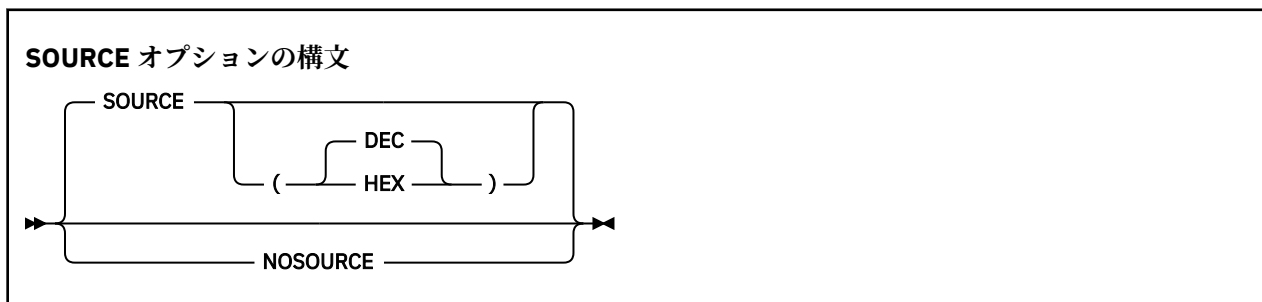
ABO の利点について詳しくは、[IBM Automatic Binary Optimizer for z/OS 製品ページ](#)を参照してください。

関連概念

[40 ページの『ストレージとそのアドレス可能性』](#)

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。



デフォルトは、SOURCE (DEC)

省略形は S | NOS

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。コンパイラーは、プログラムのソース・リストにシーケンス番号を生成します。

コンパイラー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。

DEC

SOURCE (DEC) が有効になっている場合、ソースのリストの行番号は 10 進形式になります。

サブオプションなしの SOURCE には、SOURCE (DEC) と同じ効果があります。

HEX

SOURCE (HEX) が有効な場合は、ソースのリストの行番号は 16 進形式になります。

SOURCE (HEX) は、コンパイラー呼び出しオプションとして、または COBOL ソース・プログラムの PROCESS または CBL ステートメントで、指定できます。

SOURCE (HEX) は、インストール先デフォルトとしては指定できません。

注：

- SOURCE (HEX) が有効な場合は、コンパイラー・オプション NUMBER および SEQUENCE を使用できません。
- SOURCE (HEX) を使用しても、ランタイム・メッセージでは 10 進数の行番号が表示されるので、ランタイム・メッセージのソースを探すときには、ユーザーがランタイム・メッセージ (または ABEND) の 10 進数の行番号を 16 進数に変換して、ソース・リストの該当行を見つける必要があります。

制限したい場合は SOURCE 出力、使用*CONTROL SOURCE または NOSOURCE あなたの PROCEDURE DIVISION。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。

以下の例は、診断メッセージが組み込まれたソースのリストを示しています。

```
00006C          Check-results section.
00006D          If ec = 0 then

== 00006D> IGYPS0086-I "EC" is reserved in ISO/IEC FCD 1989:2002, Programming language COBOL,
          and may be reserved in a future release of this compiler.
```

505 ページの『例: MAP 出力』

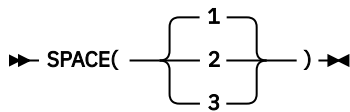
関連参照

*CONTROL (*CBL) ステートメント (*Enterprise COBOL for z/OS 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



デフォルトは、SPACE (1)

省略形: なし

SPACE が意味を持つのは、SOURCE コンパイラー・オプションが有効な場合だけです。

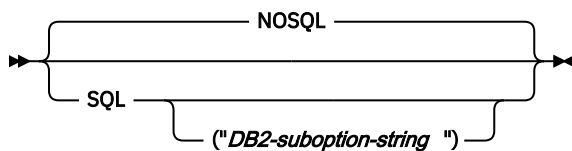
関連参照

[442 ページの『SOURCE』](#)

SQL

SQL コンパイラー・オプションを使用して、Db2 コプロセッサを使用可能にし、Db2 サブオプションを指定します。

SQL オプションの構文



デフォルトは、NOSQL

省略形: なし

COBOL ソース・プログラムに SQL ステートメント (EXEC SQL ステートメント)が含まれている場合、LP(64)が有効なときに SQL オプションを指定する必要がある、コンパイル時に Db2 コプロセッサがコンパイラーに対して使用可能になっている必要があります。Db2 プリコンパイラーは LP(64) ではサポートされません。

SQL オプションを使用すると、Db2 コプロセッサはデータベース要求モジュール (DBRM) を ddname DBRMLIB に書き込みます。

NOSQL オプションを指定すると、ソース・プログラムで検出されたすべての SQL ステートメントが診断され、破棄されます。

引用符またはアポストロフィのどちらかを使用して、Db2 サブオプションのストリングを区切ります。

複数の CBL ステートメントで、長いサブオプション・ストリングを複数のサブオプション・ストリングに分割することができます。例:

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL= 'SQL("string1")'
//COBOL.SYSIN DD *
CBL SQL("string2")
CBL SQL('string3')
IDENTIFICATION DIVISION.
PROGRAM-ID. DRIVER1.
. . .
```

それぞれの Db2 サブオプションは、指定された順に連結されます。そのため、上の例では、コンパイラーは次のサブオプション・ストリングを Db2 コプロセッサに渡します。

```
"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 Db2 オプションの複数インスタンスが見つかった場合は、各オプションで最後に指定されたものが使用されます。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。

注：SQL コンパイラー・オプションは、一連のプログラムの最初のプログラムに対してのみ指定できます (バッチ・コンパイル)。シーケンス内の最初のプログラム以外のプログラムに対して CBL または PROCESS ステートメントで SQL を指定すると、コンパイラー診断メッセージを受け取ります。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[561 ページの『SQL オプションを使用したコンパイル』](#)

[562 ページの『Db2 サブオプションの分離』](#)

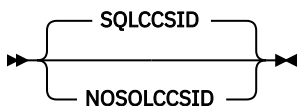
関連参照

[376 ページの『矛盾するコンパイラー・オプション』](#)

SQLCCSID

SQLCCSID コンパイラー・オプションは、CODEPAGE コンパイラー・オプションが COBOL プログラムの SQL ステートメントの処理に影響を与えるかどうかを制御するために使用します。

SQLCCSID オプションの構文



デフォルトは、SQLCCSID

省略形は SQLC | NOSQLC

LP(64) コンパイラー・オプションが有効な場合は、LP(32) と同じように動作する SQLCCSID オプションがサポートされます。

SQLCCSID オプションは、組み込みの Db2 コプロセッサ(SQL コンパイラー・オプション)を使用した場合にのみ有効です。

SQLCCSID が有効な場合、CODEPAGE コンパイラー・オプションの設定は、組み込みの Db2 コプロセッサを使用したとき、COBOL プログラム内の SQL ステートメントの処理に影響を与えます。

NOSQLCCSID が有効である場合:

1. CODEPAGE コンパイラー・オプションは、ストリング・リテラルのエンコード方式として、および変換済み SQL ステートメントが含まれる COBOL アプリケーション・ソースのエンコード方式としてのみ使用されます。
2. Db2 (文字ストリング) ホスト変数は、CODEPAGE コンパイラー・オプションの影響を受けません。代わりに、Db2 (文字ストリング) ホスト変数のエンコード方式は、DSNHDECP ファイルにある CCSID 値からもたらされます。つまり、Db2 は、Db2 データ (ホスト変数) のエンコード方式を DSNHDECP ファイルで決定します。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[564 ページの『SQLCCSID または NOSQLCCSID オプションを使用したプログラミング』](#)

関連参照

[563 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』](#)

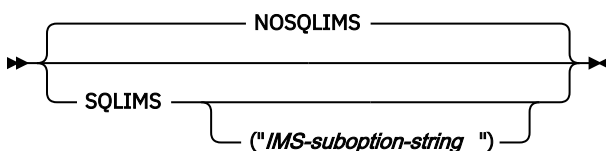
[385 ページの『CODEPAGE』](#)

[443 ページの『SQL』](#)

SQLIMS

SQLIMS コンパイラ・オプションを使用して、IMS SQL コプロセッサを使用可能にし、Information Management System (IMS) サブオプションを指定します。COBOL ソース・プログラムに SQLIMS ステートメント (EXEC SQLIMS ステートメント) が含まれている場合は、SQLIMS オプションを指定する必要があります。

SQLIMS オプションの構文



デフォルト。NOSQLIMS

省略形: なし

NOSQLIMS オプションを指定すると、ソース・プログラムで検出されたすべての SQLIMS ステートメントが診断され、破棄されます。

引用符またはアポストロフィのどちらかを使用して、IMS サブオプションのストリングを区切ります。

複数の CBL ステートメントで、長いサブオプション・ストリングを複数のサブオプション・ストリングに分割することができます。例:

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOLE='SQLIMS("string1")'
//COBOL.SYSIN DD *
  CBL SQLIMS("string2")
  CBL SQLIMS('string3')
  IDENTIFICATION DIVISION.
  PROGRAM-ID. DRIVER1.
  . . .
```

それぞれの IMS サブオプションは、指定された順に連結されます。そのため、上の例では、コンパイラは次のサブオプション・ストリングを IMS SQL コプロセッサに渡しています。

```
"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 IMS オプションのインスタンスが複数検出される場合は、各サブオプションで最後に指定されたものが有効となります。コンパイラは、連結 IMS 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。

LP(64)コンパイラ・オプションが有効な場合、SQLIMS オプションはサポートされません。ユーザーがこのオプションを明示的に指定すると、診断メッセージが出力されます。

注: SQLIMS コンパイラー・オプションは、一連のプログラムの最初のプログラムに対してのみ指定できます (バッチ・コンパイル)。シーケンス内の最初のプログラム以外のプログラムに対して CBL または PROCESS ステートメントで SQLIMS を指定すると、コンパイラー診断メッセージを受け取ります。

関連概念

569 ページの『IMS SQL コプロセッサ』

関連タスク

571 ページの『SQLIMS オプションを使用したコンパイル』

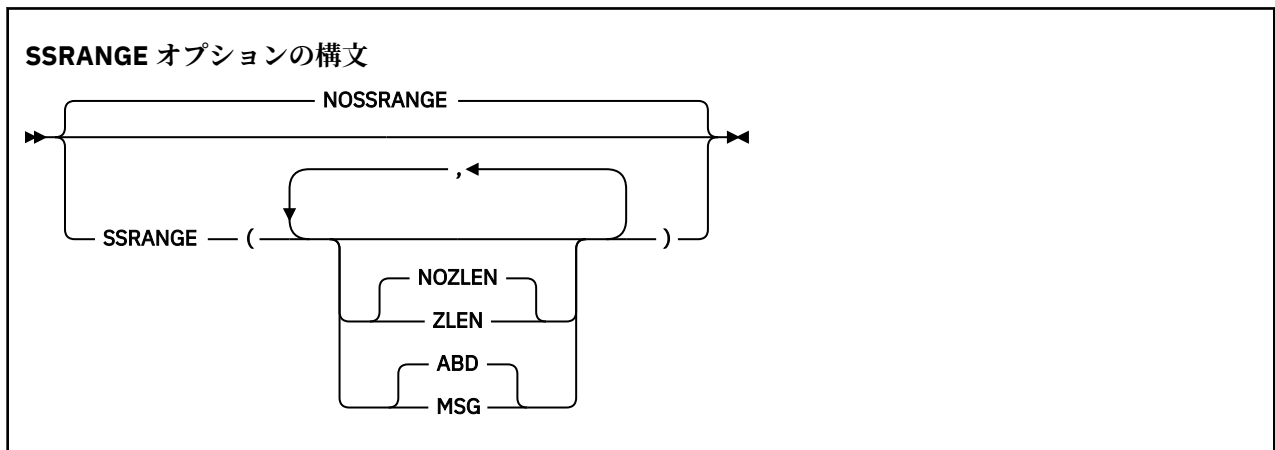
572 ページの『IMS サブオプションの分離』

関連参照

376 ページの『矛盾するコンパイラー・オプション』

SSRANGE

範囲外のストレージ参照を検査するコードを生成するには、SSRANGE を使用します。



デフォルトは、NOSSRANGE

サブオプションのデフォルトは SSRANGE のみが指定されている場合、NOZLEN, ABD です。

省略形は SSR | NOSSR

SSRANGE は、ALL 添え字を含む添え字、またはインデックスが、関連する表の領域外の領域を参照しようとするかどうかを検査するコードを生成します。各添え字または指標の妥当性は、個別に検査されることはありません。代わりに、テーブルの範囲外を参照しないように、有効アドレスが検査されます。

サブオプションなしで SSRANGE を指定すると、SSRANGE (NOZLEN, ABD) の指定として受け入れられます。

注: SSRANGE オプションが有効な場合、範囲検査はコンパイラーによって生成され、検査は常にランタイムに実行されます。ランタイム・オプション CHECK(OFF) を指定して、ランタイムのコンパイル済み範囲検査を無効にすることはできません。

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。

non-UTF-8 データ項目および関数値の参照変更式は、以下のことを確認するために検査されます。

- 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。
- 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区域を参照していない。
- 長さの値 (指定されている場合) が 1 以上である。

ZLEN および NOZLEN サブオプションは、コンパイラーが参照変更の長さを検査する方法を制御します。

- ZLEN が有効な場合、コンパイラーは、参照変更の長さがゼロ以上であることを確認するコードを生成します。ゼロ長の参照変更指定は、ランタイムに SSRANGE エラーを受け取りません。

- NOZLEN が有効な場合、コンパイラーは、参照変更の長さが 1 以上であることを確認するコードを生成します。長さゼロの参照変更指定は、ランタイムに SSRANGE エラーを受け取ります。これは、以前の COBOL バージョンで SSRANGE がどのように動作したかと互換性があります。

MSG および ABD サブオプションは、範囲検査が失敗した場合の COBOL プログラムのランタイム動作を制御します。

- MSG が有効であり、範囲検査が失敗した場合は、ランタイム警告メッセージが発行されます。また、影響を受けるステートメントは引き続き実行されます。プログラムは続行され、他の範囲外の条件を識別する可能性があります。
- ABD が有効であり、範囲検査が失敗した場合、最初の範囲外条件はランタイム・エラー・メッセージになり、プログラムは異常終了します。その次の潜在的な範囲外状態を見つけるには、最初の範囲外状態を修正し、プログラムを再コンパイルして再実行します。他のすべての潜在的な範囲外状態を特定するには、このプロセスを何度も繰り返さなければならない可能性があります。

無制限グループまたはその従属項目の場合、検査は参照変更式に対してのみ行われます。無制限グループに従属するテーブルに対する添字付きまたは索引付きの参照は、検査されません。

関連概念

118 ページの『[参照修飾子](#)』

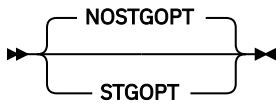
関連タスク

492 ページの『[有効範囲の検査](#)』

STGOPT

STGOPT オプションは、ストレージ最適化を制御します。

STGOPT オプションの構文



デフォルトは、NOSTGOPT

省略形は S0 | NOS0

STGOPT を指定した場合、コンパイラーは以下のデータ項目のいずれかまたはすべてを破棄し、これらの項目用にストレージを割り振りません。

- 未参照 LOCAL-STORAGE および WORKING-STORAGE のレベル 77 およびレベル 01 の基本データ項目
- レベル 01 グループ項目 (どの従属項目も参照されない場合)
- 未参照特殊レジスター

注: STGOPT オプションは、VOLATILE 節を持つデータ項目では無視されます。詳しくは、「*Enterprise COBOL for z/OS 言語解説書*」の『VOLATILE 節』を参照してください。

コンパイラーは、破棄されたデータ項目をその VALUE 節内の値に初期化するコードを生成しません。

さらに、STGOPT、データ項目 LOCAL-STORAGE SECTION パフォーマンスを最適化するためにメモリ内で並べ替えることができます。

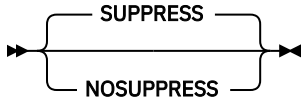
NOSTGOPT が有効になっている場合、すべてのデータ項目 (参照されないデータ項目を含む) のストレージはコンパイラーによって割り振られ、データ項目がパフォーマンス向上のために再配列されることはなく、VALUE 節で定義されたデータ項目はいずれも、たとえ参照されることがなくても必ず初期化されます。

また、RULES (UNREF | NOUNREFALL | NOUNREFSOURCE) 参照されていないデータ項目に対して警告メッセージを発行するかどうかを制御するオプション。詳しくは、[437 ページの『RULES』](#)を参照してください。

SUPPRESS

NOSUPPRESS オプションは、コピーブック情報をリストに表示できるように、プログラム内のすべての COPY ステートメントの SUPPRESS 句を無視する場合に使用します。コピーブック情報はデバッガーやツールなどで使用でき、その場合それらのソース・コードを変更する必要はありません。

SUPPRESS オプションの構文



デフォルトは、SUPPRESS

省略形は SUPP | NOSUPP

NOSUPPRESS

COPY ステートメントの SUPPRESS 句を無視します。

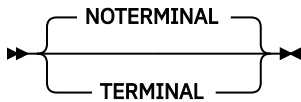
SUPPRESS

COPY ステートメントの SUPPRESS 句を有効にします。

TERMINAL

TERMINAL を使用すると、進行メッセージと 診断メッセージを SYSTERM DD 名に送ることができます。

TERMINAL オプションの構文



デフォルトは次のとおりです:NOTERMINAL

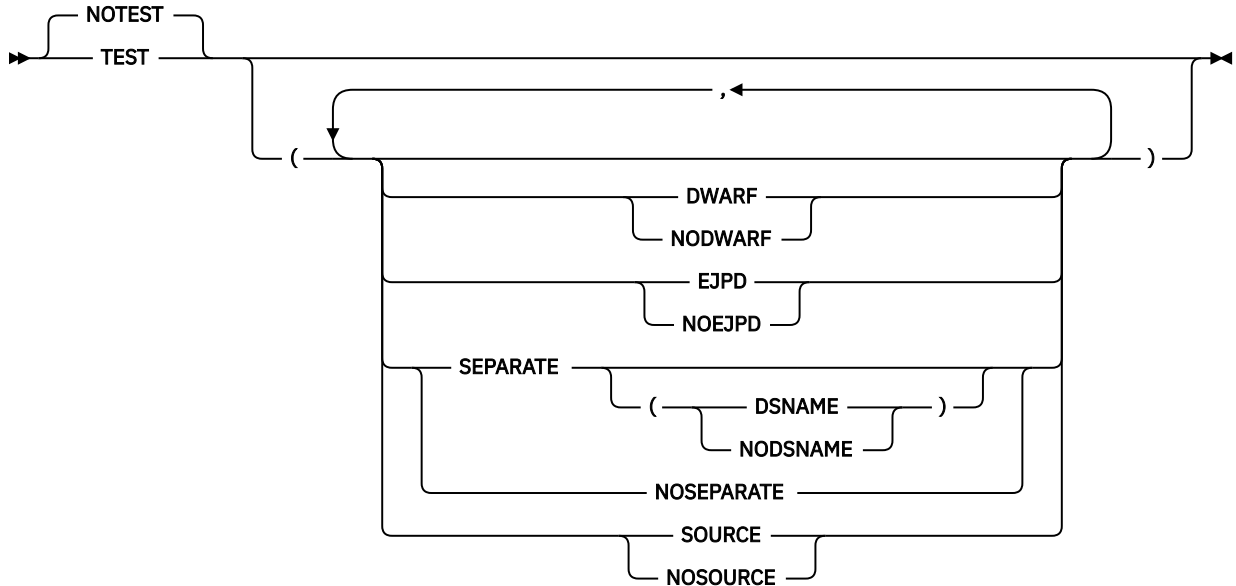
省略形は TERM | NOTERM

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 を使用して、IBM z/OS デバッガ や Fault Analyzer などの問題判別ツールによるデバッグを可能にするオブジェクト・コードを生成します。TEST を使用すると、言語環境によって生成される定様式ダンプにシンボリック変数を組み込むこともできます。

TEST オプションの構文



デフォルトは、NOTEST(NODWARF, NOSOURCE, NOSEPARATE)

サブオプションのデフォルトは以下のとおりです。

- NODWARF, NOSOURCE, NOSEPARATE (NOTEST がサブオプションなしで指定されている場合)
- NOEJPD, DWARF, SOURCE, NOSEPARATE (TEST がサブオプションなしで指定されている場合)

省略形: なし

サブオプションの省略形は以下のとおりです。

- NOSO | SO (SOURCE | NOSOURCE の場合)
- NOSEP | SEP (SEPARATE | NOSEPARATE の場合)

DWARF | NODWARF

TEST(DWARF)が有効な場合は、完全な DWARF 診断情報がオブジェクト・プログラムに組み込まれるか、SEPARATE サブオプションが有効な場合は別のデバッグ・ファイルに組み込まれます。このオプションは、CEEDUMP や IBM Fault Analyzer などのアプリケーション障害分析ツールで、最高のユーザビリティを実現します。

NOTEST(DWARF)が有効になっている場合、デバッグ情報は、TEST(DWARF)で使用可能な DWARF 情報のサブセットです。NOTEST(DWARF)が有効な場合に生成される DWARF 診断情報は、IBM z/OS デバッガでは使用できません。デバッガを使用する必要がなく、TEST オプションのパフォーマンスへの影響を回避する一方で、CEEDUMP や IBM Fault Analyzer などのアプリケーション障害分析ツールのユーザビリティを向上させたい場合は、NOTEST(DWARF)の使用を検討してください。

コンパイラによって生成されたデバッグ情報は、業界標準の DWARF 形式です。DWARF について詳しくは、「DWARF/ELF エクステンション ライブラリー・リファレンス」の『[About Common Debug Architecture](#)』を参照してください。

NODWARF が有効になっている場合、DWARF 診断情報はオブジェクト・プログラムに組み込まれず、別のデバッグ・ファイルに書き込まれません。

注:

- SOURCE および SEPARATE は、NODWARF では許可されません。
- TEST または NOTEST の DWARF サブオプションを指定する場合は、CODEPAGE オプションを、COBOL ソース・プログラムに使用される CCSID に設定する必要があります。DBCS リテラルまたは DBCS ユ

ユーザー定義語で日本語文字を使用するプログラムは、CODEPAGE オプションを日本語コード・ページ CCSID に設定してコンパイルする必要があります。詳しくは、[385 ページの『CODEPAGE』](#)を参照してください。

EJPD | NOEJPD

EJPD および NOEJPD は、実動デバッグ・セッションでの IBM z/OS デバッガコマンド JUMPTO および GOTO の使用可能化を制御します。EJPD および NOEJPD は、TEST オプションとゼロ以外の OPTIMIZE レベル (OPTIMIZE(1) または OPTIMIZE(2)) を指定した場合にのみ有効になります。

TEST(EJPD) およびゼロ以外の OPTIMIZE レベルを指定した場合は、以下のようになります。

- JUMPTO コマンドと GOTO コマンドが有効になります。
- プログラムの最適化の程度は低減されます。最適化は、ステートメント内で実行されますが、ほとんどの最適化はステートメント境界を超えません。

TEST(NOJPD) およびゼロ以外の OPTIMIZE レベルを指定した場合は、以下のようになります。

- JUMPTO および GOTO コマンドは有効になっていませんが、SET WARNING OFF IBM z/OS デバッガコマンドを使用する場合は、JUMPTO および GOTO を使用できます。このシナリオでは、JUMPTO と GOTO の結果は予測できません。
- 通常程度のプログラムの最適化が実行されます。

注：EJPD は NOTEST では許可されません。

SOURCE | NOSOURCE

SOURCE を指定すると、コンパイラーによって生成される DWARF デバッグ情報に拡張ソース・コードが組み込まれます。

注：NODWARF が指定されている場合、SOURCE は許可されません。

NOSOURCE を指定した場合、生成される DWARF デバッグ情報には拡張ソース・コードは含まれません。TEST(NOSOURCE) で IBM z/OS デバッガを使用してデバッグすることはできません。

SEPARATE[(DSNAME | NODSNAME)] | NOSEPARATE

SEPARATE がサブオプションなしで指定されている場合、デフォルトは SEPARATE(NODSNAME) です。

デバッグ機能を保持しながら、ディスク上のプログラム・オブジェクト・サイズを制御するには、SEPARATE または SEPARATE(NODSNAME) を指定します (NOSEPARATE はロードされたプログラム・オブジェクトのサイズには影響しません)。生成された DWARF デバッグ情報は、オブジェクト・プログラムではなく SYSDEBUG データ・セットに書き込まれます。デバッグ能力を維持しながらモジュール・サイズを制御する方法については、後述の[セクション](#)を参照してください。

SEPARATE(DSNAME) が有効になっている場合、コンパイル時に使用された SYSDEBUGdataset 名がオブジェクト・プログラムに保管されます。この名前は、DWARF 情報が必要なランタイムにデフォルトとして使用されます。dataset 名は、SYSDEBUGCOBOL デバッグ・ファイル・ユーザー出口 IGZIUXB を使用してオーバーライドできます。NODSNAME が有効になっている場合、IGZIUXB ユーザー出口は、プログラムの DWARF デバッグ情報を見つけるための唯一のメカニズムを提供することに注意してください。

注：

- NODWARF が指定されている場合、SEPARATE は許可されません。
- SYSDEBUG データ・セット内の DWARF デバッグ情報のサポートには、IBM z/OS Debugger 14.1 以降が必要です。
 - IBM Debug for z Systems 14.1 (5655-Q50) (旧 IBM Debug Tool for z/OS) 以降
 - IBM Developer for z Systems 14.1 (5724-T07) 以降
 - IBM Application Delivery Foundation for z Systems 3.1 (5655-AC6) 以降

生成された DWARF デバッグ情報をオブジェクト・プログラムに組み込むには、NOSEPARATE を指定します。

デバッグ能力を維持したモジュール・サイズの制御:

TEST の DWARF サブオプションにより、コンパイラーは、IBM z/OS デバッガがデータ名、段落名などを解決するために使用するデバッグ情報テーブルを生成します。この情報は、ストレージを大量に使用する場合があります。この情報をオブジェクト・プログラムにコンパイルするか、別個の SYSDEBUG データ・セットに書き込むかを選択できます。

- 実行可能ファイルを小さくするには、SEPARATE サブオプションを使用し、IBM z/OS デバッガセッション中に使用するために別個のデバッグ・ファイルを保持します。
- 個別のデバッグ・ファイルを管理する必要がないようにするには、NOSEPARATE サブオプションを指定してコンパイルします。このサブオプションを指定すると、DASD 上のオブジェクト・プログラムが大きくなることに注意してください。仮想記憶域にロードされるときサイズは、NOSEPARATE オプションを使用しても増やされません。

JCL または TSO から COBOL コンパイラーを呼び出し、NOTEST|TEST(...,SEPARATE,...)を指定すると、SYSDEBUG DD ステートメントで指定したデータ・セットに DWARF デバッグ情報が書き込まれます。このステートメントのコーディング方法および SYSDEBUG データ・セットの詳細については、デバッグ・データ・セットの定義方法および論理レコード長とブロック・サイズについての下記の関連情報を参照してください。

z/OS ユニックスシェルから COBOL コンパイラーを起動し、NOTEST|TEST(...,SEPARATE,...)を指定すると、DWARF デバッグ情報が現行ディレクトリーのファイル.dbg に書き込まれます。ここで、ファイルは COBOL ソース・ファイルの名前です。

パフォーマンスとデバッグ能力:

以下のように、得られるデバッグ能力の程度、およびプログラムのパフォーマンスを制御できます。

- 最良のパフォーマンスを得るためには、デバッグにいくつかの制約がありますが、ゼロ以外の OPTIMIZE レベル、STGOPT、および TEST(NO EJPJ)を使用してコンパイルしてください。
 - IBM z/OS デバッガコマンド JUMPTO および GOTO は、はサポートされていません。ただし、SET WARNING OFFIBM z/OS デバッガコマンドを使用する場合は、引き続き JUMPTO および GOTO を使用できます。このシナリオでは、JUMPTO と GOTO の結果は予測できません。
 - IBM z/OS デバッガコマンドは、DESCRIBE ATTRIBUTES コマンドを除き、STGOPT オプションによってプログラムから廃棄されたデータ項目を参照することはできません。
 - IBM z/OS デバッガコマンド AT CALL エントリー名はサポートされていません。
- 上記の実動デバッグ・シナリオからプログラム・パフォーマンスをいくらか低下させますが、IBM z/OS デバッガコマンド JUMPTO および GOTO の動作を予測可能にするには、ゼロ以外の OPTIMIZE レベルおよび TEST(EJPJ)を指定します。

STGOPT オプションによって破棄された項目の参照に関する制約事項、および AT CALL コマンドに関する制約事項は、ゼロ以外の OPTIMIZE レベルおよび TEST(EJPJ)を使用する場合にも適用されます。

- パフォーマンスが最も遅く、デバッグ機能が最大になるようにするには、OPTIMIZE(0)、NOSTGOPT、および TEST を指定します。

OPTIMIZE(0)オプションを指定すると、コンパイラーは低速コードを生成しますが、すべての IBM z/OS デバッガコマンドがサポートされます。

言語環境:

TEST オプションいずれかのサブオプションで指定された、および DWARF を使用した NOTEST は、以下の 2 つの機能をダンプに追加することにより、言語環境からの定様式ダンプを向上させることができます。

- 単なるオフセットではなく、失敗したステートメントを示す行番号
- プログラム変数 (DWARF が有効な場合) の値

DWARF を指定すると、ダンプにはプログラム変数と、失敗したステートメントの行番号が入ります。NODWARF を指定すると、ダンプにはプログラム変数も、失敗したステートメントの行番号も含まれません。

Enterprise COBOL は、言語環境 が提供するダンプ・サービスを使用して、言語環境 準拠のその他のメンバー言語によって生成されるダンプの内容および形式と一致するダンプを作成します。

言語環境が未処理の条件のダンプを生成するかどうかは、ランタイム・オプション TERMTHDACT の設定によって決まります。TERMTHDACT(DUMP)を指定すると、重大度 2 以上の条件が処理されない場合にダンプが生成されます。

注：IBM z/OS デバッガ は以下の製品のコンポーネントです。

- IBM Developer for z/OS Enterprise Edition (IBM Application Delivery Foundation for z/OS に組み込まれている)
- IBM z/OS デバッガ (旧称: IBM Debug for z Systems および デバッグ・ツール)
- IBM 開発者 z/OS

お客様のニーズに最も適している IBM デバッグ製品を見つけるには、<https://www.ibm.com/docs/developer-for-zos/latest?topic=debugger-overview-zos> を参照してください。

DWARF 診断資料が含まれる LLA/VLF 管理対象プログラム:

TEST(NOSEPARATE) またはその他の同等のオプションのために DWARF 診断情報がオブジェクト・プログラムに含まれており、プログラムが VLF にステージングされている場合、その情報をプログラムから抽出することはできません。実行 JCL では、IGZLOAD DD または STEPLIB DD、あるいはその両方を使用して、VLF にステージングされたプログラムをアプリケーションがロードする LLA/VLF 管理対象ライブラリーの場所を指定する必要があります。そうしないと、COBOL 可変情報およびトレースバック内のステートメント番号が CEEDUMP で欠落し、診断メッセージが CEEDUMP に表示されます。詳しくは、<https://www.ibm.com/support/pages/diagnostic-messages-ceedumps-ibm-enterprise-cobol-zos> で *CEEDUMPs for IBM Enterprise COBOL for z/OS* の診断メッセージを参照してください。

VLF にステージングされたプログラムごとに、DWARF 診断情報を抽出するために以下のステップが実行されます。

1. COBOL ランタイムは、IGZLOAD DD の割り振りを検査します。次に、ライブラリーが IGZLOAD DD に関連付けられているかどうかを検査します。例えば、名前が VLF にステージングされたプログラムと一致するメンバー (COBOL プログラムなど) が含まれているかどうかを検査します。
2. IGZLOAD DD が割り振られなかったか、メンバーが見つからなかったために前のステップが失敗した場合は、STEPLIB DD を使用して同じプロセスが試行されます。
3. メンバーが検出され、そのメンバーから取得した DWARF 診断情報とともに保管されているチェックサムが、実行中のプログラムのコンパイル単位のチェックサムと一致する場合、その情報はメンバーから抽出されます。

重複するメンバーがそれぞれ IGZLOAD DD 連結または STEPLIB DD 連結に存在する場合は、DD 連結内の最初のおカレンスを使用してチェックサムが取得されることに注意してください。

プログラムの改訂されたコピーが LLA/VLF 管理対象ライブラリーに配置されているものの、ステージングされた LLA/VLF プログラムの環境に適したリフレッシュが完了していない場合には、不一致が発生する可能性があります。

例えば、以下のすべての条件が満たされている場合は、DSN=USER.COBOL.LOADLIB を指定して IGZLOAD DD を割り振ることができます。

- USER.COBOL.LOADLIB は LNKLIST にあります。
- USER.COBOL.LOADLIB(PGM1)は VLF にステージングされます。
- PGM1 は LNKLIST の USER.COBOL.LOADLIB から実行されます。

```
//G01      EXEC  PGM=PGM1,                * CALLED FROM LNKLIST
//STEPLIB   DD    DSN=CEE1.SCEERUN,DISP=SHR
//IGZLOAD   DD    DSN=USER.COBOL.LOADLIB,DISP=SHR  * CONTAINS PGM1
```

プログラムが LLA/VLF によって管理され、VLF にステージングされ、LNKLIST 内のデータ・セットの 1 つから実行される場合は、IGZLOAD DD を割り振る必要があります。ただし、プログラムが STEPLIB DD 内のデータ・セットの 1 つにある場合は、IGZLOAD DD を割り振る必要はありません。

関連概念

DWARF/ELF エクステンション ライブラリー・リファレンス (Common Debug Architecture について)

関連タスク

[298 ページの『デバッグ・データ・セットの定義 \(SYSDEBUG\)』](#)

[499 ページの『デバッガーの使用』](#)

[538 ページの『CEEDUMP 処理での情報の抑止 \(IGZ1OPT\)』](#)

TERMTHDACT による 言語環境 ダンプの生成

(言語環境デバッグガイド)

TEST ランタイム・オプションの使用時における特別の考慮事項

(IBM z/OS デバッガ・ユーザー・ガイド)

関連参照

[294 ページの『論理レコード長とブロック・サイズ』](#)

[318 ページの『cob2 入出力ファイル』](#)

[376 ページの『矛盾するコンパイラ・オプション』](#)

[429 ページの『OPTIMIZE』](#)

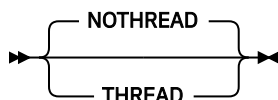
TEST | NOTEST

(言語環境プログラミング・リファレンス)

THREAD

THREAD は、COBOL プログラムが複数の スレッドまたは タスクを持つ エンクレーブで POSIX PL/I 言語環境 実行できるようにすることを示す

THREAD オプションの構文



デフォルトは、NOTHREAD

省略形: なし

THREAD オプションを使用してコンパイルされたプログラムは、スレッド化されていないアプリケーションでも使用することができます。ただし、スレッド化されたアプリケーションで COBOL プログラムを実行する場合は、言語環境のエンクレーブ内のすべての COBOL プログラムが THREAD オプションを使用してコンパイルされている必要があります。

NOTHREAD を指定した場合、その COBOL プログラムは、複数の POSIX スレッド または PL/I タスクを含むエンクレーブで実行することができません。

以前のコンパイラを使用してコンパイルされたプログラムは、Enterprise COBOL は、次のようにコンパイルされたものとして扱われます。NOTHREAD。

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
- プロシージャー名が指定されていない GO TO ステートメント
- INITIAL フレーズ PROGRAM-ID 句または INITIAL コンパイラオプション
- ネストされたプログラム
- RERUN
- 分割モジュール

- SORT または MERGE ステートメント
- STOP リテラル・ステートメント
- USE FOR DEBUGGING ステートメント

また、スレッド化されている場合とスレッド化されていない場合では、一部の言語構成要素のセマンティクスが異なります。

スレッド化されたアプリケーションの場合、プログラミングおよび環境に関するさまざまな制約が適用されますが、スレッド化されていないアプリケーションでのプログラムの使用についてはあまり制約がありません。例えば、THREAD オプションを使用してコンパイルされたプログラムは、実行時に複数の POSIX スレッドまたは PL/I タスクがアプリケーションに含まれていない場合は、CICS 環境および IMS 環境で実行することができ、AMODE 24 で実行することも、マルチスレッド化をサポートしない他のプログラムとの間で呼び出しを行うこともできます。

THREAD オプションを使用してコンパイルされたプログラムは、言語環境 事前初期設定ルーチン CEEPIPI を呼び出すことにより作成された再使用可能環境でサポートされます。しかし、再利用可能な環境は使用して RTEREUS ランタイムオプションは、THREAD オプション。

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。

AMODE 64 の考慮事項: の THREADLP(64)が有効な場合、オプションは無視されます。ユーザーが THREAD オプションを明示的に指定すると、通知メッセージが発行されます。

関連タスク

651 ページの『第 32 章 マルチスレッド化のための COBOL プログラムの準備』

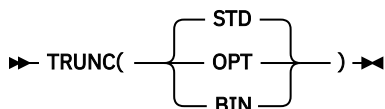
関連参照

376 ページの『矛盾するコンパイラー・オプション』

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。

TRUNC オプションの構文



デフォルトは、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。 いつ TRUNC(BIN) が有効になっている場合、すべてのバイナリ項目 (USAGE COMP、COMP-4、または BINARY) はネイティブハードウェアバイナリアイテムとして扱われ、それぞれが個別に宣言されているかのように扱われます。 USAGE COMP-5:

- 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 (IMS、Db2、C/C++、FORTRAN、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 適用する効果がある TRUNC(BIN) 個々のデータ項目に対する動作。 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・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 節に指定された数値リテラルは、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。

注: TRUNC(BIN) と NUMCHECK(BIN) が両方とも有効で、エラー・メッセージが異常終了が生成される際に、パフォーマンスを改善するために後で TRUNC(STD|OPT) に切り替えようとしている場合は、データを訂正する必要があります。 切り替えない場合は、NUMCHECK(BIN) をオフにして、アプリケーションの実行時間を減らし、エラー・メッセージや異常終了が生成されないようにすることができます。

TRUNC の例 1

```
01  BIN-VAR      PIC S99 USAGE BINARY.
...
MOVE 123451 to BIN-VAR
```

次の表に、MOVE ステートメント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	00 01 E2 3B	123451
受信側 TRUNC(STD)	51	00 33	51
受信側 TRUNC(OPT)	-7621	E2 3B	2J
受信側 TRUNC(BIN)	-7621	E2 3B	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ためです。

TRUNC の例 2

```
01  BIN-VAR      PIC 9(6)  USAGE BINARY
    . . .
    MOVE 1234567891 to BIN-VAR
```

次の表に、MOVE ステートメント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	49 96 02 D3	1234567891
受信側 TRUNC(STD)	567891	00 08 AA 53	567891
受信側 TRUNC(OPT)	567891	53 AA 08 00	567891
受信側 TRUNC(BIN)	1234567891	49 96 02 D3	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

[53 ページの『数値データの形式』](#)

関連タスク

[549 ページの『CICS オプションを使用したコンパイル』](#)

関連参照

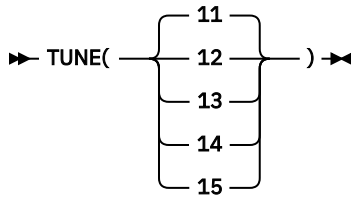
[422 ページの『NUMCHECK』](#)

VALUE 節 (*Enterprise COBOL for z/OS 言語解説書*)

TUNE

TUNE オプションは、実行可能プログラムが最適化されるアーキテクチャーを指定します。

TUNE オプションの構文



デフォルトは、ARCH が指定されている場合、デフォルトの TUNE レベルは ARCH レベルと一致します。ARCH が指定されない場合、ARCH と TUNE の両方がデフォルトで 11 になる。

省略形: なし

現在サポートされているアーキテクチャー・レベルおよびモデルのグループは以下のとおりです。

11

IBM z/Architecture モードの 2964-xxx (IBM z13) モデルおよび 2965-xxx (IBM z13s) モデル用に最適化されたコードを生成します。

12

IBM z/Architecture モードの 3906-xxx (IBM z14) モデルおよび 3907-xxx (IBM z14 ZR1) モデル用に最適化されたコードを生成します。

※13

IBM z/Architecture モードの 8561-xxx (IBM z15) モデルおよび 8562-xxx (IBM z15 T02) モデル用に最適化されたコードを生成します。

14

に最適化されたコードを生成します。IBMz16 および 3932-xxx (IBMz16 A02) モデルに最適化されたコードを IBM z/Architecture モード。

15

IBM z/Architecture モードで 9175-xxxx (IBM z17) モデルに最適化されたコードを生成します。

TUNE オプションは、実行可能プログラムが最適化されるアーキテクチャーを指定します。TUNE レベルは、有効な ARCH レベルの制限内で、コンパイラーが使用可能なマシン・インストラクションを選択する方法を制御します。TUNE オプションは、指定された TUNE アーキテクチャーで可能な限り最高のパフォーマンスを提供するために、指定された ARCH レベルで許可される命令から選択します。

注: TUNE はパフォーマンスにのみ影響します。アプリケーションを実行できるプロセッサ・モデルには影響しません。

アプリケーションが最も頻繁に実行されるマシンのアーキテクチャーに合わせるには、TUNE を選択します。TUNE レベルは、常に ARCH レベル以上でなければなりません。これは、アプリケーションを実行できるマシン用にアプリケーションを調整する必要があるためです。コンパイラーは、ARCH を下にするのではなく TUNE を上に調整することによって、これを強制します。TUNE は、アプリケーションを実行できる場所を指定せず、最適化にのみ影響します。

例えば、z15 の実動マシンと、z14 の災害復旧 (DR) マシンがある場合は、ARCH (12) および TUNE (13) を指定してコンパイルする必要があります。この方法で、プログラムは DR マシン上では適切に実行され、実動環境では可能な限り速く実行されます。

注: 指定した TUNE レベルが、指定した ARCH レベルより低い場合は、コンパイラーによって、ARCH レベルと一致するように TUNE レベルが調整されます。

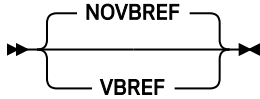
関連参照

379 ページの『ARCH』

VBREF

VBREF は、ソース・プログラムの中で使用されるすべてのステートメントと、これらのステートメントが使用されている行番号の間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ステートメントが使用された回数の合計も出します。

VBREF オプションの構文



デフォルトは、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。

VLR

VLR オプションは、可変長レコードの READ ステートメントから返されたレコード長がレコード記述と矛盾する場合に、返されるファイル状況に作用します。VLR(COMPAT)は、READ ステートメントがレコード長の競合を引き起こすプログラムがある場合、以前のバージョンから Enterprise COBOL 6 への移行を容易にする。

VLR オプションの構文



デフォルトは、VLR(STANDARD)

省略形: VLR(C | S)

READ ステートメントの実行後:

- 読み取られたレコード内の文字位置数がファイルのレコード記述項目で指定された最小サイズより小さい場合、レコード域内の読み取られた最後の有効文字の右側の部分は未定義になります。
- 読み取られたレコード内の文字位置数がファイルのレコード記述項目で指定された最大サイズより大きい場合、最大サイズの右側にあるレコード部分は切り捨てられます。

いずれの場合でも、READ ステートメントは正常に実行され、ファイル状況は VLR コンパイラ・オプション設定に応じて、00 (レコード長の矛盾状態を隠します) または 04 (レコード長の矛盾が発生したことを示します) に設定されます。

VLR(COMPAT)

VLR(COMPAT) は、読み取りレコードのサイズを FD 節の「VERYING IN SIZE FROM min TO max」宣言に照らして検査します。VLR(COMPAT) を指定した場合、READ ステートメントでレコード長の矛盾が検出されると、状況値 00 を受け取ります。

注: 以前のコンパイラとの互換性のために VLR(COMPAT)。

VLR(STANDARD)

重要: ファイル・ステータスのテストに VLR(STANDARD) を使用するには、各 READ 文にコードを追加する必要があります。

VLR(STANDARD) は、読み取りレコードのサイズを FD レベル 01 節の宣言に照らして検査します。VLR(STANDARD) を指定した場合、READ ステートメントでレコード長の矛盾が検出されると、状況値 04 を受け取ります。

VLR(STANDARD) を使えば、レコード内の未定義のデータにアクセスすることを避けられるし、切り捨てられたレコードの一部を参照しようとして保護例外が発生することも避けられる。

ヒント: あるいは、コンパイル時の情報メッセージ IGYPG3178。これは、最小と最大のレコードのサイズが RECORD IS VARYING 句の範囲と一致しないことを示す。このメッセージが表示されない場合、04 ではなく 00 のステータス値が VLR(STANDARD) と共に表示される。このメッセージが表示された場合、RECORD IS VARYING 句の範囲外の長さのレコード記述が少なくとも 1 つあることになる。コンパイラは、VLR の設定に関係なく、IGYPG3178 を発行し、このメッセージによってコンパイルが中断されることはない。IGYPG3178 については、[インフォメーション・メッセージ IGYPG3178](#) を参照のこと。

以下の例では、VLR オプションが、FD 句の宣言と FD レベル 01 句に照らし合わせて、読み込まれたレコードのサイズをチェックする方法を示している。459 ページの表 54 以下では、読み込まれたレコードの長さが異なる場合の結果のファイル・ステータス (FS) を示している。この場合、読み込まれた記録は一連の文字列である。

```
* The file contains two record definitions:
* REC-20: up to 20 characters
* REC-50: up to 50 characters
* The file must contain between 10 to 80 characters in total.
FD MYDD
  block contains 0 records
  record varying in size from 10 to 80
  recording mode V.
01 REC-20
  02 PIC X(20).
01 REC-50.
  02 PIC X(50).
```

表 54. 読み取られるレコードの長さとファイル状況					
読み取られるレコードの長さ	5	15	40	70	85
>= 最小変動 (10)	X	✓	✓	✓	✓
<= 最大変化率 (80)	✓	✓	✓	✓	X
>= 最低レベル 01 (20)	X	X	✓	✓	✓
<= 最大レベル 01 (50)	✓	✓	✓	X	X
有効なデータ?	X	X	✓	X	X
COBOL 4	FS=04 X	FS=00 ✓	FS=00 ✓	FS=00 ✓	FS=00 ✓
コボル 6VLR (COMPAT)	FS=04 X	FS=00 ✓	FS=00 ✓	FS=00 ✓	FS=00 ✓
コボル 6VLR (STANDARD)	FS=04 X	FS=04 X	FS=00 ✓	FS=04 X	FS=04 X

表 54. 読み取られるレコードの長さでファイル状況 (続き)

読み取られるレコードの長さ	5	15	40	70	85
>= 最小変動 (10)	✗	✓	✓	✓	✓
<= 最大変化率 (80)	✓	✓	✓	✓	✗
>= 最低レベル 01 (20)	✗	✗	✓	✓	✓
<=最大レベル 01 (50)	✓	✓	✓	✗	✗
有効なデータ？	✗	✗	✓	✗	✗

注記:

- ・ "FS=00" は、レコードが受理された (✓) ことを意味する。"FS=04" は、レコードが無効 (✗) であることを意味する。
- ・ 「Min varying」と"max varying"は、RECORD VARYING 句で指定された最小サイズと最大サイズを指し、この場合は 10 と 80 である。「最小レベル 01」と「最大レベル 01」は、最小と最大のレベル 01 レコード記述のサイズを指し、この場合、REC-20 の場合は 20、REC-50 の場合は 50 である。

ここで、[?]は未定義のスペース、[X]は切り捨てられたデータ、[A]のようなボックス内の他の文字は塗りつぶされたデータを表す。

- ・ レコードの長さが 5 の場合、次のように表現される：

```
[A][B][C][D][E][?][?][?][?][?][?][?][?][?][?][?][?][?][?]
|<----- 5 ----->|<----- Undefined (15) ----->|
```

レコード長 5 は、最小変化量 10 未満であり、最小レベル 01 の 20 未満であるため、どの COBOL コンパイラ・バージョンでもレコードは無効 (✗) である。

- ・ レコード長 15 は以下のように表される：

```
[A][B][C][D][E][F][G][H][I][J][K][L][M][N][O][?][?][?][?][?]
|<----- 15 ----->|<Undefined(5)>|
```

レコード長 15 は、最小変化量 10 より大きく、最小レベル 01 の 20 より小さいので、COBOL 6 VLR(STANDARD) の下ではレコードは無効(✗)である。

- ・ レコードの長さが 40 の場合、次のように表現される：

```
[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][1][2][3][4][5][6][7][8][9][0][1][2][3][4]
|<----- 40 ----->|
```

レコードの長さが 40 であることは、10～80 の変動範囲内であり、レベル 01 の要件を満たしているため、レコードは有効である (✓)。

- ・ レコード長 70 は次のように表される：

```
[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][1][2][3][4][5][6][7][8][9][0][1][2][3][4]
[5][6][7][8][9][0][1][2][3][4][5][6][7][8][9][0][1][2][3][4]
[5][6][7][8][9][0]
|<----- 70 ----->|
```

レコード長 70 は、変化する範囲 10～80 内にあるが、最大レベル 01 の 50 より大きいので、COBOL 6 VLR(STANDARD) ではレコードは無効である。

- ・ レコード長 85 は以下のように表される：

```

[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][1][2][3][4][5][6][7][8][9][0][1][2][3][4]
[5][6][7][8][9][0][1][2][3][4][5][6][7][8][9][0][1][2][3][4]
[5][6][7][8][9][0][1][2][3][4][5][6][7][8][9][0][1][2][3][4]
[5][6][7][8][9][0][1][2][3][4][5][6][7][8][9][0]
|<----- 80 ----->| [X][X][X][X][X]
|<--Truncated-->|

```

レコード長 85 は、最大変化量 80 および最大レベル 01 の 50 よりも大きいため、レコードは切り捨てられる(**X**)。

インフォメーションメッセージ IGYPG3178

情報メッセージ IGYPG3178、プログラムが"間違った長さの READ を"する可能性があるかどうかを教えてください。I/O の問題を避けるのにも役立つ。このメッセージは、ファイル定義 (FD) を修正することで解決できる"誤った長さの READ の"可能性を示すことで、VLR (COMPAT) から VLR (STANDARD) への移行を支援するために使用できます。また、実行のためにはプログラムの修正を必須とするように、メッセージの重大度を上げることができます。これを行うには、EXIT コンパイラ・オプションの MSGEXIT サブオプションを使って、メッセージ IGYPG3178 の重大度を I (RC=0) から S (RC=12), E (RC=8), または W (RC=4) に変更する。このメッセージの表示が不要な場合は、メッセージを完全に抑止することができます。

関連参照

399 ページの『出口』

可変長レコード - 不正な長さの READ
(Enterprise COBOL for z/OS 移行ガイド)

VSAMOPENFS

VSAMOPENFS オプションは、検査済みファイル保全性検査を必要とする正常な VSAM OPEN ステートメントから報告されるユーザー・ファイル状況に影響します。

VSAMOPENFS オプションの構文

```

▶▶ VSAMOPENFS( COMPAT
                SUCC )▶▶

```

デフォルトは、VSAMOPENFS (COMPAT)

省略形: VS(C | S)

COMPAT

VSAMOPENFS (COMPAT) を指定すると、VSAM OPEN ステートメントが正常に検査されたときに、ステートメントはファイル状況 97 を返します。これは、COBOL 6 より前のランタイム動作と互換性があります。

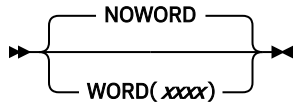
SUCC

VSAMOPENFS (SUCC) を指定すると、VSAM OPEN ステートメントが正常に検査されたときに、ステートメントはファイル状況 00 を返します。これにより、ユーザーは、通常他の正常な操作の場合と同様に、返されたファイル状況の最初の桁に 0 があるかどうかを検査するだけで済みます。

WORD

WORD(xxxx) は、コンパイル時に代替予約語テーブルを使用することを指定するために使用します。

WORD オプションの構文



デフォルトは、NOWORD

省略形は WD | NOWD

xxxx には、コンパイルで使用する予約語テーブル (IGYCxxxx) の名前の 後半の文字を指定します。IGYC は名前の最初の部分の標準 4 文字で、xxxx は 1 から 4 文字の長さにすることができます。

IBM 提供のデフォルト予約語テーブルを変更するには、代替予約語テーブルを使用します。システム・プログラマーが、設置場所に対して 1 つ以上の代替予約語テーブルを作成している可能性があります。代替予約語テーブルの名前については、システム・プログラマーにお問い合わせください。

Enterprise COBOL には、CICS アプリケーション専用の代替予約語テーブル (IGYCCICS) が用意されています。エラー・メッセージを使用して、CICS でサポートされない COBOL のワードにフラグを付けるようにセットアップされています。コンパイル時にこの CICS 予約語テーブルを使用したい場合は、コンパイラー・オプション WORD(CICS) を指定してください。

関連タスク

[549 ページの『CICS オプションを使用したコンパイル』](#)

関連参照

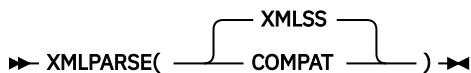
[376 ページの『矛盾するコンパイラー・オプション』](#)

[553 ページの『CICS 予約語テーブル』](#)

XMLPARSE

XMLPARSE は、XML 入力の処理に使用されるパーサーと、これによりプログラムで使用可能になる XML 処理機能を選択するために使用します。

XMLPARSE オプションの構文



デフォルトは、XMLSS

省略形: XP(X | C)

指定する場合は XMLPARSE(XMLSS) オプション、XML PARSE ステートメントは、z/OS XML システム サービスパーサー。次の XML 構文解析機能は、XMLPARSE(XMLSS) を指定した場合にのみ使用可能になります。

- XML 入力文書を XML スキーマに対して検証する (VALIDATING のフレーズ XML PARSE 声明)
- 拡張名前空間処理 (特殊レジスター XML-NAMESPACE、XML-NNAMESPACE、XML-NAMESPACE-PREFIX、および XML-NNAMESPACE-PREFIX)
- ドキュメントフラグメントを Unicode に自動変換 UTF-16 (RETURNING NATIONAL のフレーズ XML PARSE 声明)
- 入力文書のエンコードの指定 (XML PARSE ステートメントの ENCODING 句を使用)
- UTF-8 でエンコードされた XML 文書の直接構文解析
- XML のバッファーごとの XML 文書の構文解析
- XML 構文解析の System z Application Assist Processors (zAAPs) へのオフロード

指定する場合は XMLPARSE (COMPAT) オプション、XML PARSE ステートメントは、COBOL ライブラリの組み込みコンポーネントである XML パーサーを使用して処理されます。XML PARSE ステートメントの結果および操作上の動作は、COBOL 3 で得られるものと互換性があります。また、XMLPARSE (COMPAT) が使用された場合は COBOL 4 と互換性があります。上述の XMLPARSE (XMLSS) の拡張機能は使用できません。

関連タスク

687 ページの『第 38 章 XML 入力の処理』

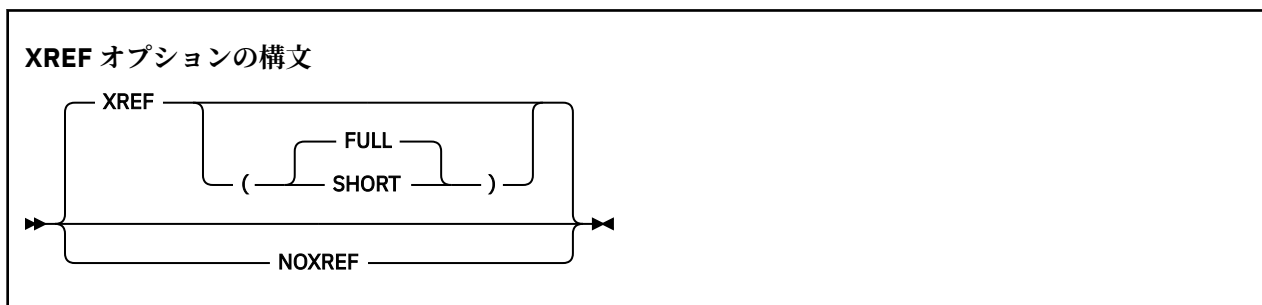
関連参照

XML PARSE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

z/OS XML System Services User's Guide and Reference

外部参照

XREF は、ソート済みの相互参照リストを作成するために使用します。



デフォルトは、XREF (FULL)

省略形は X | NOX

XREF、XREF (FULL)、または XREF (SHORT) を選択できます。サブオプションを付けずに XREF を指定すると、XREF (FULL) が有効になります。

リストのセクションには、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、およびそれらの名前が定義されている行番号が表示されます。外部プログラム名が識別されます。

531 ページの『例: XREF 出力: データ名相互参照』

533 ページの『例: XREF 出力: プログラム名相互参照』

また、関連コピーブックを取得したデータ・セットまたは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。

533 ページの『例: XREF 出力: COPY/BASIS 相互参照』

EBCDIC データ名とプロシージャ名は英数字順にリストされます。DBCS データ名とプロシージャ名は、プログラムでの物理的順序に基づいてリストされ、DBCSXREF インストール・オプションが DBCS 配列プログラムで選択された場合を除いて、EBCDIC データ名とプロシージャ名の前に表示されます。その場合は、DBCS データ名とプロシージャ名は、DBCS 配列プログラムで指定されたとおりに順序付けられます。

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、DBCS データ名とプロシージャ名、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。
- SYSLIB 連結に複数のデータ・セットが存在する場合、COPY/BASIS 相互参照が不完全であるか、欠落している可能性があります。この欠損が発生する可能性があるのは、XREF が CBL または PROCESS ステートメントでのみ設定されている場合、および XREFOPT=NO がインストール・デフォルトとして設定されているか、JCL PARM パラメーターで NOXREF がコーディングされている場合です。

COPY/BASIS 相互参照が確実に完全になるようにするため、システム・プログラマーの協力を得て、XREFOPT=FULL または XREFOPT=SHORT がインストール・デフォルトとして設定されていることを検証するか、あるいは JCL PARM パラメーターで XREF オプションをコーディングしてください。

関連概念

[485 ページの『第 22 章 デバッグ』](#)

関連タスク

[500 ページの『リストの入手』](#)

関連参照

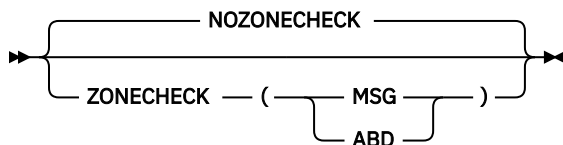
言語環境デバッグガイド (COBOL コンパイラー・オプション)

ZONECHECK

ZONECHECK オプションを使用して、送信データ項目として使用されるゾーン 10 進数データ項目の IF NUMERIC クラス・テストをコンパイラーに生成させます。

注: ZONECHECK は非推奨ですが、互換性のために許容されます。これは NUMCHECK (ZON (ALPHNUM)) に置き換えられます。

ZONECHECK オプションの構文



デフォルトは、NOZONECHECK

省略形は NOZC | ZC (MSG) | ZC (ABD)

MSG

MSG サブオプションは、送信側としてゾーン 10 進数データ項目を使用するたびに IF NUMERIC テストを実行することを要求します。データが無効な場合 (例えば、NOT NUMERIC)、ランタイム警告メッセージに行番号、データ項目名、データ項目の内容、およびプログラム名が発行されます。また、影響を受けるステートメントは引き続き実行されます。ZONECHECK (MSG) は、NUMCHECK (ZON, MSG) が有効であるかのように扱われます。

ABD

ABD サブオプションは、ゾーン 10 進数データ項目を送信側として使用するたびに IF NUMERIC テストを行うことを要求します。データが無効な場合 (例えば、NOT NUMERIC)、異常終了の原因となる終了メッセージが発行されます。ZONECHECK(ABD)は、NUMCHECK(ZON,ABD)が有効であるかのように扱われます。

ZONECHECK(MSG)と ZONECHECK(ABD)の両方とも、COBOL ステートメントでセNDERとして参照されるゾーン 10 進数データ項目ごとに、コンパイラーが暗黙的な数値クラス・テストを生成します。以下のサンプル・ステートメント内のデータ項目 B のように、受信側が送信側と受信側の両方を兼ねていない限り、受信側はチェックされません。

```
ADD A TO B
```

```
DIVIDE A INTO B
```

```
COMPUTE B = A + B
```

```
INITIALIZE B REPLACING ALPHANUMERIC BY B
```

このチェックは、各ステートメント内でデータが使用される前に実行されます。

- データが NOT NUMERIC の場合、ZONECHECK(MSG)の警告メッセージまたは ZONECHECK(ABD)の終了メッセージのいずれかが発行されます。
- データが NUMERIC の場合、ステートメントの外部動作は NOZONECHECK と同じですが、速度が遅くなる点異なります。

制約事項: CALL ステートメントの場合、ZONECHECK は BY CONTENT および BY VALUE 数値データ項目 USAGE DISPLAY のみを検査しますが、BY REFERENCE パラメーターは検査しません。

パフォーマンスの考慮事項。 ZONECHECK(MSG)および ZONECHECK(ABD)は、プログラム内の COBOL ステートメントで使用されるゾーン 10 進数データ項目の数によっては、NOZONECHECK よりもかなり遅くなります。

関連タスク

60 ページの『[非互換データの検査 \(数値のクラス・テスト\)](#)』

関連参照

422 ページの『[NUMCHECK](#)』

426 ページの『[NUMPROC](#)』

465 ページの『[ZONEDATA](#)』

ZONEDATA

の ZONEDATA オプションはコンパイラにデータは USAGE DISPLAY そして PACKED-DECIMAL データ項目が有効かどうか、また有効でない場合はコンパイラの動作がどうなっているかを示します。

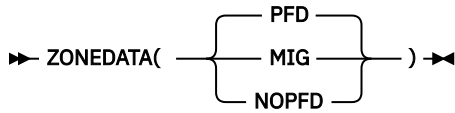
ほとんどのユーザーは有効なデータを USAGE DISPLAY そして USAGE PACKED-DECIMAL データ項目、ほとんどのユーザーは ZONEDATA(PFD)、たとえ彼らが使用していたとしても NUMPROC(NOPFD)。(NUMCHECK コンパイラー・オプションを使用して) 実行時にプログラムが無効なデータを処理していることがわかった場合でも、無効なデータの処理を回避して ZONEDATA(PFD) を使用するようにプログラムを変更する必要があります。

注: ZONEDATA オプションは非推奨ですが、互換性のために許容されており、INVDATA オプションに置き換えられています。

ZONEDATA オプションを指定すると、次のように同等の INVDATA オプションにマップされます。

```
ZONEDATA(PFD)      = NOINVDATA
ZONEDATA(NOPFD)    = INVDATA(NOFORCENUMCMP,CLEANSIGN) (or simply INVDATA)
ZONEDATA(MIG)      = INVDATA(FORCENUMCMP,CLEANSIGN)
```

ZONEDATA オプションの構文



デフォルトは、ZONEDATA(PFD)

省略形は ZD(PFD) | ZD(MIG) | ZD(NOPFD)

有効なゾーン 10 進数の各桁は、X'F0' から X'F9' の 1 バイトで表されます。各バイトの上位 4 ビットはゾーン・ビットであり、各バイトの下位 4 ビットにはその桁の値が含まれます。SIGN TRAILING の下位バイトの上位 4 ビットは、項目の符号を表します。符号は上位バイトにあり、SIGN LEADING、または別のバイトで SIGN IS SEPARATE。

ZONEDATA(PFD)

ZONEDATA(PFD) オプションが有効な場合、コンパイラは、USAGE DISPLAY データ項目および PACKED-DECIMAL データ項目にあるすべてのデータが有効であることを想定し、数値比較を行う上で可能な限り最も効率的なコードを生成します。例えば、コンパイラは数値変換を避けるために、ストリング比較を生成する場合があります。

ZONEDATA(MIG)

ZONEDATA(MIG) オプションが有効な場合、コンパイラは、ゾーン 10 進数データ項目内の各桁のゾーン・ビットを無視する数値比較を行うための命令を生成します。例えば、ゾーン 10 進数値は比較前に、PACK 命令によってパック 10 進数に変換されます。さらに、ゾーン 10 進数データ項目またはパック 10 進数データ項目に無効な数字や無効な符号コードが含まれている場合、またはゾーン 10 進数データ項目に無効なゾーン・ビットが含まれている場合、コンパイラは、COBOL 4 (またはそれ以前のバージョン) とは異なる結果を生成する可能性がある既知の最適化を実行しません。

ZONEDATA(NOPFD)

ZONEDATA(NOPFD) オプションが有効な場合、コンパイラは、COBOL 4 (またはそれ以前のバージョン) で NUMPROC(NOPFD | PFD) を使用するときと同じ方法で、ゾーン化された 10 進数データの数値比較または英数字比較の命令を生成する：

- COBOL 4 (またはそれ以前のバージョン) でゾーン・ビットの対象となっていた場合、コンパイラは英数字比較を生成し、英数字比較でもゾーン 10 進数データ項目内の各桁のゾーン・ビットが数値比較の対象となります。ゾーン 10 進数値はゾーン 10 進数のままです。
- COBOL 4 (またはそれ以前のバージョン) でゾーン・ビットが無視されていた場合、ゾーン 10 進数データ項目内の各桁のゾーン・ビットが無視される数値比較をコンパイラが生成します。ゾーン 10 進数値は比較前に、PACK 命令によってパック 10 進数に変換されます。

ゾーン化された 10 進数データの比較を生成するときに、コンパイラが COBOL 4 (またはそれ以前のバージョン) と同じ方法でゾーン・ビットを扱うためには、COBOL 6 で使用される NUMPROC サブオプションは、COBOL 4 (またはそれ以前のバージョン) で使用される NUMPROC サブオプションと一致しなければならない：

- COBOL 6 で COBOL 4 (またはそれ以前のバージョン) NUMPROC(NOPFD) の動作を得るには、COBOL 6 で ZONEDATA(NOPFD) と NUMPROC(NOPFD) を使用する。
- COBOL 6 で COBOL 4 (またはそれ以前のバージョン) NUMPROC(PFD) の動作を得るには、COBOL 6 で ZONEDATA(NOPFD) と NUMPROC(PFD) を使用する。

さらに、ゾーン 10 進数データ項目またはパック 10 進数データ項目に無効な数字や無効な符号コードが含まれている場合、またはゾーン 10 進数データ項目に無効なゾーン・ビットが含まれている場合、コンパイラは、COBOL 4 (またはそれ以前のバージョン) とは異なる結果を生成する可能性がある既知の最適化を実行しません。

注: 符号は、NUMPROC コンパイラ・オプション設定に準じた有効な符号でなければなりません。さらに、下位バイトには有効なゾーン (x'F') は符号なし、符号付きの場合は SIGN IS LEADING または SIGN IS SEPARATE。

注: ZONEDATA オプションは、無効な数字、無効な符号コード、または無効なゾーン・ビットを含む可能性がある USAGE DISPLAY または PACKED-DECIMAL データ項目の MOVE ステートメント、比較、および計算の動作に影響を与えます。

以下の例は、データ項目 VALUE1 の中間にあるゾーン・ビットに、無効なゾーン・ビット 4 が REDEFINES によって強制的に入れられているデータ項目を示しています。

```
77 VALUE0 PIC X(4) VALUE '00 0'.          *> x'F0F040F0'
77 VALUE1 REDEFINES VALUE0 PIC 9(4).
PROCEDURE DIVISION.
  IF VALUE1 = ZERO
    DISPLAY 'ZONEDATA(MIG) is in effect ' VALUE1
  ELSE
    DISPLAY 'ZONEDATA(NOPFD | PFD) is in effect ' VALUE1
  END-IF
```

この例では以下のとおりになります。

- COBOL 4 (またはそれ以前のバージョン) またはそれ以前のバージョンでは、NUMPROC(MIG) オプションが使用されている場合はテストが true になり、NUMPROC(NOPFD | PFD) の場合は false になります。
- COBOL 5 以降のバージョン:
 - 使用する場合 ZONEDATA(PFD)、テストは真である OPT(0)そして偽 OPT(1 | 2)。
 - ZONEDATA(NOPFD) が使用されている場合、テストはいずれの OPT 設定でも false です。

どの値を指定した場合でも、COBOL 6 へのマイグレーションを容易にするには、以下のようにします。

- 数字、符号コード、ゾーン・ビットが有効であれば、ZONEDATA(PFD)、COBOL 6 を使用するとき COBOL 4 (またはそれ以前のバージョン) で使用したのと同じ NUMPROC 設定を使用する。
- 無効な数字、無効な符号コード、または無効なゾーン・ビットがデータに含まれている場合は、プログラムまたはシステムを変更して、プログラムが実行時に数値データ項目に無効データを持たないようにします。

プログラムまたはシステムを訂正した後、優先 ZONEDATA(PFD) オプションを使用することができます。この作業を含めることができず、引き続き無効なデータを使用して実行する必要がある場合にのみ、ZONEDATA に対して以下の選択項目を検討してください。

- COBOL 4 (またはそれ以前のバージョン) で NUMPROC(MIG) を使用した場合は、COBOL 6 で ZONEDATA(MIG) と NUMPROC(NOPFD) を使用する。
- COBOL 4 (またはそれ以前のバージョン) で NUMPROC(NOPFD) を使用した場合は、COBOL 6 で ZONEDATA(NOPFD) と NUMPROC(NOPFD) を使用する。
- COBOL 4 (またはそれ以前のバージョン) で NUMPROC(PFD) を使用した場合は、COBOL 6 で ZONEDATA(NOPFD) と NUMPROC(PFD) を使用する。

注: 明らかに無効なデータが検出された場合、これらのオプションを使用したとしても、古いコンパイラの動作を完全に一致させることは常に可能というわけではありません。例えば、比較する場合でも、ZONEDATA(NOPFD) がすべてのケースで COBOL 4 と同じ結果をもたらすことはありません。

パフォーマンスの考慮事項、ZONEDATA(PFD)は、ZONEDATA(NOPFD | MIG)よりもランタイム・パフォーマンスが優れています。ZONEDATA(NOPFD | MIG)は、NUMPROC(PFD)が提供できる最適化の一部を使用不可にします。

関連タスク

60 ページの『[非互換データの検査 \(数値のクラス・テスト\)](#)』

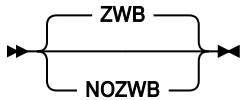
関連参照

422 ページの『[NUMCHECK](#)』

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



デフォルトは、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 内に含んでいる場合)、比較でその 10 進数項目を使用しても ZWB の影響を受けません。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。同一の COBOL プログラムでも、このオプションの設定に応じて異なる結果が生成されることがあります。

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。

COBOL ランタイム オプション

このセクションでは、COBOL ランタイム オプションを一覧表示します。

表 55. COBOL ランタイム オプション	
COBOL ランタイム オプション	目的
AIXBLD NOAIXBLD	VSAM 索引付きデータセットおよび相対データセットのアクセス方式サービス (AMS) を呼び出して、COBOL プログラムのファイルおよび索引定義手順を完了します。実行時に代替インデックスを構築します。
CBLOPTS	プログラム引数とランタイム オプションが期待される相対的な順序を指定します。デフォルトの CBLOPTS(ON) は、プログラム引数の後に「/」が続き、その後にランタイム オプションが続くことを意味します。
CBLPSHPOP	CICS PUSH HANDLE と CICS POP HANDLE を使用するかどうかを制御する。
CBLQDA	COBOL QSAM 動的割り振りを制御する。
CHECK	SSRANGE でコンパイルされた場合、範囲チェックが有効になります。コンパイルされたアプリケーションでは無視されます Enterprise COBOL5 以降。
DEBUG NODEBUG	USE FOR DEBUGGING 宣言で指定された COBOL バッチ デバッグ機能をアクティブにします。
流れ	OS/VS COBOL プログラムによって生成される FLOW 出力を制御します。

表 55. COBOL ランタイム オプション (続き)

COBOL ランタイム オプション	目的
ルテウス	ランタイム環境を暗黙的に初期化して再利用できるようにします。
SIMVRD NOSIMVRD	COBOL プログラムが VSAM KSDS を使用して可変長相対編成データセットをシミュレートするかどうかを指定します。
UPSI	COBOL プログラムを使用するアプリケーションの 8 つの UPSI スイッチをオンまたはオフに設定します。
AMODE3164	混合 AMODE COBOL プログラムを実行するときに指定する必要があります。
igzuqmsg=オン オフ	ユニークな COBOL ランタイム・メッセージのみを発する。IGZUQMSG=OFF がデフォルト。
igzrptmsg=オン オフ	COBOL ランタイム・メッセージ・レポートを作成する。IGZRPTMSG=OFF がデフォルト。
スキップワーフ=オン オフ	CEEDUMP 処理で情報を抑制します。 SKIPDWARF=OFF がデフォルトです。 538 ページの『CEEDUMP 処理での情報の抑止 (IGZ1OPT)』 を参照してください。
QSAMBUFFATL=オン オフ	境界より上 (ATL) に QSAM バッファを割り当てます。QSAMBUFFATL=OFF がデフォルトです。 見る「QSAM バッファの上限を要求しています (IGZ3OPT)」の中に <i>Enterprise COBOL</i> 移行ガイド。
QSAMBUFFINITCHAR=デフォルト スペース ビンゼロ	バッファ初期化文字。 QSAMBUFFINITCHAR=DEFAULT がデフォルトです。 見る「QSAM バッファの初期化の制御 (IGZ4OPT)」の中に <i>Enterprise COBOL</i> 移行ガイド。
VSAMDYNAMICDIR=オン オフ	動的アクセス モードを使用する VSAM KSDS/RRDS ファイルの読み取りロジックパスを選択できます。 VSAMDYNAMICDIR=OFF がデフォルトです。 475 ページの『VSAM 動的アクセス読み取りオプション VSAMDYNAMICDIR』 を参照してください。
DISABLEUOPTREPORT=オン オフ	LE ランタイム オプション RPTOPTS(ON) を指定すると、COBOL ランタイム オプション レポートがデフォルトで生成されます。このオプションを使用すると、このレポートをオフにすることができます。 DISABLEUOPTREPORT=OFF がデフォルトです。

これらのオプションの詳細については、「*z/OS Language Environment Programming Reference*」の「[Using Language Environment runtime options](#)」を参照してください。

関連参照

「実行時パフォーマンスに影響する実行時オプション」パフォーマンスチューニングガイド

COBOL ランタイム オプションの指定

このセクションでは、COBOL ランタイム環境で使用可能な COBOL ランタイム オプションを指定する方法について説明します。

- 言語環境(LE)オプションメソッドを介して。たとえば、`PARM=` または `CEEOPTS DD` ステートメント。詳細については、『*z/OS Language Environment プログラミング・ガイド*』の「[ランタイム・オプションの指定方法](#)」を参照のこと。
- `IGZOPTS DD` ステートメントまたは環境変数 `_IGZ_RUNOPTS` を介して。この方法で実行時オプションを指定する方法の詳細については、369 ページの『[COBOL 専用ランタイム・オプションの指定](#)』。
- `AMODE 31` の場合は `IGZUOPT`、`AMODE 64` の場合は `IGZQUOPT` と呼ばれるカスタマイズ可能な COBOL ランタイム オプション制御ブロックを介して。476 ページの『[IGZUOPT および IGZQUOPT ランタイム オプション制御ブロックの構築](#)』を参照してください。

各ランタイム オプションは、異なる方法で指定できます。次の表は、特定のランタイム オプションを指定する方法を示しています。

表 56. AMODE 31 COBOL ランタイム オプション			
	LE ランタイム・オプション	igzopts dd / _igz_runopts	イグズオプト
AIXBLD NOAIXBLD	X		
CBLOPTS	X		
CBLPSHPOP	X		
CBLQDA	X		
CHECK	X		
DEBUG NODEBUG	X		
FLOW NOFLOW	X		
RTEUSE ノルテレウス	X		
SIMVRD NOSIMVRD	X		
UPSI	X		
AMODE3164		X	
IGZUQMSG		X	
IGZRPTMSG		X	
スキップワーフ			X
QSAMBUFFATL			X
qsambuffinitchar さん			X
VSAM ダイナミックディレクトリ			X
レポートを無効にする			X

表 57. AMODE 64 COBOL ランタイム オプション			
	LE ランタイム・オプション	igzopts dd / _igz_runopts	IGZQUOPT
AIXBLD NOAIXBLD		X	
CBLQDA		X	

表 57. AMODE 64 COBOL ランタイム オプション (続き)

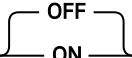
	LE ランタイム・オプション	igzopts dd / _igz_runopts	IGZQUOPT
DEBUG NODEBUG		X	
SIMVRD NOSIMVRD		X	
UPSI		X	
AMODE3164		X	
IGZUQMSG		X	
IGZRPTMSG		X	
スキップワープ			X
QSAMBUFFATL			X
qsambuffinitchar さん			X
VSAM ダイナミックディ レクトリ			X
レポートを無効にする			X
注: 次のランタイム オプションは AMODE 64 ではサポートされていません。 • CBLOPTS • CBLPSHPOP • CHECK • 流れ • ルテウス			

一意の COBOL ランタイム・メッセージ (IGZUQMSG)

IGZUQMSG は、実行時に一意の COBOL 実行時メッセージだけを出すかどうかを指定する。

IGZUQMSG(ON) では、複数のメッセージを発する必要がある場合、最初のユニークなメッセージが発せられるが、それ以降の同じメッセージは無視される。2つのメッセージは、同じ COBOL プログラム内で同じ位置で発行された場合、同一である。

IGZUQMSG オプションの構文

►► IGZUQMSG — (—  —) ◄◄

デフォルトのオプションは IGZUQMSG(OFF) です。

オフ

このオプションは、すべての COBOL ランタイム・メッセージを出力することを示す。

オン

このオプションは、ユニークな COBOL ランタイム・メッセージが発行されることを示す。

注:

- このオプションは、COBOL ランタイム・メッセージ (IGZnnn "nnn" が数値の場合、IGZ0279 など) を発行する COBOL 5.1 以降のリリースに適用される。

- COBOL 4.2 およびそれ以前のリリースでは、このオプションは黙って無視される。
- マルチスレッド環境では、このオプションは黙って無視される。
- IGZRPTMSG(ON) オプションは、繰り返されたメッセージと発せられたメッセージに関するレポートを生成するために使用される。

例

COBOL Source Code:

```
CBL NUMCHECK,OPT(0)
  IDENTIFICATION DIVISION.
  PROGRAM-ID. 'DEMO'.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 RES PIC S9(3).

  01 VAL1X                PIC X(3) VALUE X'12345F'.
  01 VAL1  REDEFINES VAL1X PIC 9(4) COMP-3.

  01 VAL2                PIC 9(4) COMP-3 VALUE 567.
  01 VAL3 PIC S9(3) VALUE -6.

  PROCEDURE DIVISION.
  PERFORM 2 TIMES
    COMPUTE RES = VAL1 + VAL2 *>IGZ0279W
    COMPUTE RES = VAL3 ** 2.3 *>IGZ0048W
  END-PERFORM
  STOP RUN.
END PROGRAM 'DEMO'.
```

以下の出力は、**IGZUQMSG(OFF)** オプションで実行時に生成される：

IGZ0279W とは、COBOL ソースコード内の コマンドのために 2 回発せられる。IGZ0048W PERFORM 2 TIMES

```
IGZ0279W The value X'12345F' of data item VAL1 at the time of reference by statement number 1
on line 16 in program DEMO failed the NUMERIC class test or contained a value larger than the
PICTURE clause as detected by the NUMCHECK compiler option.
```

```
IGZ0048W A negative base was raised to a fractional power in an exponentiation expression in
program DEMO at displacement X'240'. The absolute value of the base was used.
```

```
IGZ0279W The value X'12345F' of data item VAL1 at the time of reference by statement number 1
on line 16 in program DEMO failed the NUMERIC class test or contained a value larger than the
PICTURE clause as detected by the NUMCHECK compiler option.
```

```
IGZ0048W A negative base was raised to a fractional power in an exponentiation expression in
program DEMO at displacement X'240'. The absolute value of the base was used.
```

以下の出力は、**IGZUQMSG(ON)** オプションで実行時に生成される：

IGZ0279W と。IGZ0048W

```
IGZ0279W The value X'12345F' of data item VAL1 at the time of reference by statement number 1
on line 16 in program DEMO failed the NUMERIC class test or contained a value larger than the
PICTURE clause as detected by the NUMCHECK compiler option.
```

```
IGZ0048W A negative base was raised to a fractional power in an exponentiation expression in
program DEMO at displacement X'240'. The absolute value of the base was used.
```

レポート COBOL ランタイム・メッセージ (IGZRPTMSG)

IGZRPTMSG は、ランユニット終了時に COBOL ランタイム・メッセージ・レポートを生成するかどうかを指定する。

```
+++++ BEG OF COBOL RUN-TIME MESSAGE REPORT +++++
=====
= RUN-TIME MESSAGE LIST                                     =
=====
ID      PGM NAME ENTRY   OFFSET   LINEID   MSG ID   REPEATED   ISSUED
```

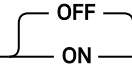
```
...
+++++ END OF MESSAGE Report +++++
```

IGZRPTMSG(ON) を使用すると、上記の COBOL ランタイム・メッセージ・レポートは、COBOL アプリケーションの開始時からの COBOL ランタイム・メッセージの呼び出し場所と発生回数に関する詳細を提供する。

AMODE 64 の場合、レポートは stderr ファイルハンドルに書き込まれる。

AMODE24 または AMODE31 の場合、非 CICS の場合、レポートは言語環境 MSGFILE ランタイム・オプションで指定された ddname に書き込まれます。詳細は [MSGFILE](#) を参照。で CICS で、レポートは CESE キューに書き込まれる。

IGZRPTMSG オプションの構文

► IGZRPTMSG — () ◄

デフォルトのオプションは IGZRPTMSG(OFF) です。

オフ

このオプションは、COBOL ランタイム・メッセージ・レポートを生成しないことを示す。

オン

このオプションは、COBOL ランタイム・メッセージ・レポートを生成することを示す。

注：

- このレポートには、COBOL 5 以降のリリースでのみ発行される COBOL ランタイム・メッセージが記録されている。
- このレポートには、COBOL 4.2 およびそれ以前のリリースで発行されたメッセージは記録されない。
- IGZRPTMSG(ON) オプションはマルチスレッド環境ではサポートされていないため、空のレポートとなる。
- IGZUQMSG(ON) オプションは、繰り返される同じメッセージを無視するために使われる。

例

COBOL Source Code:

```
CBL NUMCHECK,OPT(0)
  IDENTIFICATION DIVISION.
    PROGRAM-ID. 'DEMO'.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
    WORKING-STORAGE SECTION.
      01 RES PIC S9(3).

      01 VAL1X                PIC X(3) VALUE X'12345F'.
      01 VAL1 REDEFINES VAL1X PIC 9(4) COMP-3.

      01 VAL2                PIC 9(4) COMP-3 VALUE 567.
      01 VAL3 PIC S9(3) VALUE -6.

  PROCEDURE DIVISION.
    PERFORM 2 TIMES
      COMPUTE RES = VAL1 + VAL2 *>IGZ0279W
      COMPUTE RES = VAL3 ** 2.3 *>IGZ0048W
    END-PERFORM
  STOP RUN.
END PROGRAM 'DEMO'.
```

以下の出力は、**IGZRPTMSG(ON)** オプションを使用した実行時に生成される：

IGZ0279W とは、COBOL ソースコード内の コマンドのために 2 回発せられる。IGZ0048W PERFORM 2 TIMES

IGZPRTMSG(ON) オプションを指定すると、COBOL ランタイム・メッセージ・レポートが生成される。

IGZ0279W The value X'12345F' of data item VAL1 at the time of reference by statement number 1 on line 16 in program DEMO failed the NUMERIC class test or contained a value larger than the PICTURE clause as detected by the NUMCHECK compiler option.

IGZ0048W A negative base was raised to a fractional power in an exponentiation expression in program DEMO at displacement X'240'. The absolute value of the base was used.

IGZ0279W The value X'12345F' of data item VAL1 at the time of reference by statement number 1 on line 16 in program DEMO failed the NUMERIC class test or contained a value larger than the PICTURE clause as detected by the NUMCHECK compiler option.

IGZ0048W A negative base was raised to a fractional power in an exponentiation expression in program DEMO at displacement X'240'. The absolute value of the base was used.

```
+++++ BEG OF COBOL RUN-TIME MESSAGE REPORT +++++
=====
= RUN-TIME MESSAGE LIST                                     =
=====
ID      PGM NAME ENTRY   OFFSET   LINEID  MSG ID  REPEATED  ISSUED
-----
   1  DEMO      26A0C630 +00000240      0 IGZ0048      2      2
   2  DEMO      26A0C630 +00000450     16 IGZ0279      2      2
                                     -----
                                TOTAL      4      4
+++++ END OF MESSAGE Report +++++
```

以下の出力は、IGZUQMSG(ON) と IGZPRTMSG(ON) オプションを指定して実行時に生成される：

IGZUQMSG(ON) オプションでは、IGZ0279W と IGZ0048W が 2 回繰り返されるが、COBOL RUN-TIME MESSAGE REPORT の REPEATED と ISSUED 列に示されるように、1 回しか放出されない。

IGZ0279W The value X'12345F' of data item VAL1 at the time of reference by statement number 1 on line 16 in program DEMO failed the NUMERIC class test or contained a value larger than the PICTURE clause as detected by the NUMCHECK compiler option.

IGZ0048W A negative base was raised to a fractional power in an exponentiation expression in program DEMO at displacement X'240'. The absolute value of the base was used.

```
+++++ BEG OF COBOL RUN-TIME MESSAGE REPORT +++++
=====
= RUN-TIME MESSAGE LIST                                     =
=====
ID      PGM NAME ENTRY   OFFSET   LINEID  MSG ID  REPEATED  ISSUED
-----
   1  DEMO      26A0C630 +00000240      0 IGZ0048      2      1
   2  DEMO      26A0C630 +00000450     16 IGZ0279      2      1
                                     -----
                                TOTAL      4      2
+++++ END OF MESSAGE Report +++++
```

以下のセクションでは、COBOL RUN-TIME MESSAGE REPORT の各ヘッダーの意味を説明する：

Pgm 名

COBOL プログラム名。プログラム名が 8 文字より長い場合は、COBOL プログラムのレポートを作成するのと同じ方法で別々に表示される。

エントリー

PGM NAME という COBOL プログラムのエントリアドレス。

OFFSET

ENTRY、COBOL ランタイム・メッセージ内部ルーチンが呼び出される場所へのオフセット。このオフセットは、COBOL プログラム・リストを LIST オプションで生成したときに見つけることができる。

ラインアイディー

ランタイム・メッセージから抽出された COBOL ソース行番号。例えば、LINEID の 16 は '...on line 16...' から抽出された には、次のようなランタイム・メッセージがある：

```
IGZ0279W The value X'12345F' of data item VAL1 at the time of reference by statement number
1 on line 16 in program DEMO failed the NUMERIC class...
```

ランタイムメッセージに抽出すべき行番号はないが、DWARF 診断情報がある場合、DWARF の情報から LINEID が抽出される。DWARF については [TEST](#) を参照。

MSG ID :

COBOL ランタイムのメッセージ番号。

REPEATED

繰り返される COBOL ランタイムメッセージの番号。

ISSUED

発行される COBOL ランタイム・メッセージの番号。

VSAM 動的アクセス読み取りオプション VSAMDYNAMICDIR

VSAM の動的アクセス方式を VSAM エンジンへの直接キー読み取り要求に変更するには、COBOL ランタイム オプション VSAMDYNAMICDIR を指定する必要があります。この代替ロジック・パスは、zHyperLink を利用する IBM のハードウェア・ディスク・システムを利用する。zHyperLink アプリケーションの応答時間を改善し、I/O に敏感なワークロードの応答時間を大幅に短縮します。動的アクセス方法を変更しない場合、デフォルトの実行時動作は、キーによってレコードを指し示し、レコードを順番に読み取ることです。

注：VSAM の動的アクセス方法を変更することで、zHyperLink を利用することができる。zHyperLink がインストールされていない場合は、VSAMDYNAMICDIR を使用しないでください。zHyperLink が利用可能かどうかは、IBM z/OS ドキュメントの [zHyperLink I/O](#) を参照のこと。

重要：このオプションを利用するアプリケーションを制御するには、以下の手順に従ってください。[476 ページの『IGZUOPT および IGZQUOPT ランタイム オプション制御ブロックの構築』](#)。

この機能を使用するには、VSAMDYNMAICDIR=ON オプションを使用する IGZUOPT (AMODE 31 の場合) または IGZQUOPT (AMODE 64 の場合) COBOL ランタイム オプション モジュールを作成します。

IGZUOPT または IGZQUOPT モジュールを構築するときは、次の構文で VSAMDYNAMICDIR オプションを指定します。

```
IGZXOPT VSAMDYNAMICDIR=ON | OFF
```

VSAMDYNAMICDIR が指定されていない場合、デフォルトは VSAMDYNAMICDIR=OFF です。

詳細は、[476 ページの『IGZUOPT および IGZQUOPT ランタイム オプション制御ブロックの構築』](#)を参照してください。

VSAMDYNAMICDIR をオンにする方法については、言語環境サンプル・データ・セット .SCEESAMP のサンプル JCL IGZ5OPT を参照してください。

COBOL ランタイム オプション レポート オプション DISABLEUOPTREPORT を無効にする

LE ランタイム オプション RPTOPTS(ON) が指定されている場合、COBOL ランタイム オプション レポートが生成されます。COBOL ランタイム オプション レポートを抑制する必要がある場合は、DISABLEUOPTREPORT=ON を使用する IGZUOPT (AMODE 31 の場合) または IGZQOPT (AMODE 64 の場合) COBOL ランタイム オプション モジュールを作成できます。

IGZUOPT または IGZQOPT モジュールを構築するときは、次の構文で DISABLEUOPTREPORT オプションを指定します。

DISABLEUOPTREPORT が指定されていない場合、デフォルトは DISABLEUOPTREPORT=OFF です。

このオプションを変更する JCL ジョブの投入方法については、言語環境サンプル・データ・セット .SCEESAMP のサンプル JCL IGZ6OPT を参照してください。

IGZUOPT および IGZQUOPT ランタイム オプション制御ブロックの構築

IGZUOPT (AMODE 31 の場合) および IGZQUOPT (AMODE 64 の場合) モジュールは、COBOL プログラムの実行時に構築して提供するロード モジュールを参照します。これらのモジュールは、アセンブラ マクロ IGZLOPT を使用するアセンブラ プログラムです。

IGZLOPT マクロは、アセンブルしてから IGZUOPT または IGZQUOPT アプリケーションと呼ばれるロード モジュールにリンクする必要があるアセンブラ プログラムで使用されるアセンブラ マクロです。このロード モジュール IGZUOPT または IGZQUOPT は、実行中のアプリケーションの STEPLIB DD ステートメントに表示されるロード モジュール PDS で使用可能である必要があります。COBOL ランタイムは必要に応じてこのモジュールを検索し、モジュールの設定に基づいてランタイム オプション制御フラグを設定します。そのようなモジュールが見つからない場合は、IBM のデフォルトが優先される。

重要: 指定されたオプションを使用するアプリケーションの STEPLIB ストリームにのみ、このオプション制御ブロック ロード モジュールを追加する必要があります。特に、VSAMDYNAMICDIR オプションを使用する場合は、パフォーマンスに影響するため、どのアプリケーションでこの機能を利用する必要があるかを認識しておく必要があります。

COBOL IGZUOPT ランタイム オプションの JCL サンプル

コンパイラデータセット .SCEESAMPIGZUOPT COBOL ランタイム制御ブロックを構築する JCL サンプルが含まれています。JCL は ASSEMBLE ステップと LINK ステップを実行します。アセンブラは、IGZLOPT マクロを使用してカスタム COBOL ランタイム オプションを定義するサンプル アセンブラ コードをコンパイルします。LINK ステップは、実行時に COBOL アプリケーションに必要なロード モジュールを構築します。

表 58. COBOL ランタイム オプションと対応する JCL サンプル	
オプション	サンプル
スキップワープ	SCEESAMP(IGZ1OPT)
QSAMBUFFFFATL	SCEESAMP(IGZ3OPT)
qsambuffinitchar さん	SCEESAMP(IGZ4OPT)
VSAM ダイナミックディレクトリ	SCEESAMP(IGZ5OPT)
レポートを無効にする	SCEESAMP(IGZ6OPT)

関連タスク

476 ページの『[IGZUOPT および IGZQUOPT ランタイム オプション制御ブロックの構築](#)』

538 ページの『[CEEDUMP 処理での情報の抑止 \(IGZ1OPT\)](#)』

「QSAM バッファの上限を要求しています (IGZ3OPT)」の中に移行ガイド

「QSAM バッファの初期化の制御 (IGZ4OPT)」の中に移行ガイド

475 ページの『[VSAM 動的アクセス読み取りオプション VSAMDYNAMICDIR](#)』

475 ページの『[COBOL ランタイム オプション レポート オプション DISABLEUOPTREPORT を無効にする](#)』

COBOL ランタイム オプション レポート

APAR の PTF 付き PH56036 AMODE 31 の場合または PH56037 AMODE 64 がインストールされている場合、LE ランタイム オプション RPTOPTS(ON) を指定すると、COBOL ランタイム オプション レポートが自動的に生成されます。このレポートは、LE ランタイム オプション レポートに加えて表示されます。

このレポートを生成する必要がある場合は、COBOL カスタマイズ・ランタイム・オプション
DISABLEUOPTREPORT=ON を指定する IGZUOPT (AMODE 31 用) または IGZQUOPT (AMODE 64 用)
モジュールを作成する必要があります。

COBOL ランタイム オプション レポートには、IGZOPTS DD、_IGZ_RUNOPTS、または IGZUOPT または
IGZQUOPT モジュールによって入力されたランタイム オプションのみがリストされます。

注: COBOL ランタイム オプション レポートでは、LAST WHERE SET 列に次の状態が示されます。

- 「LE オプション設定から」: IGZUOPT モジュールが見つかりません。すべての設定は、ベース LE オプシ
ョン設定にデフォルト設定されます。
- 「IGZUOPT」: IGZUOPT モジュールが見つかり、そのモジュールから設定が取得されました。
- 「DD:IGZOPTS」: ランタイム オプションは、IGZOPTS DD ステートメントによって指定または検出されま
した。
- "_IGZ_RUNOPTS": ランタイム オプションは、_IGZ_RUNOPTS 環境変数を介して指定または検出されま
した。

例

次の表に、ランタイム オプションのシナリオ例を示します。

表 59. AMODE 31 ランタイムシナリオの例		
数値	シナリオ	結果
例 1	IGZUOPT モジュールが見つかり ません。 RPTOPTS(ON)、aixbld、および DEBUG が指定されました。	COBOL と LE の両方のランタイム オプション レポートが生成され ます。
例 2	IGZUOPT モジュールが見つかり ました。 RPTOPTS(ON)、aixbld、DEBUG、 および DISABLEUOPTREPORT=ON が指 定されました。	LE ランタイム オプション レポー トのみが生成されます。

例 1 のレポート出力は次のとおりです。

```
COBOL Run-time Options Report ...
This report supplements Language Environment Options Report
```

LAST WHERE SET	OPTION
From LE Option setting	NOAMODE3164
From LE Option setting	NOSKIPDWARF
From LE Option setting	NOQSAMBUFATL
From LE Option setting	BUFFINITCHAR(DEFAULT)
From LE Option setting	NOVSAMDYNAMICDIR
From LE Option setting	NODISABLEUOPTREPORT

End of COBOL Run-time Option Report

```
Options Report for Enclave ...
Language Environment V02 R04.00
```

LAST WHERE SET	OPTION
IBM-supplied default	ABPERC(NONE)
IBM-supplied default	ABTERMENC(ABEND)
Invocation command	AIXBLD
IBM-supplied default	ALL31(ON)
...	
IBM-supplied default	COUNTRY(US)


```
Invocation command      DEBUG
...
```

例 2 の出力は次のとおりです。

```
Options Report for Enclave ...
Language Environment V02 R04.00
```

```
LAST WHERE SET          OPTION
-----
IBM-supplied default    ABPERC(NONE)
IBM-supplied default    ABTERMENC(ABEND)
Invocation command      AIXBLD
IBM-supplied default    ALL31(ON)
...
IBM-supplied default    COUNTRY(US)
Invocation command      DEBUG
...
```

表 60. AMODE 64 ランタイムシナリオの例

数値	シナリオ	結果
例 3	IGZQUOPT モジュールが見つかった。 VSAMDYNAMICDIR=ON が指定されました。	COBOL と LE の両方のランタイムオプションレポートが生成されます。
例 4	IGZQUOPT モジュールが見つかった。 DISABLEUOPTREPORT=ON が指定されました	LE ランタイム オプション レポートのみが生成されます。

例 3 のレポート出力は以下の通り：

```
COBOL Run-time Options Report ...
This report supplements Language Environment Options Report
```

```
LAST WHERE SET          OPTION
-----
From LE Option setting  NOAIXBLD
From LE Option setting  NOSIMVRD
From LE Option setting  NODEBUG
From LE Option setting  CBLQDA(OFF)
From LE Option setting  NOAMODE3164
IGZQUOPT                NOSKIPDWARF
IGZQUOPT                NOQSAMBUFATL
IGZQUOPT                BUFFINITCHAR(DEFAULT)
IGZQUOPT                VSAMDYNAMICDIR
IGZQUOPT                NODISABLEUOPTREPORT
```

End of COBOL Run-time Option Report

```
Options Report for Enclave PGMA ...
Language Environment V02 R04.00
```

```
LAST WHERE SET          OPTION
-----
IBM-supplied default    CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC)
PARMLIB(CEEPRM0W)       DYNDUMP(*USERID,NODYNAMIC,NOTDUMP)
IBM-supplied default    ENVAR("")
...
```

例 4 のレポート出力は次のとおりです。

```
Options Report for Enclave PGMA ...
Language Environment V02 R04.00
```

```
LAST WHERE SET          OPTION
-----
IBM-supplied default    CEEDUMP(60,SYSOUT=*,FREE=END,SPIN=UNALLOC)
PARMLIB(CEEPRM0W)       DYNDUMP(*USERID,NODYNAMIC,NOTDUMP)
```

```
IBM-supplied default      ENVAR("")  
...
```


第 21 章 コンパイラ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のに役立つステートメントがいくつかあります。

以下のコンパイラ指示ステートメントがあります。

BASIS ステートメント

この拡張ソース・プログラム・ライブラリー・ステートメントは、コンパイルのソースとして、完全な COBOL プログラムを提供します。形式化および処理の規則については、COPY ステートメントの *text-name* の説明を参照してください。

CALLINTERFACE 指示

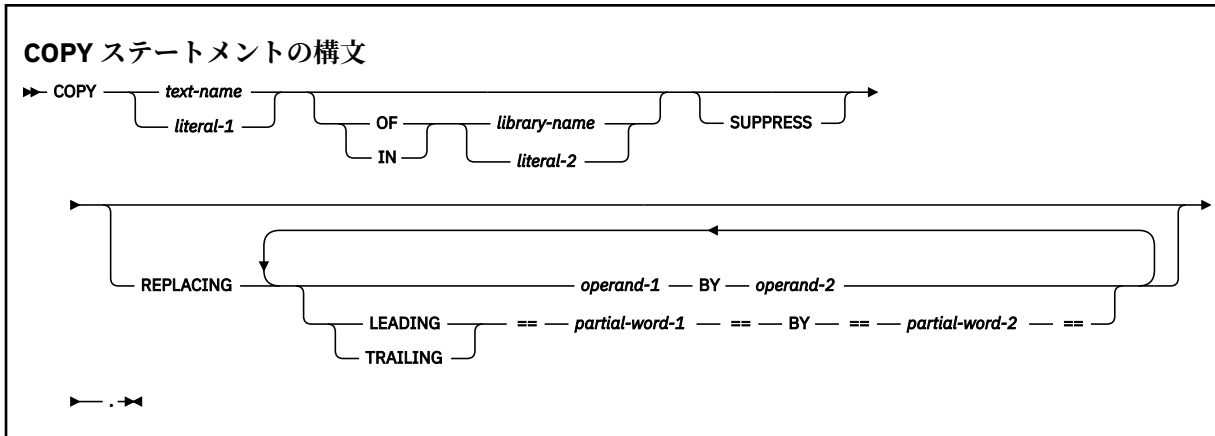
CALLINTERFACE 指示は、CALL ステートメントおよび SET ステートメントのインターフェース規約を指定します。指定した規約は、ソース内で別の CALLINTERFACE ディレクティブが検出されるまで有効のままになります。

CALLINTERFACE 指示は手続き部でのみ使用でき、その効果は現行コンパイル単位に限定されます。

*CONTROL (*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、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。名前の *CONTROL と *CBL は同義語です。

COPY ステートメント



このライブラリー・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。

テキスト名もライブラリー名もプログラム内で固有である必要はありません。これらは、プログラム内の他のユーザー定義語と同じにすることができますが、下線は使用できません。

テキスト名およびライブラリー名の固有性は、システム依存名の形成規則と変換規則が適用された後に判定されます。library-name を省略した場合は、SYSLIB と見なされます。

JCL を使用したコンパイル:

text-name、library-name、および literal-1 および literal-2 は、次のように処理されます。

- 名前 (1 文字から 30 文字までの長さ) は 8 文字に切り捨てられます。text-name および library-name の最初の 8 文字だけが、識別名として使用されます。これらの 8 文字は、COBOL ライブラリー内で固有でなければなりません。
- 名前は大文字に変換されます。
- 先頭文字でも最後の文字でもないハイフンはゼロ (0) に変換され、警告メッセージが出されます。
- 先頭文字が数値である場合、文字 1 から 9 は A から I へ変換され、ゼロ (0) は J へ変換され、警告メッセージが出されます。

例:

```
COPY INVOICES1Q
COPY "Company-#Employees" IN Personellib
```

IN/OF 句の中の *library-name* は、コピー元の区分データ・セットを識別する DD 名です。次の例に示すように、DD ステートメントを使用して、*library-name* を定義します。

```
//COPYLIB DD DSNAME=ABC.COB,VOLUME=SER=111111,
//          DISP=SHR,UNIT=3380
```

複数のコピー・ライブラリーを指定するには、JCL または JCL と IN/OF 句の組み合わせのいずれかを使用してください。JCL だけを使用する場合は、SYSLIB に対する DD ステートメント上でデータ・セットを連結します。あるいは、複数の DD ステートメントを定義し、COPY ステートメントに IN/OF 句を組み込みます。

コピー・ライブラリーの最大ブロック・サイズは、データ・セットが常駐している装置によって決まります。

z/OS ユニックス シェルでコンパイルするとき:

cob2 コマンドを使用してコンパイルすると、z/OS UNIX ファイルシステムからコピーブックが組み込まれます。*text-name*、*library-name*、および *literal-1* および *literal-2* は、次のように処理されます。

- ユーザー定義語は大文字に変換されます。リテラルは変換されません。UNIX は大/小文字の区別をするので、ファイル名が小文字であるか大/小文字混合である場合は、それをリテラルとして指定します。
- *text-name* がリテラルのとき、*library-name* が省略されていると、*text-name* は直接的に使用されます。すなわち、ファイル名、相対パス名、または絶対パス名 (先頭文字が / の場合) として使用されます。例:

```
COPY "MyInc"
COPY "x/MyInc"
COPY "/u/user1/MyInc"
```

- *text-name* がユーザー定義語で、その名前の環境変数が定義されていると、その環境変数の値が、コピーブックを含むファイルの名前として使用されます。

その名前の環境変数が定義されていない場合には、コピーブックは、以下の名前として、この順に探索されます。

1. *text-name.cpy*
2. *text-name.CPY*
3. *text-name.cbl*
4. *text-name.CBL*
5. *text-name.cob*
6. *text-name.COB*
7. *text-name*

- *library-name* がリテラルである場合は、それはコピー・ファイル *text-name* までの実際のパス (相対または絶対) として扱われます。
- *library-name* がユーザー定義語であると、それは環境変数として扱われます。その環境変数の値がパスとして使用されます。環境変数が設定されていないと、エラーとなります。

- *library-name* と *text-name* の両方が指定された場合、コンパイラーは、*library-name* と *text-name* の間にパス区切り文字 (/) を入れて 2 つの値を連結することによって、コピーブックのパス名を形成します。例えば、次のような設定があるとします。COPY MYCOPY OF MYLIB:

```
export MYCOPY=mystuff/today.cpy
export MYLIB=/u/user1
```

この設定の結果、次のようになります。

```
/u/user1/mystuff/today.cpy
```

library-name が環境変数で、この環境変数の値がコピーブックのコピー元のパスを識別する場合は、次の例のように、**export** コマンドを使用して *library-name* を定義します。

```
export COPYLIB=/u/mystuff/copybooks
```

環境変数の名前は大文字でなければなりません。複数のコピー・ライブラリーを指定するには、コロン (:) で区切った複数のパス名を、環境変数に設定してください。

library-name が省略されたときに、*text-name* が絶対パス名でないと、コピーブックは、次の順序で探索されます。

1. 現行ディレクトリー
2. -I cob2 オプションで指定されたパス
3. SYSLIB 環境変数で指定されたパス

テキスト置換規則など、COPY ステートメントに関する追加情報については、関連参照を参照してください。

DEFINE 指示

DEFINE ディレクティブはコンパイル変数を定義/定義解除します。コンパイル変数はいずれかの条件付きコンパイル・ディレクティブ (DEFINE、EVALUATE、および IF) の内部で使用できます。コンパイル変数は、現在示しているリテラル値に対するシンボル参照として扱われます。

DELETE ステートメント

この拡張ソース・ライブラリー・ステートメントは、BASIS ソース・プログラムから COBOL ステートメントを除去します。

EJECT 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、次のソース・ステートメントが次のページの最上部に印刷されるように指定します。

ENTER ステートメント

ステートメントは、コメントとして扱われます。

EVALUATE 指示

EVALUATE ディレクティブは、コンパイル・グループに組み込むソース行を選択する分岐方式を提供します。

IF 指示

IF ディレクティブは、片方向または両方向の条件付きコンパイルを提供します。

INLINE 指示

INLINE ディレクティブを使用すれば、コンパイラーがプロシージャをインライン化に適格とみなすことを選択的に防止することができます。

INSERT ステートメント

このライブラリー・ステートメントは、COBOL ステートメントを BASIS ソース・プログラムに追加します。

PROCESS (CBL) ステートメント

このステートメントは、最外部プログラムの IDENTIFICATION DIVISION ヘッダーの前に指定するもので、プログラムのコンパイルで使用するコンパイラー・オプションを示します。

REPLACE ステートメント

このステートメントは、ソース・プログラム・テキストを置き換えるために使用されます。

SERVICE LABEL ステートメント

このステートメントは、制御の流れを示すために CICS 変換プログラムによって生成され、CEE3SRP の呼び出しのために再開点で使用する必要があります。一般的に使用されるものではありません。

SKIP1/2/3 ステートメント

このステートメントは、ある行がソース・リストでスキップされることを示します。

TITLE ステートメント

このステートメントは、ソース・リストの各ページの最上部に表題 (ヘッダー) が印刷されるように指定します。

USE ステートメント

USE ステートメントは、以下のエレメントを指定するための宣言を提供します。

- エラー処理プロシージャ: EXCEPTION/ERROR
- デバッグ行およびセクション: DEBUGGING

関連タスク

[5 ページの『ソース・リストのヘッダーの変更』](#)

[299 ページの『z/OS のもとでのコンパイラー・オプションの指定』](#)

[312 ページの『z/OS ユニックス のもとでのコンパイラー・オプションの指定』](#)

[311 ページの『z/OS ユニックス のもとでの環境変数の設定』](#)

[852 ページの『反復コーディングの除去』](#)

関連参照

[315 ページの『cob2 の構文およびオプション』](#)

CALLINTERFACE (*Enterprise COBOL for z/OS* 言語解説書)

DEFINE (*Enterprise COBOL for z/OS* 言語解説書)

EVALUATE (*Enterprise COBOL for z/OS* 言語解説書)

IF (*Enterprise COBOL for z/OS* 言語解説書)

INLINE (*Enterprise COBOL for z/OS* 言語解説書)

COPY ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

第 22 章 デバッグ

アプリケーションの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 つの異なる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。

プログラムの問題が容易に検出されず、利用できるデバッガーがない場合、プログラムのストレージ・ダンプを分析する必要があります。

対話式デバッグの場合は、z/OS デバッガを使用できます。z/OS デバッガは、次のような生産性拡張機能を提供します。

- 対話式デバッグ (フルスクリーン・モードまたは行モード) またはバッチ・モードのデバッグ
対話式フルスクリーン・モード・セッション中に、3270 装置上のフルスクリーン・サービスとセッション・パネル・ウィンドウを使用して、実行中のプログラムをデバッグすることができます。
- COBOL 類似コマンド
サポートされる高水準言語ごとに、ブレークポイントで取るべきアクションを指定するコマンドが、そのプログラム言語に類似した構文で提供されます。
- 混合言語デバッグ
異なる言語で作成されたプログラムが入っているアプリケーションをデバッグすることができます。z/OS デバッガが、実行中のプログラムまたはサブプログラムの言語を自動的に判別します。
- COBOL-CICS デバッグ
z/OS デバッガは、対話式モードとバッチ・モードの両方で CICS アプリケーションのデバッグをサポートします。
- リモート・デバッグのサポート
Eclipse ベースのデバッグ・インターフェースおよび IBM Z Open Debug を使用して、z/OS 上で実行されるプログラムをデバッグすることができます。

注：IBM z/OS デバッガは、以下の製品で提供される共通コンポーネントです。

- IBM Debug for z/OS (旧称: IBM Debug Tool for z/OS)
- IBM Developer for z/OS (旧称 IBM Developer for z Systems および Rational Developer for z Systems)
- IBM Developer for z/OS (Enterprise Edition)。これは、IBM Application Delivery Foundation for z/OS パンドルにも提供されています。
- IBM Z and Cloud Modernization Stack

異なる製品で提供されるデバッグ機能は異なります。詳細情報は、[IBM z/OS Debugger の概要](#)を参照してください。

COBOL アプリケーションを作成するために使用している Enterprise COBOL のレベルに応じて、デバッガーを特定のバージョンに更新してください。

- z/OS Debugger xxx 以降は、Enterprise COBOL 6.5 およびそれ以前のバージョンをサポートしています
- z/OS Debugger 15.0.5 以降は、Enterprise COBOL 6.4 以前のバージョンをサポートします
- z/OS Debugger 14.2 以降は、Enterprise COBOL 6.3 以前のバージョンをサポートします
- z/OS Debugger 14.1 は、Enterprise COBOL 6.2 以前のバージョンをサポートします
- z/OS Debugger 14.0 は、Enterprise COBOL 6.1 以前のバージョンをサポートします
- Debug Tool 13.1 は、Enterprise COBOL 5.1 以前のバージョンをサポートします。

関連タスク

[486 ページの『ソース言語によるデバッグ』](#)

[490 ページの『コンパイラー・オプションを使用したデバッグ』](#)

[499 ページの『デバッガーの使用』](#)
[500 ページの『リストの入手』](#)
[538 ページの『CEEDUMP 処理での情報の抑止 \(IGZ1OPT\)』](#)
[IBM z/OS デバッガ・ユーザー・ガイド](#)

関連参照

[IBM z/OS デバッガ・リファレンスとメッセージ](#)
[言語環境デバッグガイド \(システム・ダンプのフォーマットと分析、サンプル COBOL プログラムのデバッグ\)](#)

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。

このタスクについて

障害のあるプログラムが既の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
- プロシージャの問題

関連タスク

[486 ページの『プログラム・ロジックのトレース』](#)
[487 ページの『入出力エラーの検出および処理』](#)
[487 ページの『データの妥当性検査』](#)
[488 ページの『初期化されていないデータの移動、初期化、または設定』](#)
[488 ページの『プロシージャに関する情報の生成』](#)

関連参照

[ソース言語のデバッグ \(Enterprise COBOL for z/OS 言語解説書\)](#)

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。

このタスクについて

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、使用したデバッグ・エイドを削除するか使用不可にして、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 (*Enterprise COBOL for z/OS 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。

このタスクについて

ファイル状況キーをデバッグで使用するには、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を確認します。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チャー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チャー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チャー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。

それぞれをコード化する USE AFTER STANDARD ERROR セクション内のステートメントこれに続く DECLARATIVES キーワードの中に PROCEDURE DIVISION。

関連タスク

270 ページの『[ERROR 宣言のコーディング](#)』

271 ページの『[ファイル状況キーの使用](#)』

関連参照

ファイル状況キー (*Enterprise COBOL for z/OS 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。

このタスクについて

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。

UVALID 組み込み関数を使用して、国別データ項目に有効な UTF-16 エンコード・データが含まれているかどうか、あるいは英数字項目または英字項目に有効な UTF-8 エンコード・データが含まれているかどうかを検査することができます。

関連タスク

[103 ページの『条件式のコーディング』](#)

[169 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 (*Enterprise COBOL for z/OS 言語解説書*)

UVALID (*Enterprise COBOL for z/OS 言語解説書*)

初期化されていないデータの移動、初期化、または設定

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。

このタスクについて

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 (*Enterprise COBOL for z/OS 言語解説書*)

SET ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

[405 ページの『INITCHECK』](#)

プロシージャ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。

このタスクについて

たとえば、プロシージャが何回実行されるかを判断するには、USE FOR DEBUGGING 宣言型であり、カウンターを使用して、そのプロシージャに制御が渡される回数を追跡します。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、7 桁目の D によって識別されるステートメントです。プログラム内のデバッグ行をアクティブにするには、WITH DEBUGGING MODE 条項に関する SOURCE-COMPUTER ラインの ENVIRONMENT DIVISION。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。それぞれをコード化する USE FOR DEBUGGING 別のセクションで宣言します。デバッグ・ステートメントは、次のようにコーディングします。

- 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 条項を使用し、DEBUG ランタイムオプション。

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。

489 ページの『例: USE FOR DEBUGGING』

関連参照

SOURCE-COMPUTER 段落 (*Enterprise COBOL for z/OS* 言語解説書)

デバッグ行 (*Enterprise COBOL for z/OS* 言語解説書)

デバッグ・セクション (*Enterprise COBOL for z/OS* 言語解説書)

DEBUGGING 宣言 (*Enterprise COBOL for z/OS* 言語解説書)

例: USE FOR DEBUGGING

この例では、DISPLAY 声明と USE FOR DEBUGGING プログラムをテストするための宣言型。

DISPLAY ステートメントは、情報を端末または出力データ・セットに書き込みます。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.
```

の DISPLAY 声明文 DECLARATIVES SECTION このメッセージは手順ごとに表示されます Some-Routine 実行:

```
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。

このタスクについて

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。

- 初期設定されずに使用されるデータ項目 (INITCHECK)
- 重複データ名のような構文エラー (NOCOMPILE)
- 送信データ項目として使用される無効なデータ項目 (NUMCHECK)
- 無効な COBOL プログラム (PARMCHECK)
- セクションの欠落 (SEQUENCE)
- 無効な添え字値 (SSRANGE)
- z/OS で動作しなくなったプログラム 2.5 + (LSACHECK)

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・エンティティー定義および参照。COPY または BASIS ステートメントからのテキスト名およびライブラリー名、およびコピーブックを取得する関連データ・セットまたはファイル (XREF)
- DATA DIVISION 内のデータ項目 (MAP)
- ステートメント参照 (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) を取得できます。

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

[490 ページの『コーディング・エラーの検出』](#)

[491 ページの『行シーケンス問題の検出』](#)

[491 ページの『無効な COBOL データや無効な COBOL プログラムの検査』](#)

[492 ページの『有効範囲の検査』](#)

[492 ページの『診断するエラーのレベルの選択』](#)

[494 ページの『プログラム・エンティティー定義および参照の検出』](#)

[495 ページの『データ項目のリスト』](#)

[500 ページの『リストの入手』](#)

関連参照

[371 ページの『COBOL 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。

このタスクについて

TSO フォアグラウンドでコンパイルする場合、TERM コンパイラー・オプションを使用し、ユーザーのデータ・セットを SYSTERM データ・セットとして定義することにより、メッセージを画面に送信することができます。

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。すべてのエラーが **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

[388 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。

このタスクについて

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られます。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

[439 ページの『SEQUENCE』](#)

無効な COBOL データや無効な COBOL プログラムの検査

ご使用のプログラムで実行時に処理される COBOL データが無効かどうかを判別するには、**INITCHECK** および **NUMCHECK** を使用します。ご使用のプログラムで実行時にパラメーターが一致しないために **WORKING-STORAGE** の終わりを越えたデータが破壊されるかどうかを判別するには、**PARMCHECK** を使用します。

このタスクについて

INITCHECK オプションは、初期化されていないデータ項目がないかを調べ、データ項目が初期化されずに使用されていると警告メッセージを発行します。

NUMCHECK オプションは、以下のデータ項目が送信データ項目として使用されている場合に、そのデータ項目を妥当性検査します。

- **NUMCHECK(ZON)** が指定されると、コンパイラーは、COBOL ステートメントで送信データ項目として使用されるゾーン 10 進数 (数値 **USAGE DISPLAY**) データ項目に対して暗黙的な数値クラス・テスト用のコードを生成します。
- **NUMCHECK(PAC)** が指定されると、コンパイラーは、COBOL ステートメントで送信データ項目として使用されるパック 10 進数 (**COMP-3**) データ項目に対して暗黙的な数値クラス・テスト用のコードを生成します。
- **NUMCHECK(BIN)** が指定されると、コンパイラーは、バイナリー・データ項目の内容が **PICTURE** 節より大きいかどうかをテストするために、**ON SIZE ERROR** に似たコードを生成します。この追加コードは、送信データ項目として使用されるバイナリー・データ項目に対してのみ生成されます。 **COMP-5** データ項目の場合、この **ON SIZE ERROR** コードは生成されないことに注意してください。

PARMCHECK オプションは、WORKING-STORAGE の終わりを超えて書き込みを行うサブプログラムを検出します。このオプションが指定されると、コンパイラーは、WORKING-STORAGE において最後の項目の後に続く余分なデータ項目を生成します。その余分なデータ項目は、呼び出されたサブプログラムが、WORKING-STORAGE の終わりを超えたデータを破壊したかどうかを検査するために実行時に使用されます。

パフォーマンスの考慮: PARMCHECK や NUMCHECK が指定されると、パフォーマンスが多少低下する可能性があります。これは、無効な COBOL データがないかを調べるためのオーバーヘッドが余計にかかるためです。PARMCHECK が指定されると、コンパイラーは、CALL ステートメントを持つプログラムに対して低速のコードを生成します。NUMCHECK が更新され、無効データの冗長検査が取り除かれたため、実行時パフォーマンスが向上しました。ただし、NONUMCHECK を使用したときのパフォーマンスが最高であり、OPT(1) および OPT(2) で、OPT(0) のときよりも向上します。

関連参照

[405 ページの『INITCHECK』](#)

[422 ページの『NUMCHECK』](#)

[431 ページの『PARMCHECK』](#)

[845 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』](#)

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。

このタスクについて

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: 含まれるグループの最大境界内に、指定されたテーブル・エレメントの有効アドレスがあるかどうか。（このチェックは UNBOUNDED テーブルとグループ。
- 可変長データ参照 (OCCURS DEPENDING ON 節を含むデータ項目への参照): 実際の長さがゼロ以上であるかどうか、またグループ・データ項目に定義された最大長内にあるかどうか。（このチェックは UNBOUNDED グループ。
- 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、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場合は、実行時に検査が行われます。

参照されるデータを含むデータ項目の範囲外に有効アドレスがある場合は、エラー・メッセージが生成され、プログラムは停止します。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

[446 ページの『SSRANGE』](#)

[845 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。

このタスクについて

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 61. 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。

注: EXIT オプションの MSGEXIT サブオプションを指定すると、一部のエラー・メッセージを抑制し、その他のエラー・メッセージの重大度を変更できます。

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。

493 ページの『例: 組み込みメッセージ』

関連タスク

307 ページの『コンパイラー・メッセージのリストの生成』

関連参照

308 ページの『コンパイラー診断メッセージの重大度コード』

307 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』

402 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
...
090671**      /
090672**      *****
090673**      ***          I N I T I A L I Z E      P A R A G R A P H          **
090674**      ***  Open files. Accept date, time and format header lines.      **
090675**      ***  Load location-table.      **
090676**      *****
090677**      100-initialize-paragraph.
090678**      move spaces to ws-transaction-record                                IMP
331
090679**      move spaces to ws-commuter-record                                IMP
307
090680**      move zeroes to commuter-zipcode                                IMP
```

```

318      090681**          move zeroes to commuter-home-phone          IMP
319      090682**          move zeroes to commuter-work-phone          IMP
320      090683**          move zeroes to commuter-update-date          IMP
324      090684**          open input update-transaction-file          204
==090684==> IGYPS2052-S An error was found in the definition of file "LOCATION-FILE". The
                    reference to this file was discarded.
      090685**          location-file          193
      090686**          i-o commuter-file          181
      090687**          output print-file          217
      090688**          if commuter-file-status not = "00" and not = "97"          241
      090689**      1      display "100-OPEN"
      090690**      1      move 100 to comp-code          231
      090691**      1      perform 500-vsam-error          91069
      090692**      1      perform 900-abnormal-termination          91114
      090693**          end-if
      090694**          accept ws-date from date          UND
==090694==> IGYPS2121-S "WS-DATE" was not defined as a data-name. The statement was discarded.
      090695**          move corr ws-date to header-date          UND
455
==090695==> IGYPS2121-S "WS-DATE" was not defined as a data-name. The statement was discarded.
      090696**          accept ws-time from time          UND
==090696==> IGYPS2121-S "WS-TIME" was not defined as a data-name. The statement was discarded.
      090697**          move corr ws-time to header-time          UND
449
==090697==> IGYPS2121-S "WS-TIME" was not defined as a data-name. The statement was discarded.
      090698**          read location-file          193
==090698==> IGYPS2053-S An error was found in the definition of file "LOCATION-FILE". This
                    input/output statement was discarded.
      090699**          at end
      090700**      1      set location-eof to true          256
      090701**          end-read
...
LineID  Message code  Message text
      IGYSC0090-W  1700 sequence errors were found in this program.
      IGYSC3002-I  A severe error was found in the program. The "OPTIMIZE" compiler option was
cancelled.
      160  IGYDS1089-S  "ASSIGNN" was invalid. Scanning was resumed at the next area "A" item, level-
number, or
      193  IGYGR1207-S  The "ASSIGN" clause was missing or invalid in the "SELECT" entry for file
"LOCATION-FILE".
      269  IGYDS1066-S  The file definition was discarded.
item.      "REDEFINES" object "WS-DATE" was not the immediately preceding level-1 data
      90602  IGYPS2052-S  The "REDEFINES" clause was discarded.
this file  An error was found in the definition of file "LOCATION-FILE". The reference to
      90694  IGYPS2121-S  was discarded. Same message on line: 90684
"WS-DATE" was not defined as a data-name. The statement was discarded.
      90696  IGYPS2121-S  Same message on line: 90695
"WS-TIME" was not defined as a data-name. The statement was discarded.
      90698  IGYPS2053-S  Same message on line: 90697
output statement  An error was found in the definition of file "LOCATION-FILE". This input/
was discarded. Same message on line: 90709
Messages      Total      Informational      Warning      Error      Severe      Terminating
Printed:      13          1          1          11
* Statistics for COBOL program IGYTCARA:
*   Source records = 1755
*   Data Division statements = 295
*   Procedure Division statements = 479
*   Generated COBOL statements = 0
*   Program complexity factor = 486
End of compilation 1, program IGYTCARA, highest severity 12.
Return code 12

```

プログラム・エンティティ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データ・セットまたはファイルへの、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。

このタスクについて

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れられます。

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関連参照を参照してください。

531 ページの『例: XREF 出力: データ名相互参照』

533 ページの『例: XREF 出力: プログラム名相互参照』

533 ページの『例: XREF 出力: COPY/BASIS 相互参照』

535 ページの『例: XREF 出力: 組み込み相互参照』

関連タスク

500 ページの『リストの入手』

関連参照

463 ページの『外部参照』

データ項目のリスト

MAP (HEX|DEC) コンパイラー・オプションを使用して、DATA DIVISION 項目および暗黙的に宣言されたすべての項目のリストを作成します。MAP 出力を使用して、システム・ダンプのデータ項目の内容を突き止めます。

このタスクについて

MAP (HEX|DEC) オプションを指定すると、圧縮 MAP 情報を含む組み込み MAP 要約が、COBOL ソース・データ定義の右側に生成されます。

- MAP (HEX) または、サブオプションなしの MAP を指定した場合、グループ内のデータ項目オフセットは 16 進表記になります。
- MAP (DEC) を指定した場合、グループ内のデータ項目オフセットは 10 進表記になります。

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。

一部を選択または禁止することができます MAP リストと埋め込み MAP 要約*CONTROL MAP|NOMAP (または*CBL MAP|NOMAP) ステートメントをソース全体に含めます。例:

```
*CONTROL NOMAP
  01  A
  02  B
*CONTROL MAP
```

505 ページの『例: MAP 出力』

関連タスク

500 ページの『リストの入手』

関連参照

418 ページの『MAP』

AMODE 64 プログラムのデバッグ

AMODE 64 プログラムのデバッグには、CEE320nS メッセージと CEEDUMP トレースバックを使用できます。

次のソースコードが与えられた：

```
ID DIVISION.  
PROGRAM-ID. ZERODIVD.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01 WS-TALLY PIC 9 COMP.  
PROCEDURE DIVISION.  
PROCESS-SEG.  
    DISPLAY '* ROUTINE TO GENERATE ABEND! *'  
    MOVE 0 TO WS-TALLY  
    COMPUTE WS-TALLY = WS-TALLY / WS-TALLY.  
    GOBACK.
```

このプログラムを実行するとアベンドが発生し、言語環境が CEE3209S。アベンドが発生すると、言語環境は実際のエンタリー・ポイントに対するコンパイル・ユニット・オフセットとエンタリー・オフセットを表示する。AMODE 64 では、このオフセットはプログラム開始点から x'10'。COBOL のコンパイル・リストを見ると、言語環境の abend メッセージに表示されるオフセットが x'10'、ずれているように見える。

AMODE 64 プログラムのエンタリー・ポイントは、x'0'ではなくオフセット x'10' から始まります。アベンドが原因で AMODE 64 プログラムをデバッグするには、言語環境アベンド・メッセージ(CEE320nS)のオフセット・アドレスに x'10'を追加して、コンパイル・リストで失敗したステートメントを特定する必要があります。

AMODE 31 のエラーメッセージとリスト

このプログラムをビルドして AMODE 31 で実行すると、次のようなメッセージが出る：

```
CEE3209S The system detected a fixed-point divide exception (System Completion  
Code=0C9).  
          From compile unit ZERODIVD at entry point ZERODIVD at compile unit offset +000001E0 at  
entry offset +000001E0  
          at address 268001E0.
```

LIST コンパイラー・オプションは以下の出力を生成する。最初の実行命令 (BC) はオフセット x'0'で発生し、プログラム開始からのエンタリー・オフセット x'1E0' はアベンド位置を指している。

000002:	PROGRAM-ID.				
ZERODIVD.					
000000		000002	PROC		
ZERODIVD					
000000 47F0 F014		000002	BC	R15,20(,R15)	# Skip over
constant area					
000004 01C3 C5C5		000002	DC	X'01C3C5C5'	# Eyecatcher:
CEE					
000008 0000 0198		000002	DC	X'00000198'	# Stack
Size					
00000C 0000 0590		000002	DC	X'00000590'	# Offset to
PPA1					
000010 47F0 F001		000002	BC	R15,1(,R15)	# Wrong Entry
Point: cause exception					
000014		000002	L0098:	EQU	
*					
000014 90EC D00C		000002	STM	R14,R12,12(,R13)	# Save GPRs
Used					
000018 41A0 F024		000002	LA	R10,36(,R15)	# Args for boot
strap routine					
00001C 98EF F034		000002	LM	R14,R15,52(,R15)	
#					
000020 47F0 F020		000002	BC	R15,32(,R15)	# Branch to boot
strap routine					
000024		000002	L0100:	EQU	# Boot Strap Info
Block					
000024 0000 0000		000002	DC	X'00000000'	# address of entry
label					
000028 0000 00A2		000002	DC	X'000000A2'	# Storage
allocation size					
00002C 0000 06D6		000002	DC	X'000006D6'	# address of Saved
Option String					
000030 0000 05D0		000002	DC	X'000005D0'	# address of entry
point name					
000034 0000 005C		000002	DC	X'0000005C'	# A(Label
L0099)					
000038 0000 0000		000002	DC	X'00000000'	# address of boot

```

strap routine(IGZXBST4)
etc...
000008:          DISPLAY '* ROUTINE TO GENERATE ABEND! *'
0001C0 D217 D0F8 307C 000008 MVC 248(24,R13),124(R3) #
0001C6 4120 D0F8 000008 LA R2,248(,R13) # _ArgumentList
0001CA 1812 000008 LR R1,R2
0001CC 58F0 3020 000008 L R15,32(,R3) #
0001D0 58C0 D080 000008 L R12,128(,R13) #
0001D4 0DEF 000008 BASR R14,R15 # Call IGZXDSP
0001D6 A768 0000 000008 LHI R6,0x0
0001DA 1846 000008 LR R4,R6
000010:          COMPUTE WS-TALLY = WS-TALLY / WS-TALLY.
0001DC 8C45 0020 000010 SRDL R4:R5,32
0001E0 B997 0046 000010 DLR R4:R5,R6 <== Offset
pointing to the abend
0001E4 4050 90A0 000010 STH R5,160(,R9) #
000011:          GOBACK.

```

AMODE 64 エラーメッセージとリスト

このプログラムをビルドして AMODE 64 で実行すると、次のようなメッセージが出る：

```

CEE3209S The system detected a fixed-point divide exception (System Completion
Code=0C9).
      From compile unit ZERO DIVD at entry point ZERO DIVD at compile unit offset
+000000000000001C6 at entry offset
      +000000000000001C6 at address 00000000268001D6.

```

LIST コンパイラー・オプションは以下の出力を生成する。最初の実行命令 (STMG) はオフセット x'10'で発生し、プログラム開始からのエントリーオフセット x'1D6' はアベンド位置を指している。

```

000002:          PROGRAM-ID. ZERO DIVD.
000000 000002 PROC ZERO DIVD
000000 00C3 00C5 000002 DC X'00C300C5' # XPLINK eyecatcher
000004 00C5 00F1 000002 DC X'00C500F1' # XPLINK eyecatcher
000008 0000 0440 000002 DC X'00000440' # Offset to PPA1
00000C 0000 0200 000002 DC X'00000200' # DSA size
000010 EB6E 4610 0024 000002 STMG R6,R14,1552(,R4) # Entry point/
compile unit start
000016 E340 4E00 FF71 000002 LAY R4,-512(,R4) #
00001C 4100 48F0 000002 LA R0,2288(,R4) # Address of
COBDSACB
000020 E300 4868 0024 000002 STG R0,2152(,R4) # Saved in XPLINK
member slot1
000026 E350 4808 0024 000002 STG R5,2056(,R4) #
00002C C080 0000 0186 000002 LARL R8,$CONSTANT_AREA
000032 B904 00A5 000002 LGR R10,R5
000036 D7A3 48F0 48F0 000002 XC 2288(164,R4),2288(R4) #
00003C E54C 48F0 0009 000002 MVHI 2288(,R4),X'0009' #
000042 E3A0 4928 0024 000002 STG R10,2344(,R4) #
000048 9181 A0A4 000002 TM 164(,R10),X'81' # IPCB_Status
00004C A714 0013 000002 JO L0003
000050 B904 001A 000002 LGR R1,R10
000054 4120 48F0 000002 LA R2,2288(,R4) # COBDSACB
000058 E350 A118 0004 000002 LG R5,280(,R10) #
00005E E360 A120 0004 000002 LG R6,288(,R10) #
etc...
000008:          DISPLAY '* ROUTINE TO GENERATE ABEND! *'
00019A A719 0001 000008 LGHI R1,0x1
00019E E320 80F4 0004 000008 LG R2,244(,R8) # +244
0001A4 4130 80B5 000008 LA R3,181(,R8) #
0001A8 E548 4898 000B 000008 MVGHI 2200(,R4),X'000B' #
0001AE E548 48A0 001E 000008 MVGHI 2208(,R4),X'001E' #
0001B4 E548 48A8 0000 000008 MVGHI 2216(,R4),X'0000' #
0001BA E350 A148 0004 000008 LG R5,328(,R10) #
0001C0 E360 A150 0004 000008 LG R6,336(,R10) #
0001C6 0D76 000008 BASR R7,R6 # Call IGZQDSP
<== The CEE3209S abend message
points to offset x'1C6',
but this is not the
abending statement
0001C8 0700 000008
NOP
000010:          COMPUTE WS-TALLY = WS-TALLY / WS-TALLY.

```

```

0001CA A7E8 0000      000010      LHI    R14,0x0
0001CE B904 00CE      000010      LGR    R12,R14
0001D2 8CCD 0020      000010      SRDL   R12:R13,32
0001D6 B997 00CE      000010      DLR    R12:R13,R14
<== The abending statement

      is actually at offset
      x'1D6'.

0001DA 40D0 90A0      000010      STH    R13,160(,R9)      #
000011:      GOBACK.

```

CEEDUMP トレースバックによる複雑なシナリオ

COBOL プログラムで代替エントリー・ポイントが指定されていたり、代替エントリー・ポイントを持つモジュールに複数のプログラムが静的にリンクされているような複雑なシナリオでは、アベンド中に「エントリー・オフセット」番号が「コンパイル・ユニット・オフセット」と一致しなくなる可能性がある。

例えば、MAIN5 プログラムが SUB5 サブプログラムと静的にリンクされており、MAIN5 が SUB5 サブプログラムへの代替エントリーポイントである SUB5ALT を呼び出した場合、アベンド時に次のような CEE3209S とトレースバックが表示される可能性がある。

```

CEE3209S The system detected a fixed-point divide exception (System Completion
Code=0C9).
      From compile unit SUB5 at entry point SUB5ALT at statement 16 at compile unit offset
+000003B4
      at entry offset +000001C6 at address 26800BA4.

```

Traceback:

DSA	Entry	E Offset	Statement	Load Mod	Program Unit	Service	Status
1	CEEHDSP	+00004AE2		CEEPLPKA	CEEHDSP	HLE77D0	Call
2	SUB5ALT	+000001C6	16	MAIN5	SUB5		Exception
3	MAIN5	+000001E4	13	MAIN5	MAIN5		Call

DSA	DSA Addr	E Addr	PU Addr	PU Offset	Comp Date	Compile
1	26D30360	268CADD8	268CADD8	+00004AE2	20210317	
2	26D301C8	268009DE	268007F0	+000003B4	20240813	COB0LV5+ EBCDIC
3	26D30030	26800000	26800000	+000001E4	20240813	COB0LV5+ EBCDIC HFP

注：トレースバック情報に表示されるオフセットは、CEE3209S メッセージのオフセットと一致します。したがって、AMODE 64 COBOL プログラムのトレースバック情報を読むには、オフセット・アドレスに x'10'を追加して、コンパイル・リストで失敗したステートメントを決定しなければならない。

用語の要約

コンパイル単位 (compile unit)

最初の実行命令から始まり、最後の実行命令まで続く実行プログラム単位。

「コンパイルユニット」には、C® プログラムや AMODE64 COBOL プログラムのように、最初の実行命令の前に存在する可能性のある非実行命令 (DC) は含まれない。プログラム・ユニットと同義。

コンパイルユニットアドレス

コンパイルユニットにおける最初の実行可能命令のアドレス。

CEEDUMP トレースバックで見つかったプログラム・ユニット・アドレス (PU Addr) と同義。

コンパイル単位オフセット

コンパイルユニットの最初の実行命令のアドレスからアベンドのアドレスまでのオフセット。

CEEDUMP トレースバックで見つかったプログラム・ユニット・オフセット (PU Offset) と同義です。

入り口点

あるルーチンを実行するためにそのルーチンに入ったときに実行される最初の命令のアドレスまたはラベル。

ロード・モジュールやプログラム・オブジェクトの中では、ロード・モジュールが呼び出されたときにコントロールが渡される場所である。

エントリー・ポイントは、プログラムのメイン・エントリー・ポイント（最初に実行可能な命令）であることもあれば、プログラムが呼び出されたときに制御が渡される、プログラム内で定義された代替エントリー・ポイントであることもある。

エントリーオフセット

プログラム起動時に制御が渡された最初の実行命令のアドレスから、アベンドのアドレスまでのオフセット。

プログラム・ユニット

コンパイルユニットの同義語。

プログラム単位アドレス

コンパイル・ユニット・アドレスの同義語。CEEDUMP のトレースバックでは「PU Addr」と略される。

プログラム・ユニット・オフセット

コンパイル・ユニット・オフセットの同義語。CEEDUMP のトレースバックでは「PU Offset」と略記される。

-

デバッガーの使用

z/OS デバッガを使用して、Enterprise COBOL プログラムをデバッグすることができます。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。

このタスクについて

リモート・デバッグの場合、Eclipse ベースのデバッグ・インターフェースおよび IBM Z Open Debug を、z/OS または z/OS ユニックス の下で実行されている z/OS デバッガ ホストと一緒に使用することができます。

TEST サブオプション NOSOURCE を指定すると、より小さいオブジェクト・プログラムをディスクに保存することができます。ロードされるサイズは変わりません。デバッグ情報は、z/OS デバッガなどのデバッガーまたは LE (CEEDUMP の場合) によって要求されない限り、ロードされません。NOSOURCE サブオプションを使用すると、z/OS デバッガ ソース・ウィンドウにソースを表示できません。

OPTIMIZE(0)、NOSTGOPT、および TEST コンパイラー・オプションを指定して、最も多くのデバッグ機能を取得します。

ゼロ以外の OPTIMIZE レベル、NOSTGOPT および TEST(EJPD) コンパイラー・オプションを指定すると、デバッグ機能にいくつかの制限がありますが、パフォーマンスが向上します。

最高のパフォーマンスを得ながら、引き続き z/OS デバッガを使用できるようにする (デバッグ機能は一部制限されます) には、ゼロ以外の OPTIMIZE レベル、STGOPT コンパイラー・オプション、および TEST(NOEJPD) コンパイラー・オプションを指定します。

最適パフォーマンスと対比した最大デバッグ機能を得るために使用するコンパイラー・オプションの詳細については、TEST コンパイラー・オプションに関する関連参照を参照してください。

関連タスク

IBM z/OS デバッガ・ユーザー・ガイド (プログラムをデバッグするための準備)

関連参照

[448 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。

このタスクについて

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 62. リストを取得するためのコンパイラー・オプションの使用			
使用	リスト	内容	コンパイラー・オプション
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。	短縮リスト	- 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 - compilation¹に関する診断メッセージ	NOSOURCE, NOXREF, NOVBREF, NOMAP, NOOFFSET, 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	442 ページの『SOURCE』
ストレージ・ダンプ内で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 項目のマップ	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 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) 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)	418 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。プロシージャが参照されているコンテキスト (例えば、ステートメントが 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データ・セットまたはファイルを判別する。	名前の相互参照リスト; ソートされた相互参照リスト COPY/BASIS 声明とコピーブックデータセットまたはファイル	データ名、プロシージャ名、プログラム名。これらの名前への参照。 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データ・セットまたはファイル 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名が定義された行番号を提供します。	463 ページの『外部参照』 ^{2,3}

表 62. リストを取得するためのコンパイラー・オプションの使用 (続き)

使用	リスト	内容	コンパイラー・オプション
プログラム内で障害のあるステートメントを見つける。または、プログラムの実行中に移動されるデータ項目のストレージ内で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ー・コード ³	生成されたコード	414 ページの『LIST』 ^{2,4}
PROCEDURE DIVISION セクションを移動または追加した後でも有効な論理パスがまだあるか検査する。	圧縮 PROCEDURE DIVISION リスト	圧縮されたステートメントリスト、グローバル・テーブル、WORKING-STORAGE 情報、およびリテラル	427 ページの『OFFSET』
特定のステートメントのインスタンスを見つける。	アルファベット順のステートメント	使用されたそれぞれのステートメント、各ステートメントが使用された回数、各ステートメントが使用された行番号	458 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管理するオプション (例えば FLAG) をオフにしてください。また、メッセージを選択的に抑制するには、MSGEXIT のサブオプション EXIT コンパイラオプション。

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. プロシージャ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. 生成されたオブジェクトコードのリストは、選択的に配置することで制御できます。*CONTROL LIST そして *CONTROL NOLIST (または同等に、*CBL LIST そして *CBL NOLIST) ステートメントをソースに含めます。注意: *CONTROL 声明は PROCESS (または CBL) 声明。

出力が生成されるのは、以下の場合です。

- COMPILE オプションを指定している (または NOCOMPILE(x) オプションが有効であり、エラー・レベル x 以上が発生していない)。
- OFFSET オプションを指定していない。OFFSET と LIST は互いに排他的なオプションで、OFFSET の方が優先されます。

[502 ページの『例: 短縮リスト』](#)

[504 ページの『例: SOURCE および NUMBER 出力』](#)

[505 ページの『例: MAP 出力』](#)

[508 ページの『例: 組み込みマップ要約』](#)

[511 ページの『例: ネストされたプログラム・マップ』](#)

[531 ページの『例: XREF 出力: データ名相互参照』](#)

[533 ページの『例: XREF 出力: プログラム名相互参照』](#)

[533 ページの『例: XREF 出力: COPY/BASIS 相互参照』](#)

[535 ページの『例: XREF 出力: 組み込み相互参照』](#)

[536 ページの『例: OFFSET コンパイラー出力』](#)

[536 ページの『例: VBREF コンパイラー出力』](#)

[537 ページの『例: 条件付きコンパイル出力』](#)

関連タスク

[307 ページの『コンパイラー・メッセージのリストの生成』](#)

[512 ページの『LIST 出力の読み取り』](#)

言語環境デバッグガイド (COBOL プログラムのデバッグ)

関連参照

[307 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。

```
Invocation parameters:      (1)
  OPTFILE
PROCESS(CBL) statements:   (2)
  CBL NODECK
  CBL NOADV,NODYN,ONAME,ONUMBER,QUOTE,SEQ,DUMP
  CBL NOSOURCE, NOXREF, NOVBREF, NOMAP, NOOFFSET, NOLIST
Options from SYSOPTF:      (3)
  C,NODU,FLAG(I),X,MAP,NOLIST,RENT,OPT(1),SSR
  TEST TRUNC(OPT)
Options in effect:         (4)
  NOADATA
  NOADV
  AFP(VOLATILE)
  QUOTE
  ARCH(8)
  ARITH(COMPAT)
  NOAWO
  NOBLOCK0
  BUFSIZE(4096)
  NOCICS
  CODEPAGE(1140)
  COMPILE
  NOCOPYRIGHT
  NOCURRENCY
  DATA(31)
  DBCS
  NODECK
  NODIAGTRUNC
  DISPSIGN(COMPAT)
  NODLL
  DUMP
  NODYNAM
  NOEXIT
  NOEXPORTALL
  NOFASTSRT
  FLAG(I)
  NOFLAGSTD
  HGPR(PRESERVE)
  INTDATE(ANSI)
  LANGUAGE(EN)
  LINECOUNT(60)
  NOLIST
  NOMAP
  MAXPCF(100000)
  NOMDECK
  NONAME
  NSYMBOL(NATIONAL)
  NONUMBER
  NUMPROC(NOPFD)
  OBJECT
  NOOFFSET
  OPTIMIZE(1)
  OUTDD(SYSOUT)
  PGMNAME(COMPAT)
  QUALIFY(COMPAT)
  RENT
  RMODE(AUTO)
  NORULES
  NOSERVICE
  SEQUENCE
  NOSOURCE
  SPACE(1)
  NOSQL
  SQLCCSID
```

```

NOSQLIMS
  SSRANGE(NOZLEN)
NOSTGOPT
SUPPRESS
NOTERM
  TEST(NOJPD,SOURCE)
NOTHREAD
  TRUNC(OPT)
NOVBREF
  VLR(COMPAT)
  VSAMOPENFS(COMPAT)
NOWORD
  XMLPARSE(XMLSS)
NOXREF
NOZONECHECK
  ZONEDATA(PFD)
ZWB

```

LineID	Message code	Message text (5)																
	IGYSC3002-I	A severe error was found in the program. The "OPTIMIZE" and the "STGOPT" compiler options were cancelled.																
160	IGYDS1089-S	"ASSIGNN" was invalid. Scanning was resumed at the next area "A" item, level-number, or the start of the next clause.																
192	IGYDS1050-E	File "LOCATION-FILE" contained no data record descriptions. The file definition was discarded.																
192	IGYGR1207-S	The "ASSIGN" clause was missing or invalid in the "SELECT" entry for file "LOCATION-FILE". The file definition was discarded.																
888 file	IGYPS2052-S	An error was found in the definition of file "LOCATION-FILE". The reference to this file was discarded. Same message on line: 979																
1000	IGYPS2121-S	"WS-DATE" was not defined as a data-name. The statement was discarded. Same message on line: 1001																
1004 statement	IGYPS2053-S	An error was found in the definition of file "LOCATION-FILE". This input/output statement was discarded. Same message on line: 1016																
1015	IGYPS2121-S	"LOC-CODE" was not defined as a data-name. The statement was discarded.																
1212	IGYPS2121-S	"WS-NUMERIC-DATE" was not defined as a data-name. The statement was discarded.																
1655 results	IGYPG3113-W	Truncation of high-order digit positions may occur due to precision of intermediate results exceeding 30 digits.																
<table border="1"> <thead> <tr> <th>Messages</th> <th>Total</th> <th>Informational</th> <th>Warning</th> <th>Error</th> <th>Severe</th> <th>Terminating</th> <th>(6)</th> </tr> </thead> <tbody> <tr> <td>Printed:</td> <td>13</td> <td>1</td> <td>1</td> <td>1</td> <td>10</td> <td></td> <td></td> </tr> </tbody> </table>			Messages	Total	Informational	Warning	Error	Severe	Terminating	(6)	Printed:	13	1	1	1	10		
Messages	Total	Informational	Warning	Error	Severe	Terminating	(6)											
Printed:	13	1	1	1	10													
* Statistics for COBOL program IGYTCARA: (7)																		
* Source records = 1755																		
* Data Division statements = 295																		
* Procedure Division statements = 479																		
* Generated COBOL statements = 0																		
* Program complexity factor = 486																		
End of compilation 1, program IGYTCARA, highest severity 12. (8)																		
Return code 12																		

(1)

コンパイラ呼び出し時にコンパイラに渡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、オプションが渡されていない場合には表示されません。

OPTFILE

SYSOPTF データ・セットからオプションを要求します。

(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。

NOOFFSET

PROCEDURE DIVISION の圧縮されたリストを抑制します。

NOMAP

DATA DIVISION に定義されている項目のマップ・レポートを抑制します。

(3)

SYSOPTF データ・セットから取得したオプション (OPTFILE コンパイラー・オプションが指定されたため)。

NOLIST

ソース・コードのアセンブラー言語拡張を抑制します。

TEST

このプログラムは、デバッグおよび問題判別ツール (デバッグ・ツールや Fault Analyzer など) で使用するため、および CEEDUMP にリストされているローカル変数を取得するためにコンパイルされました。

(4)

このコンパイルの開始時のオプションの状況。

(5)

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、リストの先頭に示されます。

(6)

このプログラムの診断メッセージのカウンで、重大度レベルによってグループ化されたもの。

(7)

プログラム IGYTCARA のプログラム統計。

(8)

コンパイル単位 of プログラム統計。バッチ・コンパイルを実行する場合、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。リスト内の注釈番号は、番号が付いた後続の説明と対応しています。

(1)									
LineID	PL	SL	-----*A-1-B-----2-----3-----4-----5-----6-----7- -----8 Map and Cross Reference						
(2)	(3)	(4)							
000870			/*****						
000871			*** DO MAIN LOGIC ***						
000872			***						
000873			*** Initialization. Read and process update transactions until **						
000874			*** EOF. Close files and stop run. ***						
000875			*****						
000876			procedure division.						
000877			000-do-main-logic.						
000878			display "PROGRAM IGYTCARA - Beginning".						
000879			perform 050-create-vsam-main-file.						930
000880			perform 100-initialize-paragraph.						982
000881			read update-transaction-file into ws-transaction-record						203 338
000882			at end						
000883	1	IA4390	set transaction-eof to true						253
000884			end-read.						
000885		IA4410	perform until transaction-eof						253
000886	1		perform 200-edit-update-transaction						1050
000887	1	IA4430	if no-errors						372
000888	2		perform 300-update-commuter-record						1159
000889	1		else						
000890	2		perform 400-print-transaction-errors						1312
000891	1		end-if						
000892	1		perform 410-re-initialize-fields						1373
000893	1	IA4480	read update-transaction-file into ws-transaction-record						203 338
000894	1		at end						
000895	2	IA4500	set transaction-eof to true						253
000896	1	IA4510	end-read						
000897		IA4520	end-perform.						
000898			close commuter-file update-transaction-file location-file						180 203 192
000899			print-file.						216
000900									
000901			*-----*						
000902			* File status checked after I/O operation. *						
000903			*-----*						
000904									
000905		IA4600	if not i-o-okay						241
000906	1		display "000-close"						
000907	1		move 0000 to comp-code						230
000908	1	IA4620	perform 500-vsam-error						1386
000909	1		perform 900-abnormal-termination						1432
000910		IA4630	end-if.						
000911			*****						
000912			* Paragraphs 1100 and 1200 illustrates the intrinsic *						
000913			* function computations. *						
000914			*****						
000915			perform 1100-print-i-f-headings.						1441
000916			perform 1200-print-i-f-data.						1481
000917			display " ".						
000918			display " ".						

```
000919      display "PROGRAM IGYTCARA - Normal end".
000920      stop run.
```

- (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。

例 1

以下の例は、SOURCE (DEC) が有効である場合の MAP 出力を示しています。

Data Division Map

(1) Data Definition Attribute codes (rightmost column) have the following meanings:

BL= Fixed byte-length UTF-8
D = Object of OCCURS DEPENDING
DL= Dynamic length
E = EXTERNAL
F = Fixed-length file
FB= Fixed-length blocked file
G = GLOBAL
O = Has OCCURS clause
OG= Group has own length definition
R = REDEFINES
S = Spanned file
TI= Associated type ID
U = Undefined format file
V = Variable-length file
VB= Variable-length blocked file
X = Unallocated

(2) Source LineID	(3) Hierarchy and Data Name	(4)	(5) Base Locator	(6) Displacement Structure	(7) Asmblr Data Definition	(8) Data Type	(9) Data Def Attributes
4	PROGRAM-ID IGYTCARA-----						*
58	FD COMMUTER-FILE		BLF=00001			VSAM	F
60	1 COMMUTER-RECORD		BLF=00001		DS 0CL80	Group	
61	2 COMMUTER-KEY		BLF=00001	000000000	DS 16C	Display	
62	2 FILLER		BLF=00001	000000016	DS 64C	Display	
64	FD COMMUTER-FILE-MST		BLF=00002			VSAM	F
66	1 COMMUTER-RECORD-MST		BLF=00002		DS 0CL80	Group	
67	2 COMMUTER-KEY-MST		BLF=00002	000000000	DS 16C	Display	
68	2 FILLER		BLF=00002	000000016	DS 64C	Display	
140	1 STATUS-AREA				DS 0CL8	Group	
141	2 COMMUTER-FILE-STATUS			000000000	DS 2C	Display	
142	88 I-O-OKAY						
143	2 COMMUTER-VSAM-STATUS			000000002	DS 0CL6	Group	
144	3 VSAM-R15-RETURN-CODE			000000002	DS 2C	Binary	
145	3 VSAM-FUNCTION-CODE			000000004	DS 2C	Binary	
146	3 VSAM-FEEDBACK-CODE			000000006	DS 2C	Binary	
148	77 UPDATE-FILE-STATUS				DS 2C	Display	
149	77 LOCCODE-FILE-STATUS				DS 2C	Display	
150	77 UPDPRINT-FILE-STATUS				DS 2C	Display	
152	1 FLAGS				DS 0CL3	Group	
153	2 TRANSACTION-EOF-FLAG			000000000	DS 1C	Display	
154	88 TRANSACTION-EOF						
155	2 LOCATION-EOF-FLAG			000000001	DS 1C	Display	
156	88 LOCATION-EOF						
157	2 TRANSACTION-MATCH-FLAG			000000002	DS 1C	Display	
158	88 TRANSACTION-MATCH						
159	88 TRANSACTION-MATCH-OFF						
216	1 WS-COMMUTER-RECORD		BLX=00001		DS 0CL81	Group	E
217	2 WS-COMMUTER-KEY		BLX=00001	000000000	DS 0CL16	Group	E
218	3 WS-COMMUTER-GENERIC-KEY		BLX=00001	000000000	DS 0CL5	Group	E
219	4 COMMUTER-SHIFT		BLX=00001	000000000	DS 1C	Display	E
220	4 COMMUTER-HOME-CODE		BLX=00001	000000001	DS 2C	Display	E
221	4 COMMUTER-WORK-CODE		BLX=00001	000000003	DS 2C	Display	E
222	3 COMMUTER-NAME		BLX=00001	000000005	DS 9C	Display	E
223	3 COMMUTER-INITIALS		BLX=00001	000000014	DS 2C	Display	E
224	2 COMMUTER-ADDRESS		BLX=00001	000000016	DS 18C	Display	E
225	2 COMMUTER-CITY		BLX=00001	000000034	DS 13C	Display	E
226	2 COMMUTER-STATE		BLX=00001	000000047	DS 2C	Display	E
227	2 COMMUTER-ZIPCODE		BLX=00001	000000049	DS 3P	Packed-Dec	E
396	1 DETAIL1-LINE		BLL=00001		DS 0CL121	Group	

397	2	FILLER.	BLL=00001	000000000	DS 2C	Display	
398	2	PRINT-TRANSACTION-CODE.	BLL=00001	000000002	DS 1C	Display	
399	2	FILLER.	BLL=00001	000000003	DS 4C	Display	
400	2	PRINT-RECORD-TYPE	BLL=00001	000000007	DS 3C	Display	
401	2	FILLER.	BLL=00001	000000010	DS 3C	Display	
402	2	PRINT-SHIFT	BLL=00001	000000013	DS 1C	Display	
403	2	FILLER.	BLL=00001	000000014	DS 1C	Display	
404	2	PRINT-HOME-CODE	BLL=00001	000000015	DS 2C	Display	
405	2	FILLER.	BLL=00001	000000017	DS 1C	Display	
406	2	PRINT-WORK-CODE	BLL=00001	000000018	DS 2C	Display	
407	2	FILLER.	BLL=00001	000000020	DS 2C	Display	
408	2	PRINT-NAME.	BLL=00001	000000022	DS 9C	Display	
454	1	DETAILX-LINE.	BLL=XXXXX		DS 0CL121	Group	X
455	2	FILLER.	BLL=XXXXX		DS 36C	Display	X
456	2	PRINT-CITY.	BLL=XXXXX		DS 13C	Display	X
457	2	FILLER.	BLL=XXXXX		DS 3C	Display	X
458	2	PRINT-STATE	BLL=XXXXX		DS 2C	Display	X
459	2	FILLER.	BLL=XXXXX		DS 1C	Display	X
460	2	PRINT-ZIPCODE	BLL=XXXXX		DS 5C	Display	X
461	2	FILLER.	BLL=XXXXX		DS 1C	Display	X
462	2	PRINT-WORK-PHONE.	BLL=XXXXX		DS 14C	Display	X
463	2	FILLER.	BLL=XXXXX		DS 1C	Display	X
464	2	PRINT-WORK-JUNCTION	BLL=XXXXX		DS 25C	Display	X
465	2	FILLER.	BLL=XXXXX		DS 20C	Display	X (10)
467	1	DETAIL2-LINE.	BLL=00002		DS 0CL121	Group	
468	2	FILLER.	BLL=00002	000000000	DS 36C	Display	
469	2	PRINT-CITY.	BLL=00002	000000036	DS 13C	Display	
470	2	FILLER.	BLL=00002	000000049	DS 3C	Display	
471	2	PRINT-STATE	BLL=00002	000000052	DS 2C	Display	
472	2	FILLER.	BLL=00002	000000054	DS 1C	Display	
473	2	PRINT-ZIPCODE	BLL=00002	000000055	DS 5C	Display	
474	2	FILLER.	BLL=00002	000000060	DS 1C	Display	
475	2	PRINT-WORK-PHONE.	BLL=00002	000000061	DS 14C	Display	
476	2	FILLER.	BLL=00002	000000075	DS 1C	Display	
477	2	PRINT-WORK-JUNCTION	BLL=00002	000000076	DS 25C	Display	
478	2	FILLER.	BLL=00002	000000101	DS 20C	Display	

例 2

以下の例は、SOURCE (HEX) が有効である場合の MAP 出力を示しています。

Data Division Map								
(1)								
Data Definition Attribute codes (rightmost column) have the following meanings:								
BL= Fixed byte-length UTF-8								
D = Object of OCCURS DEPENDING								
DL= Dynamic length								
E = EXTERNAL								
F = Fixed-length file								
FB= Fixed-length blocked file								
G = GLOBAL								
O = Has OCCURS clause								
OG= Group has own length definition								
R = REDEFINES								
S = Spanned file								
TI= Associated type ID								
U = Undefined format file								
V = Variable-length file								
VB= Variable-length blocked file								
X = Unallocated								
(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	
Source	Hierarchy and		Base	Displacement	Asmblr Data	Data Type	Data Def	
LineID	Data Name		Locator	Structure	Definition		Attributes	
4	PROGRAM-ID	MSGXIT-----						
*								
F	1	EC.		000000000	DS 2P	Packed-Dec		
10	1	JUNK.		000000000	DS 2C	Display		

例 3

次の例は、SOURCE (DEC) が有効で、プログラムに型定義と型インスタンスが含まれている場合の MAP 出力を示している。

Data Division Map								
(1)								
Data Definition Attribute codes (rightmost column) have the following meanings:								
BL= Fixed byte-length UTF-8								
D = Object of OCCURS DEPENDING								
DL= Dynamic length								
E = EXTERNAL								
F = Fixed-length file								
FB= Fixed-length blocked file								
G = GLOBAL								
O = Has OCCURS clause								
OG= Group has own length definition								
R = REDEFINES								
S = Spanned file								
TI= Associated type ID								
U = Undefined format file								
V = Variable-length file								
VB= Variable-length blocked file								
X = Unallocated								

(2) Source LineID	(3) Hierarchy and Type Definition	(5) Type ID	(6) Displacement Structure	(7) Asmblr Data Definition	(8) Data Type	(9) Data Def Attributes
2	PROGRAM-ID F00					
* 6	1 MYTYPE1-T	TI=00001	00000000	DS 0CL30	Group	
7	2 DATA1		00000000	DS 10C	Display	
8	2 DATA2		0000000A	DS 20C	Display	
9	1 MYTYPE2-T	TI=00002	00000000	DS 0CL370	Grp-VarLen	
10	2 DATA3		00000000	DS 30C	Display	
11	2 G1		00000001E	DS 0CL30	Group	0, TI=00001
11	3 DATA1		00000001E	DS 10C	Display	
11	3 DATA2		000000028	DS 20C	Display	
12	2 DATA4	BLV=XXXXX		DS 40C	Display	
(2) Source LineID	(3) Hierarchy and Data Name	(5) Base Locator	(6) Displacement Structure	(7) Asmblr Data Definition	(8) Data Type	(9) Data Def Attributes
2	PROGRAM-ID F00					
* 5	1 X1		00000000	DS 4C	Binary	D
13	1 Y1		00000000	DS 4C	Binary	
14	1 G2		00000000	DS 0CL420	Grp-VarLen	
15	2 G3		00000000	DS 0CL370	Grp-VarLen	TI=00002
15	3 DATA3		00000000	DS 30C	Display	
15	3 G1		00000001E	DS 0CL30	Group	0
15	4 DATA1		00000001E	DS 10C	Display	
15	4 DATA2		000000028	DS 20C	Display	
15	3 DATA4	BLV=00001		DS 40C	Display	
16	2 DATA5	BLV=00002		DS 50C	Display	
17	1 G4		00000000	DS 0CL30	Group	TI=00001
17	2 DATA1		00000000	DS 10C	Display	
17	2 DATA2		0000000A	DS 20C	Display	
18	1 G5		00000000	DS 0CL370	Grp-VarLen	TI=00002
18	2 DATA3		00000000	DS 30C	Display	
18	2 G1		00000001E	DS 0CL30	Group	0
18	3 DATA1		00000001E	DS 10C	Display	
18	3 DATA2		000000028	DS 20C	Display	
18	2 DATA4	BLV=00003		DS 40C	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。定義は上部に説明されています DATA DIVISION 地図。
- (9) データ定義属性コード。定義は上部に説明されています DATA DIVISION 地図。
- (10) DETAILX-LINE は言及されていなかった PROCEDURE DIVISION。STGOPT が指定されていたため、DETAILX-LINE は削除され、その結果、ベース・ロケータは XXXXX に設定されました。
- (11) ソース・モジュール内でソース順に使用されるタイプ名。

508 ページの『例: 組み込みマップ要約』

511 ページの『例: ネストされたプログラム・マップ』

関連参照

509 ページの『MAP 出力で使用される用語』

510 ページの『LIST および MAP 出力で使用される記号』

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. IGYTCARA.
.
.
000054 Data division.
000055 File section.
000056
000058 FD COMMUTER-FILE
000059 record 80 characters.
.
.
000060 01 commuter-record.
000061 05 commuter-key PIC x(16).
000062 05 filler PIC x(64).
(1) (2) (3)
BLF=00001 0CL80
BLF=00001,000000000 16C
BLF=00001,000000016 64C
.
.
000105 Working-storage section.
000106 01 Working-storage-for-IGYCARA pic x. 1C
000107
000108 77 comp-code pic S9999 comp. 2C
000109 77 ws-type pic x(3) value spaces. 3C
000135 01 i-f-status-area. 0CL2
000136 05 i-f-file-status pic x(2). 000000000 2C
000137 88 i-o-successful value zeroes.
000138
000139
000140 01 status-area. 0CL8
000141 05 commuter-file-status pic x(2). 000000000 2C
000142 88 i-o-okay value zeroes.
000143 05 commuter-vsam-status. 000000002 0CL6
000144 10 vsam-r15-return-code pic 9(2) comp. 000000002 2C
000145 10 vsam-function-code pic 9(1) comp. 000000004 2C
000146 10 vsam-feedback-code pic 9(3) comp. 000000006 2C
000147
000148 77 update-file-status pic xx. 2C
000149 77 loccode-file-status pic xx. 2C
000150 77 updprint-file-status pic xx. 2C
000151
000216 01 ws-commuter-record EXTERNAL. BLX=00001 0CL81
000217 05 ws-commuter-key. BLX=00001,000000000 0CL16
000218 10 ws-commuter-generic-key. BLX=00001,000000000 0CL5
000219 15 commuter-shift pic x. BLX=00001,000000000 1C
000220 15 commuter-home-code pic xx. BLX=00001,000000001 2C
000221 15 commuter-work-code pic xx. BLX=00001,000000003 2C
000222 10 commuter-name pic x(9). BLX=00001,000000005 9C
000223 10 commuter-initials pic xx. BLX=00001,000000014 2C
000224 05 commuter-address pic x(18). BLX=00001,000000016 18C
000225 05 commuter-city pic x(13). BLX=00001,000000034 13C
000226 05 commuter-state pic xx. BLX=00001,000000047 2C
000227 05 commuter-zipcode pic 9(5) comp-3. BLX=00001,000000049 3P
.
.
000395 Linkage Section.
000396 01 detail1-line. BLL=00001 0CL121
000397 05 filler pic xx. BLL=00001,000000000 2C
000398 05 print-transaction-code pic x. BLL=00001,000000002 1C
000399 05 filler pic x(4). BLL=00001,000000003 4C
000400 05 print-record-type pic x(3). BLL=00001,000000007 3C
000401 05 filler pic xxx. BLL=00001,000000010 3C
000402 05 print-shift pic x. BLL=00001,000000013 1C
000403 05 filler pic x. BLL=00001,000000014 1C
000404 05 print-home-code pic xx. BLL=00001,000000015 2C
000405 05 filler pic x. BLL=00001,000000017 1C
000406 05 print-work-code pic xx. BLL=00001,000000018 2C
000407 05 filler pic xx. BLL=00001,000000020 2C
000408 05 print-name pic x(9). BLL=00001,000000022 9C
000409 05 filler pic xx. BLL=00001,000000031 2C
000410 05 print-initials pic xx. BLL=00001,000000033 2C
.
.
000487 procedure division.
000488 000-do-main-logic.
000489 display "PROGRAM IGYTCARA - Beginning".
```

(1)

このデータ項目に使用されるベース・ロケータ。

(2)

包含構造の先頭からの 10 進変位。MAP (DEC) オプションが有効であることを示します。指定した場合 MAP (HEX) オプションまたは MAP サブオプションがない場合、16 進数の変位が表示されます。

(3)

データが定義されている方法を示す、疑似アセンブラー・コード。

関連参照

510 ページの『LIST および MAP 出力で使用される記号』

MAP 出力で使用される用語

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を説明しています。

表 63. MAP 出力で使用される用語		
用語	定義	説明
ALPHABETIC	DS <i>n</i> C	アルファベットデータ項目 (PICTURE A)
ALPHA-EDIT	DS <i>n</i> C	英字編集データ項目
AN-EDIT	DS <i>n</i> C	英数字編集データ項目
BINARY	DS 1H ² 、1F ² 、2F ² 、2C、4C、または 8C	バイナリー・データ項目 (USAGE BINARY、COMPUTATIONAL、または COMPUTATIONAL-5)
COMP-1	DS 4C	単精度内部浮動小数点データ項目 (USAGE COMPUTATIONAL-1)
COMP-2	DS 8C	倍精度内部浮動小数点データ項目 (USAGE COMPUTATIONAL-2)
DBCS	DS <i>n</i> C	DBCS データ項目 (USAGE DISPLAY-1)
DBCS-EDIT	DS <i>n</i> C	DBCS 編集済みデータ項目 (USAGE DISPLAY-1)
DISP-FLOAT	DS <i>n</i> C	浮動小数点データ項目を表示する (USAGE DISPLAY)
表示	DS <i>n</i> C	英数字データ項目 (PICTURE X)
DISP-NUM	DS <i>n</i> C	ゾーン分け 10 進データ項目 (USAGE DISPLAY)
DISP-NUM-EDIT	DS <i>n</i> C	数値編集データ項目 (USAGE DISPLAY)
FD		ファイル定義
FUNCTION-PTR	DS <i>n</i> C	関数ポインター (USAGE FUNCTION-POINTER)
グループ	DS 0CL <i>n</i> ¹	固定長英数字グループ・データ項目
GRP-VARLEN	DS 0CL <i>n</i> ¹	可変長英数字グループ・データ項目
索引	DS <i>n</i> C	指標データ項目 (USAGE INDEX)
INDEX-NAME	DS <i>n</i> C	索引名
NATIONAL	DS <i>n</i> C	カテゴリー国別データ項目 (USAGE NATIONAL)
NAT-EDIT	DS <i>n</i> C	国別編集データ項目 (USAGE NATIONAL)
NAT-FLOAT	DS <i>n</i> C	国別浮動小数点データ項目 (USAGE NATIONAL)
NAT-GROUP	DS 0CL <i>n</i> ¹	ナショナルグループ (GROUP-USAGE NATIONAL)
NAT-GRP-VARLEN	DS 0CL <i>n</i> ¹	全国可変長グループ (GROUP-USAGE NATIONAL)
NAT-NUM	DS <i>n</i> C	国別小数点データ項目 (USAGE NATIONAL)

表 63. MAP 出力で使用される用語 (続き)		
用語	定義	説明
NAT-NUM-EDIT	DS <i>n</i> C	国別数値編集データ項目 (USAGE NATIONAL)
OBJECT-REF	DS <i>n</i> C	オブジェクト参照データ項目 (USAGE OBJECT REFERENCE)
PACKED-DEC	DS <i>n</i> P	内部 10 進データ項目 (USAGE PACKED-DECIMAL または COMPUTATIONAL-3)
POINTER	DS <i>n</i> C	ポインター・データ項目 (USAGE POINTER)
PROCEDURE-PTR	DS <i>n</i> C	プロシージャ・ポインター (USAGE PROCEDURE-POINTER)
SD		ソート・ファイル定義
VSAM、QSAM、LINESEQ		ファイル処理方式
1 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
1. <i>n</i> は、固定長グループのバイト単位のサイズ、および可変長グループのバイト単位の最大サイズです。 2. SYNCHRONIZED 節が表示されると、これらのフィールドが使用されます。		

LIST および MAP 出力で使用される記号

次の表は、LIST または MAP オプションによって作成されるリストで使用される記号を説明しています。

表 64. LIST および MAP 出力で使用される記号	
記号	定義
BLF_ <i>n</i> ¹	ファイルのベース・ロケーター
BLL_ <i>n</i> ¹	ベースロケーター LINKAGE SECTION
BLO_ <i>n</i> ¹	オブジェクト・インスタンス・データのベース・ロケーター
BLT_ <i>n</i> ¹	XML-TEXT および XML-NTEXT のベース・ロケーター
BLV_ <i>n</i> ¹	位置が変更できるデータのベース・ロケーター
BLX_ <i>n</i> ¹	外部データのベース・ロケーター
ODOsv_cell	ODO 保管セル番号
Pfm_cell	PERFORM セル番号
Pfmsv_cell	保管セル番号を実行する
TSN=N	コンパイラにより一時的に作成
VLC_cell	可変長セル (ODO)
VN_cell	PERFORM ステートメントの変数名セル
VNGO_cell	ALTER ステートメントの変数名セル
VNI_cell	変数名初期化

表 64. <i>LIST</i> および <i>MAP</i> 出力で使用される記号 (続き)	
記号	定義
#Calc00000000n	OCCURS DEPENDING ON 節の後に存在するデータのアドレスを計算するコード
#WSVal00000000n	プロシージャーの WORKING-STORAGE 域を初期化するコード
_ArgumentList	プロシージャーに対する出力引数
_ACON	シンボルのアドレス
_BEtempNNN	最適化プログラムにより一時的に作成
_CAA	開始アドレス言語環境 Common Anchor Area
_CACHED_\$STATIC	(このプロシージャーの) 静的域の開始アドレスのコピー
_CONSTANT_AREA+n	Constant Area 内のオフセット
_CRENT	CAA からの (このモジュールの) ストレージのアドレス
_incomingArgumentList	プロシージャーに対する入力パラメーター
_parentDSA	ネストされたプロシージャーの場合、その親のスタックのアドレス
_QCON	シンボルのオフセット
_returnValue	プロシージャーの戻り値
_VTS_n	最適化プログラムにより一時的に作成
1. n は記入項目の番号です。ベースロケータの場合は、XXXXX、削除されたデータ項目を示します STGOPT 処理。	

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 プロシージ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
Program Attribute codes (rightmost column) have the following meanings:
  C = COMMON
  I = INITIAL (1)
  U = PROCEDURE DIVISION USING...
Source Nesting
LineID Level  Program Name from PROGRAM-ID paragraph  Program
Attributes
  2      0      NESTMAIN. . . . . U
120     1 (4) SUBPR01 . . . . . I,C,U
(2)199   2      NESTED1 . . . . . I,C,U
253     1      SUBPR02 . . . . . U
335     2      NESTED2 . . . . . C,U
      (3)
```

- (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

LIST 出力の読み取り

LIST コンパイラ出力の各部分は、プログラムのデバッグに有用である場合があります。

このタスクについて

LIST コンパイラ・オプションは、以下のようないくつかの出力を作成します。

- プログラムの初期化コードのアセンブラー・リスト (プログラム・シグニチャー情報バイト)。このリストからは、以下のようなプログラム特性を検査できます。
 - 有効なコンパイラ・オプション
 - 存在するデータ項目のタイプ
 - PROCEDURE DIVISION で使用されているステートメント
- プログラムのソース・コードのアセンブラー・リスト

異常終了発生時に実行されていた命令のストレージ内のアドレスから、その命令に対応する COBOL ステートメントを見つけることができます。障害のある命令のアドレスが見つかったら、アセンブラー・リストに進み、その命令が生成される対象となったステートメントを調べてください。プログラムのアセンブラー・リストの 3 列目に行番号があります。この行番号を使用して、リストの「Source Output」セクション内の対応する行を調べることで、そのステートメントを見つけることができます。

注: AMODE 64 COBOL プログラムの LIST コンパイラ出力では、エントリー・ポイントは AMODE 31 プログラムのようにオフセット 0 にはない。その代わりに、常にオフセット x'10' にある。エントリーポイントの前の単語は、言語環境に適合した XPLINK データレイアウトで文書化されているように、標準的な XPLINK 情報を含む。AMODE 31 と AMODE 64 におけるコンパイラの LIST レイアウトの違いについては、[522 ページの『例: プログラム初期化コード』](#)。

注: AMODE 64 COBOL プログラムをデバッグするとき、言語環境の abend メッセージ (CEE320nS) のオフセット・アドレスに x'10' を追加する必要がある。詳細については、[495 ページの『AMODE 64 プログラムのデバッグ』](#)を参照してください。

- WORKING-STORAGE に関する情報。この情報は Data Division Map と Initial Heap Storage Map に記載されている。
- の説明は storage の説明は Initial Heap Storage Map にある。ソースの WORKING-STORAGE 領域のシンボルは、に示されているようにマッピングされる。storage Initial Heap Storage Map に示されている。

使用することができます Data Division Map とともに Initial Heap Storage Map セクションで定義されたデータ項目の場所を見つける WORKING-STORAGE。これらのデータ項目は storage. Initial Heap Storage Map は、level-1 の各データのオフセットを示す。storage. Data Division Map セクションは、レベル *n* のデータ項目の、それぞれの level-1 メンバに対するオフセットを示す。この 2 つの情報を扱うことで、. storage.

DATA24 オプションを付けてコンパイルすると、行の下にマッピングされたデータ項目が Below the Line Storage に表示される。同じプロセスに従って、それらの位置を判別できます。

- プログラムで使用する定数およびリテラルに関する情報。の Constant Area プログラム内の定数とリテラル、およびコンパイラによって作成された定数とリテラルに関する情報が含まれます。このセクションには、「Constant Area」内の定数またはリテラルのそれぞれのオフセットが示されます。
- プログラム・プロローグ域 (PPA1、PPA2、PPA3、および PPA4) には、コンパイル済みプログラムの特性に関する情報があります。
- 外部シンボル辞書には、プログラム内で定義または参照されている外部シンボルのリストが含まれています。

- 動的保管域 (DSA) のマップ

DSA (スタック・フレーム と呼ばれる) のマップには、別個のコンパイル済みプロシージャに入るたびに獲得されたストレージの内容に関する情報が入れられます。

LIST 出力を理解するために、アセンブラー言語でプログラミングができる必要はありません。アセンブラー・コードの大部分を伴うコメントは、コードが実行する機能を概念的に理解するのに役立ちます。

[522 ページの『例: プログラム初期化コード』](#)

[524 ページの『例: MD5 シグニチャー』](#)

[525 ページの『例: タイム・スタンプおよびバージョン情報』](#)

[525 ページの『例: コンパイラー・オプションおよびプログラム情報』](#)

[525 ページの『例: ソース・コードから生成されるアセンブラー・コード』](#)

[526 ページの『例: プログラム・プロログ域』](#)

[528 ページの『例: 初期ヒープ・ストレージ・マップ』](#)

[529 ページの『例: 定数域』](#)

[529 ページの『例: ベース・ロケーター・テーブル』](#)

[530 ページの『例: 外部シンボル』](#)

[531 ページの『例: DSA メモリー・マップ \(スタック・ストレージ・マップ\)』](#)

関連参照

[513 ページの『シグニチャー情報バイト』](#)

[505 ページの『例: MAP 出力』](#)

言語環境プログラミングガイド (スタックストレージの概要)

シグニチャー情報バイト

このトピックの表は、LIST コンパイラー・オプションを使用する時に提供されるプログラム初期設定コード・リストの一部であるプログラム・シグニチャー情報を示しています。

表 65. INFO BYTE セクションでのコンパイラー・オプション		
10 進でのオフセット	オプション	値
00	CODEPAGE	EBCDIC コード・ページに設定された CCSID 値
02	ARCH	10
		11
		12
		13
		14
3	OPTIMIZE	0
		1
		2

リストの INFO BYTE セクションには、以下の値も示されます。

- DATA DIVISION ステートメントの数
- PROCEDURE DIVISION ステートメントの数

次の表では、さまざまなシグニチャー・バイトがそれぞれの情報を表しています。

- シグニチャー・バイト 1-5、および 26-35 はコンパイラー・オプションを指しています。
- シグニチャー・バイト 6-7 は、DATA DIVISION 項目を参照する
- シグニチャー・バイト 8 は、ENVIRONMENT DIVISION 項目を参照する

- ・ シグニチャー・バイト 9-25 は、PROCEDURE DIVISION ステートメントおよび項目を参照する

表 66. シグニチャー情報バイト				
10 進 でのオ フセッ ト	シグニチ ャー・バイ ト	ビット	品	
			オン	オフ
04	28	0	SQL	NOSQL
		1	CICS	NOCICS
		2	MDECK	NOMDECK
		3	SQLCCSID	NOSQLCCSID
		4	OPTFILE	NOOPTFILE
		5	XMLPARSE (XMLSS)	XMLPARSE (COMPAT)
		6	BLOCK0	NOBLOCK0
		7	DISPSIGN (SEP)	DISPSIGN (COMPAT)
05	29	0	プログラムで Java ベースの OO 構文を使用	
		1	プログラムが RANDOM 関数を使用する	
		2	プログラムが NATIONAL データ (Unicode) を使用する	
		3	XML PARSE (スキーマ妥当性検査あり)	
		4	STGOPT	NOSTGOPT
		5	AFP (VOLATILE)	AFP (NOVOLATILE)
		6	HGPR (PRESERVE)	HGPR (NOPRESERVE)
		7	NOTEST (DWARF)	NOTEST (DWARF) 以外
06	30	0	QUALIFY (EXTEND)	QUALIFY (COMPAT)
		1	VLR (COMPAT)	VLR (STANDARD)
		2	COPYRIGHT スtringが指定	COPYRIGHT スtringが指定されていない
		3	SERVICE スtringが指定	SERVICE スtringが指定されていない
		4	INVDATA (FNC, x)	INVDATA (FNC, x) 以外 注: ビット 4 と 5 が両方ともオフの場合、NOINVDATA が有効になります。
		5	INVDATA (NOFNC, x)	Not INVDATA (NOFNC, x) 注: ビット 4 と 5 が両方ともオフの場合、NOINVDATA が有効
		6	INVDATA (x, NOCS)	ビット 4 とビット 5 がゼロの場合、INVDATA (x, CS) はゼロである必要があります。
		7	NUMCHECK (ZON (LAX))	NUMCHECK (ZON (STRICT))

表 66. シグニチャー情報バイト (続き)				
10 進 でのオ フセッ ト	シグニチ ャー・バイ ト	ビット	品	
			オン	オフ
07	31	0	NUMCHECK (ZON)	Not NUMCHECK (ZON)
		1	NUMCHECK (PAC)	Not NUMCHECK (PAC)
		2	NUMCHECK (BIN)	Not NUMCHECK (BIN)
				NONUMCHECK は、ビット 0、1、および 2 がオフの場合に有効
		3	NUMCHECK (ABD)	NUMCHECK (MSG) (0、1、または 2 のい ずれかのビットがオンの場合)
		4	PARMCHECK	NOPARMCHECK
		5	PARMCHECK (ABD) (ビット 4 がオンの 場合)	PARMCHECK (MSG) (ビット 4 がオンの 場合)
		6	NUMCHECK (ZON (NOALPHNUM))	NUMCHECK (ZON (ALPHNUM))
		7	TEST NOTEST (SEPARATE (DSNAME))	TEST NOTEST (SEPARATE (NODSNAME))
09	1	0	ADV	NOADV
		1	APOST	QUOTE
		2	DATA (31)	DATA (24)
		3	DECK	NODECK
		4	DUMP	NODUMP
		5	DYNAM	NODYNAM
		6	FASTSRT	NOFASTSRT
		7	SQLIMS	NOSQLIMS
11	2	0	LIB (常時オン)	
		1	LIST	NOLIST
		2	MAP (HEX), MAP (DEC)	NOMAP
		3	NUM	NONUM
		4	OBJECT	NOOBJECT
		5	OFFSET	NOOFFSET
		6	OPT (1), OPT (2)	NOOPT, OPT (0)
		7	OUTDD	NOOUTDD

表 66. シグニチャー情報バイト (続き)				
10 進 でのオフ セット	シグニ チャー・バ イト	ビット	品	
			オン	オフ
10	3	0	NUMPROC (PFD)	NUMPROC (NOPFD)
		1	RENT	NORENT
		2	RESIDENT (常時オン)	
		3	SEQUENCE	NOSEQUENCE
		4	予約済み	
		5	SOURCE	NOSOURCE
		6	Not NOSSRANGE	NOSSRANGE
		7	TERM	NOTERM
11	4	0	TEST	NOTEST
		1	TRUNC (STD)	Not TRUNC (STD)
		2	WORD	NOWORD
		3	VBREF	NOVBREF
		4	XREF	NOXREF
		5	ZWB	NOZWB
		6	NAME	NONAME
		7		NOCMPR2 (常時オフ)
12	5	0	予約済み	
		1	NUMCLS=ALT	NUMCLS=PRIM
		2	DBCS	NODBCS
		3	AWO	NOAWO
		4	TRUNC (BIN)	Not TRUNC (BIN)
		5	ADATA	NOADATA
		6	CURRENCY	NOCURRENCY
		7	コンパイル単位はクラス	コンパイル単位はプログラム
※13	6	0	QSAM ファイル記述子	
		1	VSAM 順次ファイル記述子	
		2	VSAM 索引付きファイル記述子	
		3	VSAM 相対ファイル記述子	
		4	ファイル記述子内の CODE-SET 節 (ASCII ファイル)	
		5	スパン・レコード	
		6	PIC G または PIC N (DBCS データ項目)	
		7	データ記述項目内の OCCURS DEPENDING ON 節	

表 66. シグニチャー情報バイト (続き)

10進でのオフセット	シグニチャー・バイト	ビット	品	
			オン	オフ
14	7	0	データ記述項目内の SYNCHRONIZED 節	
		1	データ記述項目内の JUSTIFIED 節	
		2	USAGE IS POINTER 項目	
		3	複合 OCCURS DEPENDING ON 節	
		4	DATA DIVISION 内の外部浮動小数点項目	
		5	DATA DIVISION 内の内部浮動小数点項目	
		6	行順次ファイル	
		7	USAGE IS PROCEDURE-POINTER または FUNCTION-POINTER 項目	
15	8	0	FILE-CONTROL 段落の FILE STATUS 節	
		1	INPUT-OUTPUT SECTION の I-O-CONTROL 段落の RERUN 節	
		2	SPECIAL-NAMES 段落で定義された UPSI スイッチ	
		3	WSOPT: WORKING-STORAGE SECTION を管理するためにコンパイラーが使用するメソッドを示すビット。詳しくは、「z/OS Language Environment Vendor Interfaces」の『COBOL-specific vendor interfaces』を参照。	
		4	VSAMOPENFS	
		5	1=NUMCHECK(BIN(NOTRUNCBIN)) 0=NUMCHECK(BIN(TRUNCBIN))	
16	9	0	ACCEPT	
		1	ADD	
		2	ALTER	
		3	CALL	
		4	CANCEL	
		6	CLOSE	
17	10	0	COMPUTE	
		2	DELETE	
		4	DISPLAY	
		5	DIVIDE	

表 66. シグニチャー情報バイト (続き)

10進 でのオフ セット	シグニチ ャー・バイ ト	ビット	品	
			オン	オフ
18	11	1	END-PERFORM	
		2	ENTER	
		3	ENTRY	
		4	EXIT	
		5	EXEC	
		6	GO TO	
		7	IF	
19	12	0	INITIALIZE	
		1	INVOKE	
		2	INSPECT	
		3	MERGE	
		4	MOVE	
		5	MULTIPLY	
		6	OPEN	
		7	PERFORM	
20	※13	0	READ	
		2	RELEASE	
		3	RETURN	
		4	REWRITE	
		5	SEARCH	
		7	SET	
21	14	0	SORT	
		1	START	
		2	STOP	
		3	STRING	
		4	SUBTRACT	
		7	UNSTRING	

表 66. シグニチャー情報バイト (続き)

表 66. シグニチャー情報バイト (続き)				
10 進 でのオ フセッ ト	シグニチ ャー・バイ ト	ビット	品	
			オン	オフ
27	20	0 から 7	予約済み	
28	21	0	16 進数リテラル	
		1	変更される GO TO	
		2	I-O ERROR 宣言	
		3	DECIMAL-POINT IS COMMA 節	
		4	DEBUGGING 宣言	
		5	プログラムのセグメンテーション	
		6	OPEN . . . EXTEND	
		7	EXIT PROGRAM	
29	22	0	CALL リテラル	
		1	CALL 識別子	
		2	CALL . . . ON OVERFLOW	
		3	CALL . . . LENGTH OF	
		4	CALL . . . ADDRESS OF	
		5	CLOSE . . . REEL/UNIT	
		6	使用されている指数	
		7	使用されている浮動小数点項目	
30	23	0	COPY	
		1	BASIS	
		2	プログラムの DBCS 名	
		3	プログラムのシフトアウトとシフトイン	
		4	SUPPRESS NOSUPPRESS	
		5	SSRANGE(ZLEN) (バイト 3 のビット 6 がオンの場合)	SSRANGE(NOZLEN) (バイト 3 のビット 6 がオンの場合)
		6	SSRANGE(ABD) (バイト 3 のビット 6 が オンの場合)	SSRANGE(MSG) (バイト 3 のビット 6 が オンの場合)
		7	INLINE NOINLINE	

表 66. シグニチャー情報バイト (続き)

10 進 でのオ フセッ ト	シグニチ ャー・バイ ト	ビット	品	
			オン	オフ
40	24	0	DBCS リテラル	
		1	REPLACE	
		2	参照変更が使用された	
		3	ネストされたプログラム	
		4	INITIAL (IS INITIAL またはコンパイラー・オプション INITIAL のいずれか)	
		5	COMMON	
		6	SELECT . . . OPTIONAL	
		7	EXTERNAL	
41	25 GB	0	GLOBAL	
		1	RECORD IS VARYING	
		2	VOLATILE	
		3	プログラムで UTF-8 データを使用	
		5	組み込み関数が使用された	
		6	z リテラルが検出された	
		7	RECURSIVE	
42	26	0	RMODE (ANY)	Not RMODE (ANY)
		1-3	予約済み	
		4	予約済み	
		5	INTDATE (LILIAN)	INTDATE (ANSI)
		6	TEST NOTEST (SEPARATE)	TEST NOTEST (NOSEPARATE)
		7	予約	
		43	27	0
1	PGMNAME (LONGMIXED)			Not PGMNAME (LONGMIXED)
2	DLL			NODLL
3	EXPORTALL			NOEXPORTALL
4	TEST NOTEST (SOURCE)			TEST NOTEST (NOSOURCE)
5	ARITH (EXTEND)			ARITH (COMPAT)
6	THREAD			NOTHREAD
7	TEST (EJPD)			TEST (NOEJPD)
44	28	0 から 7	ビルド・レベル情報	

表 66. シグニチャー情報バイト (続き)				
10 進 でのオ フセッ ト	シグニチ ャー・バイ ト	ビット	品	
			オン	オフ
52	32	0	LP (64)	LP (32)
		1	予約済み	
		2	SOURCE (HEX)	SOURCE (DEC)
		3	SMARTBIN	NOSMARTBIN
		4	コンパイル単位は関数です。	コンパイル単位は関数ではありません。
		5	予約済み	
		6	プログラムには少なくとも 1 つが含まれています JAVA-SHAREABLE 指令。	プログラムには JAVA-SHAREABLE 指令。
		7	プログラムには、CALL ステートメントを使った Java への呼び出しが 1 つ以上含まれている。	プログラムには、CALL ステートメントを使った Java への呼び出しが含まれていない。
53	33	0	JAVAIOP (JAVA64)	JAVAIOP (NOJAVA64)
		1	LSACHECK	NOLSACHECK
		2 から 7	予約済み	
54	34	0 から 7	予約済み	
55	35	0 から 7	TUNE アーキテクチャー・レベル	

戻りコードの検査: コンパイラーから 4 より大きな戻りコードが戻された場合は、情報バイトで示される一部のステートメントがプログラムから破棄された可能性があることを意味します。

関連参照

414 ページの『LIST』

z/OS Language Environment Vendor Interfaces (COBOL 固有のベンダー・インターフェース)

例: プログラム初期化コード

プログラム初期化コードのリストは、COBOL ソース・プログラムの特性に関する情報を提供します。プログラムの特性を検査するには、プログラム・シグニチャー情報バイトを解釈してください。

Enterprise COBOL 4 以前の初期化コードに含まれる PROGRAM-ID、COMPILED TIME、および COMPILED DATE などの情報は、Enterprise COBOL 5 以降の初期化コードには含まれません。そのため、その情報が依存するプログラムの動作は、Enterprise COBOL 5 および 6 では異なる可能性があります。詳しくは、『正しくないプログラムのエラー動作変更』(Enterprise COBOL 移行ガイド)を参照してください。

次のコード例は、AMODE 31 COBOL プログラム IGYTCARA の LIST 出力の一部を示す。

(1)	(2)	(3)	(4)	(5)
000000		000003	PROC	IGYTCARA
000000	47F0 F014	000003	BC	R15,20(,R15)
000004	01C3 C5C5	000003	DC	X'01C3C5C5'
000008	0000 0978	000003	DC	X'00000978'
00000C	0000 8910	000003	DC	X'00008910'
000010	47F0 F001	000003	BC	R15,1(,R15)
000014		000003	EQU	*
000014	90EC D00C	000003	STM	R14,R12,12(,R13)
000018	4110 F024	000003	LA	R1,36(,R15)
00001C	98EF F034	000003	LM	R14,R15,52(,R15)
				# Skip over constant area
				# Eyecatcher: CEE
				# Stack Size
				# Offset to PPA1
				# Wrong Entry Point: cause exception
		L3282:		
				# Save GPRs Used
				# Args for boot strap routine
				#

000020	07FF	000003	BR	R15	#	Branch to boot strap routine
000022	0000	000003	DC	X'0000'	#	Available half-word
000024		000003	L3284: EQU	*	#	Boot Strap Info Block
000024	0000 0000	000003	DC	X'00000000'	#	address of entry label
000028	0000 0000	000003	DC	X'00000000'	#	WSA24 allocation size
00002C	0000 8A0C	000003	DC	X'00008A0C'	#	address of Saved Option String
000030	0000 8948	000003	DC	X'00008948'	#	address of entry point name
000034	0000 0054	000003	DC	X'00000054'	#	A(Label L3283)
000038	0000 0000	000003	DC	X'00000000'	#	address of boot strap routine(IGZXBST)
00003C		000003	L3285: EQU	*	#	CEE Parameter Block
00003C	0000 0024	000003	DC	X'00000024'	#	address of infoBlockLabel
000040	0000 8A1C	000003	DC	X'00008A1C'	#	A(PARMCEE-CEEPPARMBlock)
000044		000003	L3280: EQU	*	#	Handle growing stack
000044	58F0 C31C	000003	L	R15,796(,R12)	#	Load CEECAA0GETS
000048	184E	000003	LR	R4,R14	#	Required NAB
00004A	05EF	000003	BALR	R14,R15	#	Extend Stack
00004C	0000 0000	000003	DC	X'00000000'	#	Argument list size = 0
000050	A7F4 0009	000003	J	L3281	#	Branch back
000054		000003	@MAINENT DS	0H	#	PRIMARY ENTRY POINT ADDRESS
000054		000003	L3283: EQU	*	#	User Code Entry Point
000054	18EF	000003	LR	R14,R15	#	Load NAB
000056	4100 E978	000003	LA	R0,2424(,R14)	#	New NAB Address
00005A	5500 C314	000003	CL	R0,788(,R12)	#	Exceed current storage segment?
00005E	A724 FFF3	000003	JH	L3280	#	Yes: branch to recovery code
000062		000003	L3281: EQU	*	#	Stack now has sufficient room
000062	5000 E04C	000003	ST	R0,76(,R14)	#	Update NAB
000066	A708 0010	000003	LHI	R0,16	#	COBOL Language Word upper half
00006A	8900 0010	000003	SLL	R0,16	#	shift to upper half of register
00006E	A70A 0301	000003	AHI	R0,769	#	add COBOL Language Word lower half
000072	5000 E000	000003	ST	R0,0(,R14)	#	Save Language Word
000076	50D0 E004	000003	ST	R13,4(,R14)	#	Save Back Chain
00007A	18DE	000003	LR	R13,R14	#	Set new DSA
00007C	4100 D6D0	000003	LA	R0,1744(,R13)	#	Address of COBDSACB
000080	5000 D074	000003	ST	R0,116(,R13)	#	Saved in member slot1
000084	4100 0000	000003	LA	R0,0(,R0)	#	
000088	5000 D070	000003	ST	R0,112(,R13)	#	zero member slot0

- (1) COBOL プログラムの先頭からのオフセット
- (2) アセンブラー命令の 16 進表記
- (3) ソース行番号
- (4) COBOL プログラム用に生成される疑似アセンブラー表現
- (5) 疑似アセンブラー・コードを説明するコメント

AMODE 31 コンパイラと AMODE 64 コンパイラの LIST 出力の違い

AMODE 31 COBOL プログラムでは、エントリー・ポイントはモジュール内の最初の実行可能命令である。これはモジュールの始まりと一致し、常にオフセット 0 である。

AMODE 64 COBOL プログラムでは、エントリー・ポイントはモジュール内の最初の実行可能命令でもあるが、モジュールの先頭からのオフセット x'10'。AMODE 64 プログラムのエントリポイントの前の単語は、言語環境に準拠した XPLINK データレイアウトで文書化されているように、標準的な XPLINK 情報を含んでいます。

ZERODIVD というプログラムを AMODE 31 でビルドした場合、LIST コンパイラ出力のプログラム初期化部分には次のように表示される。最初の実行可能命令 (BC) がオフセット x'0'で発生していることに注意してください。

000000		000002	PROC		
ZERODIVD					
000000	47F0 F014	000002	BC	R15,20(,R15)	# Skip over
constant area					
000004	01C3 C5C5	000002	DC	X'01C3C5C5'	# Eyecatcher:
CEE					
000008	0000 0198	000002	DC	X'00000198'	# Stack
Size					
00000C	0000 0590	000002	DC	X'00000590'	# Offset to
PPA1					
000010	47F0 F001	000002	BC	R15,1(,R15)	# Wrong Entry
Point: cause exception					
000014		000002	L0098: EQU		
*					
000014	90EC D00C	000002	STM	R14,R12,12(,R13)	# Save GPRs
Used					
000018	41A0 F024	000002	LA	R10,36(,R15)	# Args for boot

```

strap routine
00001C 98EF F034      000002      LM      R14,R15,52(,R15)
#
000020 47F0 F020      000002      BC      R15,32(,R15)      # Branch to boot
strap routine
000024      000002      L0100:    EQU      *      # Boot Strap Info
Block
000024 0000 0000      000002      DC      X'00000000'      # address of entry
label
000028 0000 00A2      000002      DC      X'000000A2'      # Storage
allocation size
00002C 0000 06D6      000002      DC      X'000006D6'      # address of Saved
Option String
000030 0000 05D0      000002      DC      X'000005D0'      # address of entry
point name
000034 0000 005C      000002      DC      X'0000005C'      # A(Label
L0099)
000038 0000 0000      000002      DC      X'00000000'      # address of boot
strap routine(IGZXBST4)
etc...

```

このプログラムを AMODE 64 でビルドすると、LIST コンパイラ出力のプログラム初期化部分に次のように表示される。最初の実行可能命令（STMG）がオフセット x'10'で発生していることに注意してください。

```

000000      000002      PROC      ZERODIVD
000000 00C3 00C5      000002      DC      X'00C300C5'      # XPLINK eyecatcher
000004 00C5 00F1      000002      DC      X'00C500F1'      # XPLINK eyecatcher
000008 0000 0440      000002      DC      X'00000440'      # Offset to PPA1
00000C 0000 0200      000002      DC      X'00000200'      # DSA size
000010 EB6E 4610 0024 000002      STMG      R6,R14,1552(,R4)      # Entry point/
compile unit start
000016 E340 4E00 FF71      000002      LAY      R4,-512(,R4)      #
00001C 4100 48F0      000002      LA      R0,2288(,R4)      # Address of
COBDSACB
000020 E300 4868 0024      000002      STG      R0,2152(,R4)      # Saved in XPLINK
member slot1

000026 E350 4808 0024      000002      STG      R5,2056(,R4)      #

00002C C080 0000 0186      000002      LARL      R8,$CONSTANT_AREA
000032 B904 00A5      000002      LGR      R10,R5
000036 D7A3 48F0 48F0      000002      XC      2288(164,R4),2288(R4)      #
00003C E54C 48F0 0009      000002      MVHI      2288(,R4),X'0009'      #
000042 E3A0 4928 0024      000002      STG      R10,2344(,R4)      #
000048 9181 A0A4      000002      TM      164(,R10),X'81'      # IPCB_Status
00004C A714 0013      000002      JO      L0003
000050 B904 001A      000002      LGR      R1,R10

000054 4120 48F0      000002      LA      R2,2288(,R4)      # COBDSACB

000058 E350 A118 0004      000002      LG      R5,280(,R10)      #
00005E E360 A120 0004      000002      LG      R6,288(,R10)      #
etc...

```

AMODE 64 プログラムのエントリー・ポイントは、x'0'ではなくオフセット x'10' から始まるため、AMODE 64 プログラムのアベンドをデバッグする場合は、言語環境のアベンド・メッセージ (CEE320nS) のオフセット・アドレスに x'10'を追加して、コンパイル・リスト内のどのステートメントが失敗しているかを判断する必要があることに注意してください。詳しくは、[495 ページの『AMODE 64 プログラムのデバッグ』](#)を参照してください。

関連参照

[513 ページの『シグニチャー情報バイト』](#)

例: MD5 シグニチャー

MD5 シグニチャーに関する LIST 出力を以下に例示します。この情報は、アプリケーション・モジュールの DWARF デバッグ・データにも含まれています。MD5 シグニチャーは Timestamp and Version Information セクションの前の 16 バイトに位置します。

```

000608 AAEE 60C2 DAA3      =X'AAEE60C2DAA3'      md5 signature
00060E 776D AEB5 E753      =X'776DAEB5E753'      md5 signature
000614 E767 C4E1      =X'E767C4E1'      md5 signature

```

注: の MD5 署名は、TEST オプションが指定されている場合、または -g は次のように指定されます **cob2** 下 z/OS ユニックス。

MD5 シグニチャーの有無はコンパイル・フラグ・ビット PPA2 によって示されます。このビットが 1 に設定されている場合、MD5 シグニチャーは存在します。このビットが 0 に設定されている場合、MD5 シグニチャーは存在しません。PPA2 について詳しくは、*z/OS Language Environment Vendor Interfaces* を参照してください。

関連参照

526 ページの『例: プログラム・プロローグ域』

例: タイム・スタンプおよびバージョン情報

次の例は、コンパイラーのバージョンおよびコンパイル日時についての LIST 出力を示しています。

```
Timestamp and Version Information
0029C8 F2F0 F2F0 =C'2024' Compiled Year
0029CC F0F7 F1F7 =C'0531' Compiled Date MMDD
0029D0 F1F2 F3F1 F2F2 =C'123122' Compiled Time HHMMSS
0029D6 F0F6 F0F3 F0F0 =C'060400' VERSION
Timestamp and Version End
```

例: コンパイラー・オプションおよびプログラム情報

次の例は、コンパイラー・オプションおよびプログラム情報の LIST 出力を示しています。

```
DATA VALIDATION AND UPDATE PROGRAM IGYTCARA Date 05/31/2024 Time 10:48:16
Compiler Options and Program Information Section
(1) (2) (3) (4) (5)
0029DC 0030 =X'0030' Size of Compiler Options and Prog Info Section
0029DE (+00) 0474 =X'0474' UNSIGNED BINARY CODE PAGE CCSID VALUE
0029E0 (+02) 06 =X'06' ARCHITECTURE LEVEL
0029E1 (+03) 00 =X'00' OPTIMIZATION LEVEL
0029E2 (+04) 1406 =X'1406' INFO. BYTES 28-29
0029E4 (+06) 0000 =X'0000' RESERVED
0029E6 (+08) A04875CC2001 =X'A04875CC2001' INFO. BYTES 1-6
0029EC (+14) 100010884909 =X'100010884909' INFO. BYTES 7-12
0029F2 (+20) 002008800C00 =X'002008800C00' INFO. BYTES 13-18
0029F8 (+26) 000001A000 =X'000001A000' INFO. BYTES 19-23
0029FD (+31) 00 =X'00' COBOL SIGNATURE LEVEL
0029FE (+32) 0000002F =X'0000002F' # DATA DIVISION STATEMENTS
002A02 (+36) 0000005B =X'0000005B' # PROCEDURE DIVISION STATEMENTS
002A06 (+40) 18808008 =X'18808008' INFO. BYTES 24-27
002A0A (+44) E2F1F6F0F1F540 =C'PXXXXXX' BUILD LEVEL INFO
Compiler Options and Program Information Section End
```

- (1) プログラム・オブジェクト内のオフセット
- (2) 10 進でのオフセット
- (3) バイトの内容 (16 進形式)
- (4) バイトのアセンブラー表現
- (5) セクション内のバイトの説明

例: ソース・コードから生成されるアセンブラー・コード

次の例は、LIST コンパイラー・オプションを使用したときに、ソース・コードから生成されるアセンブラー・コードのリストを示しています。このリストを使用して、失敗した命令に対応する COBOL ステートメントを見つけることができます。

```
000964: display "PROGRAM IGYTCARA - Beginning". (1)
```

(2)	(3)	(4)	(5)	(6)
0001EA	E320 3394 0171	000964	LAY R2,5012(,R3)	#
0001F0	D203 D5E8 2000	000964	MVC 1512(4,R13),0(R2)	#
0001F6	E320 3398 0171	000964	LAY R2,5016(,R3)	#
0001FC	D203 D5EC 2000	000964	MVC 1516(4,R13),0(R2)	#
000202	4120 39C8	000964	LA R2,2504(,R3)	#
000206	5020 D5F0	000964	ST R2,1520(,R13)	#
00020A	E320 338C 0171	000964	LAY R2,5004(,R3)	#
000210	D203 D5F4 2000	000964	MVC 1524(4,R13),0(R2)	#
000216	E320 339C 0171	000964	LAY R2,5020(,R3)	#
00021C	D203 D5F8 2000	000964	MVC 1528(4,R13),0(R2)	#
000222	D703 D5FC D5FC	000964	XC 1532(4,R13),1532(R13)	#
000228	4110 D5E8	000964	LA R1,1512(,R13)	#
00022C	E3F0 31D4 0158	000964	LY R15,4564(,R3)	#
000232	58C0 D080	000964	L R12,128(,R13)	#
000236	0DEF	000964	BASR R14,R15	#
000965:	perform 050-create-vsam-main-file.			Call "IGZXDSP"
000238	5820 D670	000965	L R2,1648(,R13)	# VN_cell
00023C	5020 D544	000965	ST R2,1348(,R13)	# PfmSv_Cell
000240	C020 0000 0007	000965	LARL R2	
000246	5020 D670	000965	ST R2,1648(,R13)	# VN_cell
00024A	A7F4 02F4	000965	J 050-CREATE-VSAM-MAIN-FILE	
00024E	5820 D544	000965	L R2,1348(,R13)	# PfmSv_Cell
000252	5020 D670	000965	ST R2,1648(,R13)	# VN_cell

- (1) 疑似アセンブラー命令が散在するソース・コード
- (2) モジュール内でのオブジェクト・コード命令の相対位置 (16 進表記)
- (3) オブジェクト・コード命令 (16 進表記)
16 進数字の最初の 2 桁または 4 桁が命令で、残りの桁は命令オペランドです。2 つのオペランドのある命令もあります。
- (4) このアセンブラー・コードに関連付けられたソース行番号
- (5) オブジェクト・コード命令 (コンパイラ生成の疑似アセンブラー)
- (6) 命令と命令で使用されるオペランドの説明

関連参照

[510 ページの『LIST および MAP 出力で使用される記号』](#)

例: プログラム・プロローグ域

次の例は、プログラム・プロローグ域の LIST 出力を示しています。プログラム・プロローグ域 (PPA) は、コンパイル済みプログラムに関する情報を含むいくつかのセクションから構成されます。

コンパイラによって生成されたプロシージャーを含め、プログラム内のプロシージャーごとに PPA1 があります。対応する PPA1 へのオフセットが、各プロシージャーの先頭からのオフセット 12 (X'C') に記録されています。PPA1 には、PPA2 セクションおよび PPA3 セクションに対するオフセットとともに、プロシージャーに関する情報が含まれています。

プログラム・プロローグ域を使用してリスト・ファイル内の情報を検索する方法については、『z/OS Language Environment Vendor Interfaces』を参照してください。

DATA VALIDATION AND UPDATE PROGRAM IGYTCARA Date 05/31/2024 Time 10:48:16			
(1)	(2)	(3)	(4)
0081E0	1CCEA506	PPA1: Entry Point Constants	
0081E4	00008310	=F'483304710'	Flags
0081E8	00008378	=A(PPA2-IGYTCARA)	
0081EC	00000000	=A(PPA3-IGYTCARA)	
0081F0	FFFE0000	=F'0'	No EPD
0081F4	40000000	=F'-131072'	Register Save Mask
0081F8	90	=F'1073741824'	Member Flags
0081F9	000978	=AL1(144)	Flags
0081FC	0000	=AL3(2424)	Callee's DSA use/8
0081FE	0012	=AL1(0)	Flags
008200	D00006D0	=H'18'	Offset/2 to CDL
008204	00000000	=F'-805304624'	State variable location
008208	00000000	=F'0'	CDL function length/2
00820C	00000000	=F'0'	CDL function EP offset
008210	00000000	=F'0'	CDL prolog
008214	00000000	=F'0'	CDL epilog
008218	0008 ****	=F'0'	CDL end
		AL2(8),C'IGYTCARA'	

```
PPA1  End
```

プログラムごとに1つのPPA2があります。PPA2に対するオフセットが、各PPA1に記録されています。のPPA2 オフセットが含まれています Timestamp and Version Information リストのセクションと PPA4 セクション。

TEST オプションが有効になっていない場合の PPA2 セクションは次のようになります。

```
PPA2:  Entry Point Constants
000800 04002203      =F'67117571'      Flags
000804 FFFFFFF800    =A(CEESTART-PPA2)
000808 00000058      =F'88'            A(PPA4-PPA2)
00080C FFFFFFFB00    =A(TIMESTAMP-PPA2)
000810 FFFFFFF800    =A(PrimaryEntryPoint-PPA2)
000814 02200000      =F'35651584'      Flags
PPA2  End
```

TEST オプションが有効になっている場合の PPA2 セクションは次のようになります。

```
PPA2:  Entry Point Constants
000830 04002203      =F'67117571'      Flags
000834 FFFFFFF7D0    =A(CEESTART-PPA2)
000838 00000058      =F'88'            A(PPA4-PPA2)
00083C FFFFFFFB00    =A(TIMESTAMP-PPA2)
000840 FFFFFFF7D0    =A(PrimaryEntryPoint-PPA2)
000844 02600000      =F'39845888'      Flags
PPA2  End
```

COBOL ソース・ファイルでは各プログラム (ネストされた各プログラムを含む) に PPA3 が1つあります。各エントリーには、PPA3 自体、ベース・ロケーター・テーブル、および特殊レジスター・テーブルを基準にしたオフセットが含まれています。また、PPA3 には、プログラムの先頭から最初の COBOL ステートメントまでのオフセットも含まれています。

```
PPA3:  Entry Point Constants
0014D8 00000000      =F'0'             Flags
0014DC 000000C0      =F'192'           A(Base_Locator_Table-PPA3)
0014E0 000000D8      =F'216'           A(Special_Register_Table-PPA3)
0014E4 00000184      =X'184'           A(User_Entry-CUEntry)
PPA3  End
```

プログラムごとに1つの PPA4 があります。さまざまなコンパイラ生成テーブルへのオフセットを持ちます。ストレージ (の Initial Heap Storage Map そして Below The Line Storage Map セクション)。PPA4 に対するオフセットは、PPA2 のフィールドに記録されています。

```
PPA4:  Entry Point Constants
000710 22000000      =F'570425344'     Flags 1
000714 00020100      =F'131328'        Flags 2
000718 00000000      =F'0'             A(NORENTstatic)
00071C 00000000      =F'0'             Q(RENTstatic)
000720 0000006C      =F'108'           A(DATA31_address_cell-RENTstatic)
000724 FFFFFFF8F0    =F'-1808'         A(Code-PPA4)
000728 000000760     =F'1888'         Code Length
00072C 00000000      =F'0'             Length NORENTstatic
000730 00000070      =F'112'           Length RENTstatic
000734 00000094      =F'148'           Length DATA31
000738 003F          =X'3F'           A(CUName-PPA4)
00073A 0000          =X'0'           PPA4 Minor Ver
000744 7FFFFFFF      =X'7FFFFFFF'     Offset UsrWkrStrg
000748 00000000      =X'0'           Length UsrWkrStrg
00074C 00           =X'0'           Has Externals
00074D 0000          =X'0'           A(SYSDEBUGName-PPA4)
PPA4  End
```

- (1) オブジェクト・モジュール内の PPA フィールドの相対位置 (16 進形式)
- (2) フィールドの内容 (16 進数)
- (3) アセンブラーに類似したフィールド定義構文
- (4) フィールドの内容の説明

関連参照

WORKING-STORAGE SECTION の変更

例: 初期ヒープ・ストレージ・マップ

リストにある3つのマップ・セクション (INITIAL HEAP STORAGE MAP、WORKING-STORAGE MAP、および BELOW THE LINE STORAGE MAP) はまとめてプログラムの初期ヒープ・ストレージ・マップと呼ばれます。これらの領域に対するストレージはプログラムの開始時に割り振られ、実行単位の終了まで、またはプログラムが取り消されるまで維持されます。

これら3つのマップ・セクションのレイアウトはよく似ています。

- 最初の列には、コンパイラーによって割り振られたストレージのブロックを基点にした項目のオフセットが示されます。
- 2番目の列はシンボルのサイズで、これにはそのすべてのサブレベル・メンバーが含まれます。
- 3番目の列には領域の記述名が示されます。

レイアウトの例として以下の INITIAL HEAP STORAGE MAP を参照してください。NORENT コンパイラー・オプションが有効になっている場合、WORKING-STORAGE データ項目は INITIAL HEAP STORAGE MAP セクションでマップされます。COBOL データ項目の場合、オフセットは、コンパイラーによって割り振られたストレージのブロックからレベル1 データ項目の先頭までのオフセットです。このブロックの開始アドレスは、Constant Area にあります。

```
***** INITIAL HEAP STORAGE MAP *****
      OFFSET (HEX)   LENGTH (HEX)   NAME
              0             4      BLL_Ptrs
              4             C      BLT_Ptrs
             10            60      GPCB
             70             4      WS-BASE-ADDRESS
             74             8      TS2=6
```

コンパイラー・オプション RENT および DATA(31) が有効になっている場合、WORKING-STORAGE データ項目は WORKING-STORAGE MAP の下に表示されます。オプション RENT および DATA(24) が有効になっている場合、WORKING-STORAGE データ項目は BELOW THE LINE STORAGE MAP の下に表示されます。コンパイラーによって内部データ項目が生成されてロケーターがマップされている場合は、INITIAL HEAP STORAGE MAP セクションも表示されます。

```
***** BELOW THE LINE STORAGE MAP *****
      OFFSET (HEX)   LENGTH (HEX)   NAME
              0             4      JNIENVPTR
              8             2      RETURN-CODE
             10             2      SORT-RETURN
             18             8      SORT-CONTROL
             20             4      SORT-CORE-SIZE
             28             4      SORT-FILE-SIZE
             30             4      SORT-MODE-SIZE
             38             8      SORT-MESSAGE
             40             4      TALLY
             48             1      SHIFT-OUT
             50             1      SHIFT-IN
             58             4      XML-CODE
             60            1E      XML-EVENT
             80             4      XML-INFORMATION
             88            50      COMMUTER-FILE
             D8            50      COMMUTER-FILE-MST
            128            7A      PRINT-FILE
            1A8             1      WORKING-STORAGE-FOR-IGYCARA
            1B0             2      COMP-CODE
            1B8             3      WS-TYPE
            1C0             2      I-F-STATUS-AREA
            1C8             8      STATUS-AREA
            1D0             2      UPDATE-FILE-STATUS
```

例: 定数域

次の例は、COBOL ソースからのストリングおよびその他のリテラルと、コンパイラーによって生成されたストリングおよびその他のリテラルに関する LIST 出力を示しています。

コンパイラは、Constant Area 開始アドレスをロードすることにより Constant Area それぞれの定数またはリテラルに固定オフセットを追加します。

(1) (2)		CONSTANT AREA:		(3)	(4)
006A98 (+0)	00CCDDFF 00000000	C9C7E8E3	C3C1D9C1 00000000	00000000 C9C7E9E2 D9E3C3C4	
.....IGYTCARA.....	IGZSRTCD				
006AB8 (+32)	40000A00 40000000	00000008	00000000 E2E8E2D6	E4E34040 00100000 00000000	
.....SYSOUT					
006AD8 (+64)	0E000000 00000001	0F000000	0000001E 00000000	40000000 00000003 0064003C	
.....					
006AF8 (+96)	000FE800 9F0F0000	00000011	00000000 E3D9C1D5	E2C1C3E3 4B40C3D6 C4C50000	..Y.....TRANSACTION.
CODE..					
006B18 (+128)	0000000E 00000000	E2C8C9C6	E340C3D6 C4C50040	40400000 C8D6D4C5 40D3D6C3	SHIFT CODE ..HOME
LOC					
006B38 (+160)	4B40C3D6 C4C50000	E6D6D9D2	40D3D6C3 4B40C3D6	C4C50000 D3C1E2E3 40D5C1D4	. CODE..WORK LOC. CODE..LAST
NAM					
006B58 (+192)	C5404040 40400000	C9D5C9E3	C9C1D3E2 40404040	40400000 C4E4D7D3 C9C3C1E3	
E ..INITIALS ..	..DUPLICAT				
006B78 (+224)	C540D9C5 C34B0000	D9C5C34B	40D5D6E3 40C6D6E4	D5C40000 C1C4C4D9 C5E2E240	E REC...REC. NOT
FOUND..ADDRESS					
006B98 (+256)	40404040 40400000	C3C9E3E8	40404040 40404040	40400000 E2E3C1E3 C540C3D6	..CITY ..STATE
CO					
006BB8 (+288)	C4C54040 40400000	E9C9D7C3	D6C4C540 40404040	40400000 C8D6D4C5 40D7C8D6	DE ..ZIPCODE ..HOME
PHO					
006BD8 (+320)	D5C54040 40400000	E6D6D9D2	40D7C8D6 D5C54040	40400000 C8D6D4C5 40D1E4D5	NE ..WORK PHONE ..HOME
JUN					
006BF8 (+352)	C3E3C9D6 D5400000	E6D6D9D2	40D1E4D5 C3E3C9D6	D5400000 C4D9C9E5 C9D5C740	CTION ..WORK
JUNCTION ..DRIVING					
006C18 (+384)	E2E3C1E3 E4E20000	40D9C5D7	D6D9E340 407B7A40	C9C7E8E3 C3C1D9C1 40404040	STATUS.. REPORT #:
IGYTCARA					
006C38 (+416)	40404040 40404040	40404040	40404040 40404040	40404040 40404000 00000033	
					
006C58 (+448)	C3D6D4D4 E4E3C5D9	40C6C9D3	C540E4D7 C4C1E3C5	40D3C9E2 E3404040 40404040	COMMUTER FILE UPDATE
LIST					
006C78 (+480)	40404040 40404040	40404040	40404040 40400000	00000032 40404040 40404040	
					
006C98 (+512)	D7C1C7C5 407B7A40	00000000	00000010 40D7D9D6	C7D9C1D4 407B7A40 C9C7E8E3	PAGE #: PROGRAM #:
IGYT					
006CB8 (+544)	C3C1D9C1 40404040	404040D9	E4D540E3 C9D4C57A	40000000 00000025 7A000000	CARA RUN
TIME:					
006CD8 (+576)	00000030 00000000	D9E4D540	C4C1E3C5 7A400000	0000000A 61000000 0000000B	RUN
DATE:					

- (1) CSECT 内のオフセット。
- (2) ベース 10 でのオフセット。
- (3) 定数域のバイトを含んだ 8 列
- (4) 文字表現。 表示できない文字にはドット (.) が使用されています。

例: ベース・ロケーター・テーブル

次の例は、ベース・ロケーター・テーブルの LIST 出力を示しています。

Base Locator Table			
008AB0	01	=X'1'	Table Version
008AB1	00	=X'0'	Reserved
008AB2	0008	=H'8'	Header length
008AB4	00000010	=F'16'	Array byte length
008AB8	2A00	=X'2A00'	Flags & info (element 1)
008ABA	00000014	=X'14'	Offset to cells
008ABE	03	=X'3'	Cell count
008ABF	0A00	=X'A00'	Flags & info (element 2)
008AC1	00000000	=X'0'	Offset to cells
008AC5	05	=X'5'	Cell count
008AC6	0000	=X'0'	Flags & info (end of
array)			
Base Locator Table End			

ベース・ロケーター・テーブルについて詳しくは、「z/OS Language Environment Vendor Interfaces」を参照してください。

例: 特殊レジスター・テーブル

特殊レジスター・テーブルの LIST 出力を以下に例示します。特殊レジスター・テーブルの形式はベース・ロケータ・テーブルの形式とよく似ています。

Special Register Table			
0015B0	01	=X'1'	Table Version
0015B1	00	=X'0'	Reserved
0015B2	0008	=H'8'	Header length
0015B4	00000006	=F'6'	Array byte length
0015B8	12	=X'12'	Flags & info (element 1)
0015B9	00000018	=X'18'	Offset to cells
0015BD	00	=X'0'	Flags & info (end of
array)			
Special Register Table End			

特殊レジスター・テーブルに含まれる各エントリーは以下の項目で構成されています。

- 以下の情報を表す 1 バイト
 - 特殊レジスター ID 番号 (ビット 0 からビット 4 まで)。ID = 1 は RETURN-CODE レジスターを表します
 - アクセス・モード (ビット 5 からビット 8 まで)
 - MODE = 0 の場合、基底アドレス = スタックの先頭
 - MODE = 1 の場合、基底アドレス = NORENT 静的
 - MODE = 2 の場合、基底アドレス = 32 ビット RENT 静的
 - MODE = 3 の場合、24 ビット NORENT 静的
- 特殊レジスターに対するオフセット

特殊レジスター・テーブルの最後はヌル・バイトで示されます。

例: 外部シンボル

次の例は、プログラムで定義または参照される外部シンボルの LIST 出力を示しています。外部シンボル辞書には、プログラム内で定義または参照される外部シンボルごとに 1 つの項目が含まれています。

各項目には、アドレス、長さ、およびシンボル・タイプが含まれています。シンボル・タイプには以下があります。

- ED 外部定義
- SD セクション定義
- LD ラベル定義
- ER 外部参照
- PR 疑似レジスター

EXTERNAL SYMBOL DICTIONARY					
	TYPE	ID	ADDR	LENGTH	NAME
	SD	1	000000	000000	IGYTCARA
	ED	2	000000	000000	C_CEESG003
	ED	3	000000	008AC8	C_CODE
	LD	4	000000	000000	IGYTCARA#C

ER	5	000000	000000	CEESTART
ER	6	000000	000000	CEEBETBL
ED	7	000000	000000	C_WSA
PR	8	000000	002204	IGYTCARA#S
ED	9	000000	000022	B_IDRL
ER	10	000000	000000	IGZXBST
ER	11	000000	000000	IGYTCARA
ER	12	000000	000000	IGZXPRS
ER	13	000000	000000	IGZXCMSG
ER	14	000000	000000	IGZXDSP
ER	15	000000	000000	IGZXVCLS

例: DSA メモリー・マップ(スタック・ストレージ・マップ)

次の例は、動的保管域 (DSA) の LIST 出力を示しています。DSA には、別個にコンパイルされたプロシージャに入ったときに獲得されたストレージの内容に関する情報が含まれています。

```

* * * * *   S T A C K       S T O R A G E       M A P   * * * * *
      1           2           3
      OFFSET (HEX)  LENGTH (HEX)  NAME

Block name: IGYTCARA

      80           4      _@CAA
      C8           3      _BEtemp200
      CC           3      _BEtemp204
      D0           3      _BEtemp208
      D4           3      _BEtemp212
      D8           3      _BEtemp216
      DC           3      _BEtemp220
      E0           3      _BEtemp224
      E4           3      _BEtemp228
      E8           10     _BEtemp232
      F8           20     _BEtemp248
     118           20     _BEtemp280
     138           4      _BEtemp312
     13C           4      _BEtemp316
     140           4      _BEtemp320
     144           4      _BEtemp324
     148           4      _BEtemp328
     14C           4      _BEtemp332
     150           4      _BEtemp336
     154           4      _BEtemp340
     158           4      _BEtemp344
     15C           4      _BEtemp348
     160           4      _BEtemp352
     164           4      _BEtemp356
     168           4      _BEtemp360
     16C           4      _BEtemp364
     170           4      _BEtemp368
     174           4      _BEtemp372
     178           4      _BEtemp376

```

- (1) DSA の先頭からの DSA フィールドの 16 進オフセット
- (2) DSA フィールド長 (16 進数)
- (3) 記号名

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。

例 1

以下の例は、SOURCE(DEC) が有効である場合の XREF 出力を示しています。

An "M" preceding a data-name reference indicates that the data-name is modified by this reference.

(1A) Defined	(2) Cross-reference of data-names	(3) References
265	ABEND-ITEM1	
266	ABEND-ITEM2	
347	ADD-CODE	1102 1162
381	ADDRESS-ERROR.	M1126
280	AREA-CODE.	1236 1261 1324 1345
382	CITY-ERROR	M1129
1C	A.	1D
39	A.	3A
38	AG	

(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) Defined	(6) Cross-reference of procedures	(7) References
877	000-DO-MAIN-LOGIC	
930	050-CREATE-STL-MAIN-FILE . .	P879
982	100-INITIALIZE-PARAGRAPH . . .	P880
1441	1100-PRINT-I-F-HEADINGS. . . .	P915
1481	1200-PRINT-I-F-DATA.	P916
1543	1210-GET-MILES-TIME.	P1510
1636	1220-STORE-MILES-TIME.	P1511
1652	1230-PRINT-SUB-I-F-DATA. . . .	P1532
1676	1240-COMPUTE-SUMMARY	P1533
1050	200-EDIT-UPDATE-TRANSACTION. .	P886
1124	210-EDIT-THE-REST.	P1116
1159	300-UPDATE-COMMUTER-RECORD . .	P888
1207	310-FORMAT-COMMUTER-RECORD . .	P1164 P1179
1258	320-PRINT-COMMUTER-RECORD. . .	P1165 P1176 P1182 P1192
1288	330-PRINT-REPORT	P1178 P1202 P1256 P1280 P1340 P1365 P1369
1312	400-PRINT-TRANSACTION-ERRORS .	P890

例 2

以下の例は、SOURCE(HEX) が有効である場合の XREF 出力を示しています。

An "M" preceding a data-name reference indicates that the data-name is modified by this reference.

(1B) Defined	(2) Cross-reference of data-names	(3) References
1C	A.	1D
39	A.	3A
38	AG	

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。

533 ページの『例: XREF 出力: プログラム名相互参照』

533 ページの『例: XREF 出力: COPY/BASIS 相互参照』

535 ページの『例: XREF 出力: 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、ライブラリ名とコピーブックの相互参照を示しています。データセット名関連するコピーブックの XREF コンパイラオプション下 z/OS。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				
(1) Text-name (Member)	(1) Library (DDNAME)	(2) File name (Data set name)	(3) Concat Level	(4) ISPF Created
ACTIONS	OTHERLIB	USERID.COBOL.COPY	0	1992/07/11
ACTIONS	SYSLIB	USERID.COBOL.COPY	0	1992/07/11
CUSTOMER	ALTDDXXY	USERID.COBOL.LIB3	0	2007/06/01
CUSTOMER	SYSLIB	USERID.COBOL.LIB2PDSE	1	2007/06/07
HOUSE	ALTDDXXY	USERID.COBOL.LIB2	1	2007/06/07
HOUSE	SYSLIB	USERID.COBOL.LIB2PDSE	1	
IMOTOR	SYSLIB	USERID.COBOL.LIB4X	3	2007/06/07

ISOVERFY	SYSLIB	USERID.COBOL.COPY	0
NSMAP	SYSLIB	USERID.COBOL.LIB3	2

- (1) Text-name (テキスト名) および Library (ライブラリー: Library-name (ライブラリー名) の省略形) は、ソースのステートメント `COPY text-name OF library-name` (例: `COPY ACTIONS OF OTHERLIB`) からのものです。
- (2) COPY メンバーがコピーされたデータ・セットの名前。
- (3) 連結レベルの省略形。指定されたデータ・セットが指定された DD 名の連結で、最初のデータ・セットから何レベルの深さにあるのかを示します。
- 例えば、上の例の 4 つのデータ・セットは DD 名 SYSLIB に連結されています。

DDNAME	DSNAME	(concatenation level)
SYSLIB DD	DSN=USERID.COBOL.COPY,	0
DD	DSN=USERID.COBOL.LIB2PDSE,	1
DD	DSN=USERID.COBOL.LIB3,	2
DD	DSN=USERID.COBOL.LIB4X	3

- 例えば、上のリストで表示されているメンバー NSMAP は、SYSLIB 連結の最初のデータ・セットから 2 レベル下にある、データ・セット USERID.COBOL.LIB3 で検出されたということになります。
- (4) PDSE が ISPF において STATS ON で編集された場合は、作成日が表示されます。
- ヒント:** 下 z/OSSYSLIB 連結に複数のデータセットがある場合、COPY/BASIS 相互参照は、場合によっては不完全であったり欠落している可能性があります。詳細については、XREF コンパイラー・オプションに関する関連参照を参照してください。
- z/OS ユニックス シェルでコンパイルした場合、相互参照は以下の抜粋のように表示されます。

COPY/BASIS cross-reference of text-names, library names, and file names		
(5) Text-name	(5) Library-name	(6) File name
'/copydir/copyM.cbl'	SYSLIB	/u/JSMITH/cobol//copydir/copyM.cbl
'/copyA.cpy'	SYSLIB	/u/JSMITH/cobol//copyA.cpy
'cobol/copyA.cpy'	ALTDD2	/u/JSMITH/cobol/copyA.cpy
'copy/stuff.cpy'	ALTDD2	/u/JSMITH/copy/stuff.cpy
'copydir/copyM.cbl'	SYSLIB	/u/JSMITH/cobol/copydir/copyM.cbl
'copydir/copyM.cbl'	SYSLIB (default)	/u/JSMITH/cobol/copydir/copyM.cbl
'stuff.cpy'	ALTDD	/u/JSMITH/copy/stuff.cpy
"copyA.cpy"	(7) SYSLIB (default)	/u/JSMITH/cobol/copyA.cpy
"reallyXXVeryLongLon>	SYSLIB (default)	(8) <JSMITH/cobol/
reallyXXVeryLongLongName.cpy		
OTHERDD	ALTDD2	/u/JSMITH//copy/other.cob
. . .		
Note: Some names were truncated. > = truncated on right < = truncated on left		

(5)

から COPY ソース内のステートメント。たとえば、COPY 対応する声明三番目上記の相互参照の項目は次のようになります。

```
COPY 'cobol/copyA.cpy' Of ALTDD2
```

(6)

COPY メンバーをコピーした元のファイルの完全修飾パス

(7)

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。

(8)

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。

関連参照

463 ページの『外部参照』

例: XREF 出力: 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-main-file.          932 (1)
000882                  perform 100-initialize-paragraph.            984
000883                  read update-transaction-file into ws-transaction-record 204 340
000884                      at end
000885                      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
000992              location-file                                  204
000993              i-o commuter-file                             193
000994              output print-file                             181
. . .
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
001456              after new-page.                                222 635
. . .
```

(1)

プログラム内のデータ名またはプロシージャー名の定義の行番号。

(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: OFFSET コンパイラー出力

次の例は、圧縮されたステートメントのリスト、グローバル・テーブル、WORKING-STORAGE 情報、およびリテラルが含まれているコンパイラー・リストを示しています。このリストは、OFFSET コンパイラー・オプションからの出力です。

DATA VALIDATION AND UPDATE PROGRAM IGYTCARA Date 05/31/2024 Time 10:48:16								
(1)	(2)	(3)						
LINEID	HEXLOC	VERBCODE	LINEID	HEXLOC	VERBCODE	LINEID	HEXLOC	VERBCODE
000011	0001B8	DISPLAY	000012	0001CC	PERFORM	000021	0001D0	IF
000022	0001E2	MOVE	000024	0001EA	MOVE	000012	000204	PERFORM
000013	000214	DISPLAY	000014	000228	PERFORM	000021	00022C	IF
000022	00023E	MOVE	000024	000246	MOVE	000014	00024A	PERFORM
000015	00025A	DISPLAY	000021	000272	IF	000022	000284	MOVE
000024	00028C	MOVE	000016	000290	PERFORM	000017	0002A0	DISPLAY
000018	0002B8	STOP						

(1)

行番号。ユーザーの行番号またはコンパイラー生成の行番号がリストされます。

(2)

この動詞用に生成されたコードの、プログラムの先頭からのオフセット (16 進表記)。

動詞は出現順にリストされます。直前の命令とは異なる行/ステートメントのペアに対応する命令が生成されるたびにリストされます。

(3)

使用されるステートメント。

注: エラーメッセージのフォーマットに使用されている行外コードのため、言語環境生成されたオフセットは、「コンパイル単位から{name}エントリーポイントで{name}コンパイル単位オフセット{offset} ...」、プログラムのオフセット範囲外である可能性があります。このような場合は、COBOL メッセージのステートメント番号 (IGZnnnns) を参照して、問題を特定してください。

関連参照

[427 ページの『OFFSET』](#)

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ステートメント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	1010 1012
2	ADD	1290 1306
1	CALL	1406
5	CLOSE	898 945 970 1526 1535
20	COMPUTE	1506 1640 1644 1657 1660 1663 1664 1665 1678 1682 1686 1691 1696 1701 1709 1713
		1718 1723 1728 1733
2	CONTINUE	1062 1069
2	DELETE	964 1193
48	DISPLAY	878 906 917 918 919 933 940 942 947 953 960 966 972 996 997 998 999 1003 1006 1037
		1090 1168 1171 1185 1195 1387 1388 1389 1390 1391 1392 1393 1401 1402 1403 1404
		1405 1433 1485 1486 1492 1497 1498 1520 1521 1528 1529 1624
2	EVALUATE	1161 1557
47	IF	887 905 932 939 946 952 959 965 971 993 1002 1036 1051 1054 1071 1074 1077 1089
		1102 1111 1115 1125 1128 1131 1134 1137 1141 1145 1148 1151 1167 1184 1194 1240
		1247 1265 1272 1289 1321 1330 1339 1351 1361 1484 1496 1519 1527
183	MOVE	907 937 957 983 984 985 986 987 988 1004 1011 1013 1025 1038 1052 1055 1060 1067
		1072 1075 1078 1079 1080 1081 1082 1083 1091 1103 1112 1126 1129 1132 1135 1139
		1143 1146 1149 1152 1160 1163 1169 1175 1177 1180 1181 1186 1191 1196 1201 1208
		1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1229 1230

			1231 1232 1233 1234 1235 1239 1241 1244 1248 1250 1251 1253 1254 1255 1257 1258
			1259 1260 1264 1266 1269 1273 1275 1276 1278 1279 1291 1294 1299 1301 1303 1307
			1313 1314 1315 1316 1317 1318 1319 1320 1322 1323 1327 1328 1331 1333 1334 1336
			1338 1341 1342 1343 1344 1348 1349 1352 1354 1355 1357 1362 1364 1368 1374 1375
			1376 1377 1378 1379 1380 1381 1414 1417 1422 1425 1445 1446 1447 1448 1450 1451
			1452 1457 1464 1489 1502 1507 1508 1509 1517 1551 1561 1566 1571 1576 1581 1586
			1591 1596 1601 1606 1611 1616 1621 1626 1627 1679 1683 1688 1693 1698 1703 1710
			1715 1720 1725 1730 1735
5	OPEN		931 951 989 1443 1483
62	PERFORM.		879 880 885 886 888 890 892 908 909 915 916 934 935 941 943 948 949 954 955 961
			962 967 968 973 974 1000 1005 1008 1023 1039 1092 1093 1116 1164 1165 1170 1172
			1176 1178 1179 1182 1187 1188 1192 1197 1198 1202 1246 1256 1271 1280 1329 1340
			1350 1359 1365 1369 1504 1510 1511 1532 1533
8	READ		881 893 958 1014 1026 1085 1490 1514
1	REWRITE.		1183
4	SEARCH		1058 1065 1413 1421
46	SET.		883 895 1016 1028 1041 1057 1064 1084 1087 1363 1412 1420 1493 1499 1516 1522 1548
			1550 1559 1560 1564 1565 1569 1570 1574 1575 1579 1580 1584 1585 1589 1590 1594
			1595 1599 1600 1604 1605 1609 1610 1614 1615 1619 1620 1639 1643
2	STOP		920 1434
4	STRING		1236 1261 1324 1345
33	WRITE.		938 1166 1292 1293 1295 1296 1297 1298 1300 1302 1305 1454 1459 1462 1465 1467 1469
			1471 1512 1654 1655 1667 1668 1669 1740 1742 1744 1745 1746 1747 1748 1749 1750

- (1) そのステートメントがプログラムで使用されている回数。
- (2) ステートメント
- (3) そのステートメントが使用されている行番号。

例: 条件付きコンパイル出力

次の例は、条件付きコンパイル ステートメントを含むプログラムのリストを示しています。

例 1

この例では、デフォルトの条件付きコンパイル出力を示します。 リスト内の注釈番号は、番号が付いた後続の説明と対応しています。

LineID	PL	SL	-----*A-1-B-----2-----3-----4-----5-----6-----7- -----8	Map and Cross Reference
000001			identification division.	
000002			program-id. prog.	
000003			data division.	
000004			working-storage section.	
000005			01 x pic 9(9) binary.	
000006			procedure division.	
000007			MainProgram.	
000008			>>define var as 12	
000009			>>evaluate var	
000010			>>when 10	
000011	*		display 'var is 10'	(1)
000012			>>when 11 thru 13	
000013			display 'var is 11, 12 or 13'	(2)
000014			>>when other	
000015	*		display 'invalid value'	(1)
000016			>>end-evaluate	
000017			goback.	
000018			end program prog.	

- (1) EVALUATE ディレクティブのこれらのブランチは、コンパイル時に false であったため、それらの中のコードは結果のプログラムから省略されました。
- (2) EVALUATE ディレクティブのこのブランチは、コンパイル時に true と評価されたため、その中のコードは結果のプログラムに組み込まれました。

例 2

この例では、条件付きコンパイルの出力を示します。CONDCOMP(SKIPSRC)が有効です。 ラベルの付いた偽の枝に注目してください (1) 前の例のは省略されています。

LineID	PL	SL	-----A-1-B-----2-----3-----4-----5-----6-----7- -----8	Map and Cross Reference
000001			identification division.	

```

000002      program-id. prog.
000003      data division.
000004      working-storage section.
000005      01 x pic 9(9) binary.
000006      procedure division.
000007      MainProgram.
000008          >>define var as 12
000009          >>evaluate var
000010          >>when 10
000011              >>when 11 thru 13
000012                  display 'var is 11, 12 or 13'          (2)
000013              >>when other
000014                  >>end-evaluate
000015                  goback.
000016      end program prog.
000017
000018

```

関連参照

EVALUATE ディレクティブ (*Enterprise COBOL for z/OS 言語解説書*)

条件付きコンパイル (*Enterprise COBOL for z/OS 言語解説書*)

CEEDUMP 処理での情報の抑止 (IGZ10PT)

TEST (DWARF) が有効な場合、WORKING-STORAGE SECTION のサイズによっては、CEEDUMP 処理には、大量の情報がダンプに含まれる可能性があります。この情報は、ジョブの JCL をセットアップすることで実行時に抑止できます。

このタスクについて

1. 言語環境 サンプル・データ・セット .SCEESAMP で、サンプル JCL IGZ10PT を使用して、IGZUOPT というロード・モジュールを作成します。JOB カードおよびロード・ライブラリー名を変更し、この JCL を実行して IGZUOPT を生成します。
2. このモジュールを STEPLIB 連結内のデータ・セットに入れると、CEEDUMP 処理中に DWARF 情報が抑止されます。これにより、CEEDUMP 内の COBOL プログラムの出力の量を減らすことができます。

JCL の JOB STEP1 は、IGZXOPT という MACRO を呼び出すアセンブラー・プログラムを翻訳します。このマクロは、特殊な COBOL ランタイム・オプションを指定するために使用されます。現在、以下の構文では、SKIPDWARF オプションのみがサポートされています。

```
IGZXOPT SKIPDWARF=ON | OFF
```

SKIPDWARF の設定値は ON または OFF で、デフォルト値は OFF です。

- ON を指定すると、CEEDUMP での DWARF 処理は抑止されます。
- OFF が指定されている場合、DWARF 処理は正常に続行されます。OFF を指定することは、STEPLIB から IGZUOPT を省略することと同じです。

関連参照

448 ページの『TEST』

COBOL プログラムのレポートの生成

DD 名を使用する IGZPROUTJCL の実行ステップで、Enterprise COBOL 5 以降のリリースでコンパイルされた COBOL プログラムのレポートを生成します。

このタスクについて

レポートには、ID、プログラム名、コンパイラー バージョン、コンパイルの日時、サービス情報などの詳細が記載されます。詳細については SERVICE オプションについては、440 ページの『サービス』。

動的、静的、DLL 経由で呼び出されるプログラムは、プログラム終了時にアクティブであれば、COBOL ACTIVE PROGRAMS。キャンセルされたダイナミックコールされたプログラムの詳細は、CANCELLED PROGRAMS の見出しの下に別途印刷される。静的に呼び出されるプログラム、および DLL リンクを介して

呼び出されるプログラムはキャンセルできません。そのため本報告書では、これらのプログラムを CANCELLED PROGRAMS の見出しに含めようとはしていない。CANCEL ステートメントの詳細については、ステートメントを参照してください。Enterprise COBOL for z/OS 言語解説書. DLL プログラムの詳細については、645 ページの『DLL リンケージと動的呼び出しの併用』を参照してください。

注：プログラムが複数回呼び出されたりキャンセルされたりした場合、レポートにはそのプログラムが 1 回だけリストされます。COBOL ACTIVE PROGRAMS または CANCELLED PROGRAMS 終了時のステータスに基づいて見出しが付けられます。例えば、プログラムが終了時にキャンセルされた後、再び動的に呼び出される場合、COBOL ACTIVE PROGRAMS。番組が終了時に再びキャンセルされた場合、CANCELLED PROGRAMS の見出しで表示される。

各動的、静的、または DLL プログラムには、一意の 6 桁の ID が割り当てられます。メインプログラムを識別するために、6 桁の ID の前に文字「M」が付きます。生成されたレポートでは、プログラム名は 8 文字に制限され、サービス文字列は 10 文字に制限されます。プログラム名またはサービス文字列のいずれかがこれらの制限を超える場合は、その前にアスタリスクが付きます。レポートの最後にある別のセクションに、より長いプログラム名またはサービス文字列と、それに対応する ID が表示されます。プログラム名の詳細については、参照してください。Enterprise COBOL for z/OS 言語解説書。

報告書には以下のプログラムは記載されていない：

- IBM Enterprise COBOL for z/OS 4.2 以前でコンパイルされたプログラム
- でコンパイルされたプログラム。LP(64)
- COBOL プログラムを呼び出すか、COBOL プログラムから呼び出される COBOL 以外のプログラム

推奨属性 IGZPROUTddname は次のとおりです。

```
DSORG = PS  RECFM = FB  LRECL = 80
```

IGZPROUT ddname の設定例を以下に示します。

```
//IGZPROUT DD SYSOUT=*
```

または

```
//IGZPROUT DD DSN=dynamic.call.report.dsname,DISP=SHR
```

または

```
//IGZPROUT DD DSN=dynamic.call.report.dsname,  
// UNIT=SYSDA,SPACE=(TRK,(5,10),RLSE),  
// DCB=(DSORG=PS,LRECL=80,RECFM=FB,BLKSIZE=0),  
// DISP=(NEW,CATLG,DELETE)
```

以下は、レポートの出力例です。

```
+++++++ BEG OF COBOL 5 AND LATER RELEASES PROGRAM REPORT V02 ++++++  
=====
```

ID	NAME	VERSION	DATE/TIME COMPILED	SERVICE
000001	PROG1	060400	2024/03/20	
000002	PROG2	060200	2024/03/20 10:56:40	SERVICE 1
000003	PROG3	060400	2024/03/20 10:56:42	*SERVICE ST
000004	*PROG4LON	060300	2024/03/20 10:56:42	
000005	*PROG5THI	060400	2024/03/20 10:56:42	*LONG SERVI
M000006	PROG8	060200	2024/03/20 10:56:42	

```
=====
```

ID	NAME	VERSION	DATE/TIME COMPILED	SERVICE
000007	PROG6	060300	2024/03/20 10:56:40	
000008	PROG9	060400	2024/03/20 10:56:40	*SERVICE ST

```
=====
```

```
=====
= LONG PROGRAM NAME LIST                                     = (5)
=====
ID      PROGRAM NAME
-----
000004  PROG4LONGPROGRAMNAME1
000005  PROG5THISISALONGPROGRAMNAMETHISISALONGPROGRAMNAMETHISISALONGPROGRAMNAM
        ETHISISALONGPROGRAMNAMETHISISALONGPROGRAMNAMETHISISALONGPROGRAMNAMETHI
        SISALONGPROGRAMNAME1

=====
= LONG SERVICE STRING LIST                                   = (6)
=====
ID      SERVICE STRING
-----
000003  SERVICE STRING LONG 1
000005  LONG SERVICE STRING CAN HAVE A MAXIMUM LENGTH OF 64 CHARACTERS!
000008  SERVICE STRING LONG 2

+++++ END OF PROGRAM REPORT +++++
```

(1)

COBOL ACTIVE PROGRAMS セクションはデフォルトで常に表示されている。

(2)

- ID: 各プログラムには、報告書全体を通して簡単に参照できるよう、固有の ID が割り当てられている。この例では、PROG1 に 000001 の ID が割り当てられている。番組の総数は、最大の ID の値を特定することによって決定できる。の値は、キャンセルされた番組があるかどうかによって、COBOL ACTIVE PROGRAMS または CANCELLED PROGRAMS セクションにある可能性がある。
- NAME: NAME 列は COBOL PROGRAM-ID を示し、COBOL ACTIVE PROGRAMS セクションに 8 文字まで表示できる。プログラム名が 8 文字を超えると、名前の前にアスタリスク (*) が表示され、長い名前であることを示す。完全なプログラム名は、対応する ID とともに、LONG PROGRAM NAME LIST セクションに別途記載されている。例えば、プログラム ID 000004 は 8 文字より長い名前を持っている。最初の 8 文字、PROG4LON はアスタリスク付きで表示され、フルネームは LONG PROGRAM NAME LIST セクションに印字される。

注: 動的呼び出しの場合、PROGRAM-ID 段落または ENTRY ステートメントは、プログラムを含むプログラムオブジェクトの対応するプログラムオブジェクト名またはプログラムオブジェクトエイリアスと同一である必要があります。もし、PROGRAM-ID 段落または ENTRY ステートメントが、プログラムを含むプログラムオブジェクトの対応するプログラムオブジェクト (PDSE メンバー) 名またはプログラムオブジェクトエイリアスと同一でない場合、レポートにはプログラム名がリストされます。PROGRAM-ID 段落または ENTRY プログラムオブジェクト (PDSE メンバー) 名またはプログラムオブジェクトエイリアス名をリストするのではなく、ステートメントを使用します。

- VERSION:VV はコンパイラのバージョン番号、RR はコンパイラのリリース番号、MM はコンパイラの修正番号を表します。例えば、コンパイラ・バージョン 6.4.0 は 060400 と表される。
- DATE/TIME COMPILED: コンパイルの日時を示す。
- SERVICE: SERVICE 欄には、各番組に関連する「SERVICE」文字列が表示され、「COBOL ACTIVE PROGRAMS」または「CANCELLED PROGRAMS」欄には最大 10 文字まで表示できる。文字列が 10 文字を超えると、その前にアスタリスク (*) が表示され、長い SERVICE。完全な SERVICE 文字列は、対応する ID とともに LONG SERVICE STRING LIST セクションに別途記載されている。例えば、プログラム ID 000008、名前は PROG8、終了時にキャンセルされた。その SERVICE の文字列は 10 文字より長いので、最初の 10 文字が CANCELLED PROGRAMS セクションに表示され、全文字列が LONG SERVICE STRING LIST セクションに表示される。

(3)

PROG8、ID 値 000006 の前にある文字 'M' がメイン・プログラムを識別する。PROG8 は、他の COBOL プログラムを動的に、静的に、または DLL リンクを介して呼び出す呼び出し側プログラムである。

(4)

CANCELLED PROGRAMS セクションが表示されるのは、動的にコールされ、CANCEL ステートメントを使ってキャンセルされたプログラムがある場合のみである。

(5)

番組名が 8 文字を超えるかどうかによって、LONG PROGRAM NAME LIST セクションがオプションで表示される。[\(2\)の NAME](#) も参照。

LONG PROGRAM NAME LIST、番組名は 1 行最大 70 文字で表示され、次の行に折り返される。例えば、160 文字の名前を持つ番組 ID 000005 の場合、最初の 70 文字は最初の 2 行に表示され、残りの 20 文字は 3 行目に表示される。名前の折り返しを示す特別な文字はない。

(6)

SERVICE の文字列が 10 文字を超えるかどうかによって、LONG SERVICE STRING LIST セクションがオプションで表示される。[\(2\)の SERVICE](#) も参照。

関連参照

[607 ページの『動的呼び出しの作成』](#)

第 3 部 特定の環境に合わせた COBOL プログラムの目標

第 23 章 COBOL プログラムの開発 (CICS の場合)

このタスクについて

CICS 用に作成された COBOL プログラムは、CICS Transaction Server のもとで実行することができます。CICS サービスを使用する CICS COBOL アプリケーション・プログラムは、CICS のコマンド・レベル・インターフェースを使用する必要があります。

CICS コンパイラ・オプションを使用すると、Enterprise COBOL コンパイラは、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 CICS ステートメントの両方を処理します。別の CICS 変換プログラムを使用して CICS ステートメントを COBOL コードに変換すること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。

CICS 下で COBOL プログラムを実行するには、プログラムのコンパイルおよびバインド後に、CICS テーブルの更新など、他のステップをいくつか実行する必要があります。しかし、これらの CICS トピックは、本書の説明範囲を超えています。詳細については、関連タスクを参照してください。

CBLP SHPOP ランタイム・オプションを設定することにより、ランタイム・エラーの処理方法を決定できます。CICS HANDLE および CBLP SHPOP については、「関連タスク」を参照してください。

関連概念

[551 ページの『組み込みの CICS 変換プログラム』](#)

関連タスク

[545 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』](#)

[549 ページの『CICS オプションを使用したコンパイル』](#)

[551 ページの『分離型の CICS 変換プログラムの使用』](#)

[554 ページの『CICS HANDLE を使用したエラー処理』](#)

言語環境プログラミングガイド (CICS での条件処理:

CBLP SHPOP ランタイム・オプションを使用)

[CICS アプリケーションの開発](#)

関連参照

[384 ページの『CICS』](#)

CICS のもとで実行する COBOL プログラムのコーディング

実行するプログラムをコーディングするには CICS、コード CICS コマンドの PROCEDURE DIVISION を使用することにより EXEC CICS コマンド形式。

このタスクについて

```
EXEC CICS command-name command-options  
END-EXEC
```

CICS コマンドの基本形式は上に示したようなものです。EXEC コマンド内では、スペースをワード分離文字として使用してください。コンマやセミコロンは使用しないでください。COBOL 文を以下の範囲に記述しないでください。EXEC CICS コマンド。

制約事項: オブジェクト指向構文を持つ COBOL プログラムは実行できません。ジャワ相互運用性で CICS。また、CICS 下で実行するプログラムを作成する場合は、以下のコードを使用しないでください。

- FILE-CONTROL エントリ ENVIRONMENT DIVISION ただし、FILE-CONTROL エントリは、SORT 声明
- DATA DIVISION の FILE SECTION (FILE SECTION が SORT ステートメントで使用される場合を除く)
- 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する

- USE 宣言部分 (USE FOR DEBUGGING を除く) を使用する
- 次の COBOL 言語ステートメントを使用する
 - ACCEPT 形式 1: データ転送 (システムの日付と時刻を取得するには、形式 2 の ACCEPT を使用できません)
 - CLOSE
 - DELETE
 - DISPLAY UPON CONSOLE
 - DISPLAY UPON SYSPUNCH
 - MERGE
 - OPEN
 - READ
 - RERUN
 - REWRITE
 - START
 - STOP *literal*
 - WRITE

別途 CICS 翻訳者の場合は、REPLACE 文に含まれる EXEC コマンドの後に PROCEDURE DIVISION プログラムのヘッダーを指定しないと、コマンドは翻訳されません。

ファイルの入出力のコーディング: ほとんど入出力処理に CICS コマンドを使用する必要があります。したがって、ファイルを記述したり、OPEN、CLOSE、READ、START、REWRITE、WRITE、または DELETE ステートメントをコーディングすることはありません。かわりに、CICS コマンドを使用してデータの検索、更新、挿入、および削除を行います。

16 MB 境界より上で実行されるように COBOL プログラムをコーディングする: Enterprise COBOL では、16 MB 境界より上で実行されるように COBOL プログラムをコーディングするとき、以下の制約事項が適用されます。

- IMS/ESA® を DBCTL なしで使用しているときは、DL/I CALL ステートメントがサポートされるのは、呼び出して渡されるすべてのデータが 16 MB 境界より下に常駐している場合だけです。このため、DATA(24) コンパイラ・オプションを指定する必要があります。しかし、IMS/ESA を DBCTL と共に使用している場合は、DATA(31) コンパイラ・オプションを使用して、16 MB より上に常駐するデータを渡すことができます。

使用する場合は EXEC DLIDL/I の代わりに CALL ステートメントでは、DATA(31) レベルに関係なく IMS 製品。

- 受け取りプログラムが AMODE 31 でリンク・エディットされている場合、渡されるアドレスは、31 ビットまたは 24 ビットの長さで、左端バイトが 0 に設定されていなければなりません。
- 受け取りプログラムが AMODE 24 でリンク・エディットされている場合、渡されるアドレスは 24 ビットの長さでなければなりません。

データ項目の内容を表示します。 DISPLAY システム論理出力デバイス (SYSOUT、SYSLIST、SYSLST) への出力は、CICS。DISPLAY 出力は、言語環境のメッセージ・ファイル (一時データ・キュー CESE) に書きこまれます。DISPLAY . . . UPON CONSOLE そして DISPLAY . . . UPON SYSPUNCH ただし、許可されません。DISPSIGN オプションを指定して、符号付き数値項目の DISPLAY の出力形式設定を制御することができます。

関連概念

551 ページの『組み込みの CICS 変換プログラム』

関連タスク

262 ページの『CICS のもとでのソート』

547 ページの『CICS のもとでのシステム日付の取得』

[547 ページの『COBOL プログラムとの間の呼び出し』](#)

[549 ページの『ECI 呼び出しの成否の判断』](#)

[551 ページの『分離型の CICS 変換プログラムの使用』](#)

関連参照

[262 ページの『CICS SORT アプリケーションの制約事項』](#)

[395 ページの『DISPSIGN』](#)

CICS のもとでのシステム日付の取得

システム日付を取得するには CICS プログラムを使用するには、format-2ACCEPT 声明または CURRENT-DATE 固有の機能。

このタスクについて

これらのいずれかを使用できます format-2ACCEPT 声明で CICS システムの日付を取得するには:

- ACCEPT *identifier*-2 FROM DATE(2 桁の年)
- ACCEPT *identifier*-2 FROM DATE YYYYMMDD
- ACCEPT *identifier*-2 FROM DAY(2 桁の年)
- ACCEPT *identifier*-2 FROM DAY YYYYDDD
- ACCEPT *identifier*-2 FROM DAY-OF-WEEK (1 桁の整数、1 は月曜日を表す)

これを使えば format-2ACCEPT 声明で CICS システム時間を取得するには:

- 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。

使用しないでください format-1ACCEPT 声明文 CICS プログラム。

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 (*Enterprise COBOL for z/OS* 言語解説書)

COBOL プログラムとの間の呼び出し

電話をかけたり、VS COBOL II、MVS および VM 用 COBOL、COBOL for OS/390 & VM、そして Enterprise COBOL プログラムを使用して CALL 声明。

このタスクについて

分離型の CICS 変換プログラムまたは組み込みの CICS 変換プログラムで処理された、個別にコンパイルされた COBOL プログラムを呼び出す場合は、DFHEIBLK および DFHCOMMAREA を、CALL ステートメントの 1 番目と 2 番目のパラメーターとして渡す必要があります。

分離型の CICS 変換プログラムまたは組み込みの CICS 変換プログラムによって処理される呼び出し先プログラムは、言語に関して CICS がサポートする任意の関数を含むことができます。

動的呼び出し

CICS のもとで実行する場合、COBOL 動的呼び出しを使用できます。COBOL プログラムの場合含む EXEC CICS 声明または含む EXEC SQL 声明では、NODYNAM コンパイラ オプションが必要です。この場合、CALL *identifier* を NODYNAM コンパイラ・オプションと一緒に使用すると、プログラムを動的に呼び出すことができます。

COBOL プログラムの場合含まれていない EXEC CICS 声明と含まれていない EXEC SQL ステートメントでは、コンパイルする必要はありません NODYNAM。この場合、DYNAM コンパイラー・オプションと一緒に CALL *literal* を使用するか、あるいは CALL *identifier* を使用すると、プログラムを動的に呼び出すことができます。

注：END-EXEC は、EXEC CICS ステートメントに関連付けられている場合、(EXEC SQL ステートメントに必要であっても) 後にピリオドを付けることはできません。

動的に呼び出されるプログラムは、CICS 自動インストールを使用していない場合は、CICS プログラム処理テーブル (PPT) に定義されていなければなりません。下 CICS COBOL プログラムは、RELOAD=YES オプションは、CICS PROGRAM 意味。RELOAD=YES で定義されたプログラムに対する動的呼び出しは、ストレージ不足を生じさせる可能性があります。COBOL から動的に呼び出すプログラムには、RELOAD=NO オプションを使用してください。

言語間通信 (ILC)

他の高水準言語との ILC がサポートされます。ILC がサポートされていない場合は、CICS LINK、XCTL、そして RETURN その代わり。

次の表に、COBOL プログラムとアセンブラー言語プログラム間の呼び出し関係を示しています。この表では、「言語環境プログラミングガイド」で解説されているインターフェースに準拠しているアセンブラー言語 プログラムを言語環境準拠 アセンブラー・プログラムと呼んでいます。このインターフェースに準拠していないものは、非言語環境プログラム準拠のアセンブラー・プログラムです。

表 67. CICS のもとでの COBOL およびアセンブラー間の呼び出し		
COBOL プログラムとアセンブラー・プログラム間の呼び出し	言語環境準拠アセンブラー・プログラム	非言語環境準拠アセンブラー・プログラム
Enterprise COBOL プログラムからアセンブラー・プログラムを呼び出せるか	はい	はい
アセンブラー・プログラムから Enterprise COBOL を呼び出せるか	はい	いいえ

ネストされたプログラム

組み込みの CICS 変換プログラムを使用してコンパイルする場合、この変換プログラムにより、GLOBAL 節と一緒に DFHEIBLK および DFHCOMMAREA 制御ブロックが最外部プログラムに生成されます。したがって、ネストされたプログラムをコーディングする場合は、ネストされたプログラムの呼び出しに対する引数として、これらの制御ブロックを渡す必要がありません。

ネストされたプログラムをコーディングし、別の CICS 翻訳者、パス DFHEIBLK そして DFHCOMMAREA ネストされたプログラムへのパラメータとして、EXEC コマンドまたは参照 EXEC インターフェースブロック (EIB)。このようなプログラムとその最上位プログラム間の制御階層の一部を形成するすべてのプログラムに、同じパラメータを渡す必要があります。

COBOL プログラムの呼び出しに関する詳細な情報と例については、CICS、見る [COBOL プログラムからのサブプログラムの呼び出し](#)で CICS アプリケーションの開発。

関連概念

[551 ページの『組み込みの CICS 変換プログラム』](#)

関連タスク

[551 ページの『分離型の CICS 変換プログラムの使用』](#)

[567 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』](#)

[275 ページの『プログラム呼び出し時のエラーの処理』](#)

ILC の下で CICS (言語環境 ライティング 言語間コミュニケーション 応用編)

[外部 CICS インターフェース \(CICS Transaction Server for z/OS 外部インターフェイスガイド\)](#)

関連参照

[398 ページの『DYNAM』](#)

ECI 呼び出しの成否の判断

外部 CICS インターフェース (ECI) の呼び出しの後で、RETURN-CODE 特殊レジスターの内容は予期できない値に設定されます。このため、外部 CICS インターフェースを正常に使用した後、COBOL プログラムが正常に終了した場合でも、ジョブ・ステップが未定義の戻りコードで終了する可能性があります。

このタスクについて

終了時に意味のある戻りコードが戻されるようにするには、プログラムを終了する前に RETURN-CODE 特殊レジスターを設定してください。CICS への最後の呼び出しに関する状況をジョブの戻りコードに反映させるには、外部 CICS インターフェースへの最後の呼び出しで戻された応答コードに基づいて RETURN-CODE 特殊レジスターを設定します。

関連タスク

[CICS での EXCI の使用](#)

CICS オプションを使用したコンパイル

CICS コンパイラー・オプションを使用すると、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することができます。

このタスクについて

NOCICS オプションを指定場合、コンパイラーはソース・プログラムで検出した CICS ステートメントをすべて診断し破棄します。これまで分離型の CICS 変換プログラムを使用していた場合は、NOCICS を指定する必要があります。

CICS オプションは、コンパイラー・オプション・ソース (すなわち、コンパイラー呼び出し、PROCESS または CBL ステートメント、あるいはインストール・デフォルト) のいずれにでも指定することができます。もし、CICS オプションは COBOL インストールのデフォルトなので、指定することはできません。CICS サブオプション。ただし、組み込みの CICS 変換プログラムで行った変更は非 CICS アプリケーションには適切でないので、CICS オプションをインストール・デフォルトとして使用することは推奨しません。

全て CBL または PROCESS ステートメントは、以下の規則に従って、コメント行の前に置かなければなりません。Enterprise COBOL。

COBOL コンパイラーは、CICS コンパイラー・オプションで提供された CICS サブオプション・ストリングを、組み込みの CICS 変換プログラムに渡します。コンパイラーは、サブオプション・ストリングの分析を行いません。

組み込みの CICS 変換プログラムを使用するときは、以下のオプションを指定してコンパイルする必要があります。

表 68. 組み込みの CICS 変換プログラムに必要なコンパイラー・オプション	
コンパイラー・オプション	コメント
CICS	DYNAM、または NORENT を指定すると、コンパイラーは、NODYNAM、および RENT を強制的にオンにします。
NODYNAM	CICS とともに、このオプションを有効にする必要があります。
RENT	CICS とともに、このオプションを有効にする必要があります。

さらに、IBM では、CICS 下でサポートされない言語エレメントにコンパイラーがフラグを立てるように、コンパイラー・オプション WORD(CICS) を使用することを推奨しています。

組み込みの CICS 変換プログラムを使用してプログラムをコンパイルするには、COBOL に付いている標準 JCL プロシージャ・ステートメントを使用できます。上記のコンパイラー・オプションを指定することに加えて、2つの方法で JCL を変更する必要があります。

- COBOL ステップに STEPLIB オーバーライドを指定します。
- 組み込みの CICS 変換プログラム・サービスがリンク・リストに含まれていない場合、それらのサービスを含んでいるデータ・セットを追加します。

CICS Transaction Server 6.1 用のデータ・セットのデフォルト名は `CICSTS61.CICS.SDFHLOAD` ですが、インストール済み環境によってはこの名前が変更されている場合があります。例えば、以下の行が JCL に含まれている場合があります。

```
//STEPLIB DD DSN=CICSTS61.CICS.SDFHLOAD,DISP=SHR
```

COBOL コンパイラー・リストには、組み込みの CICS 変換プログラムが生成するエラー診断 (CICS ステートメントの構文エラーなど) が含まれます。このリストは入力ソースを反映するものであり、組み込みの CICS 変換プログラムが生成する COBOL ステートメントは含まれていません。

一連のプログラムのコンパイル: CICS オプションを使用して、一連の COBOL プログラムが含まれているソース・ファイルをコンパイルする場合、オプションの優先順位 (高い順) は、次のとおりです。

- コンパイル単位を開始する CBL カードまたは PROCESS カードに指定されたオプション
- コンパイラーを開始するときに指定されたオプション
- CICS デフォルト・オプション

関連概念

[551 ページの『組み込みの CICS 変換プログラム』](#)

関連タスク

[545 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』](#)

[550 ページの『CICS サブオプションの分離』](#)

[CICS アプリケーションの開発](#)

関連参照

[384 ページの『CICS』](#)

[376 ページの『矛盾するコンパイラー・オプション』](#)

CICS サブオプションの分離

CICS サブオプションの指定を複数の CBL ステートメントに分割することができます。CICS サブオプションは累積的です。コンパイラーは、複数のソースからのサブオプションを、指定された順に連結します。

このタスクについて

例えば、JCL ファイルに以下のコードが含まれているとします。

```
//STEP1 EXEC IGYWC, . . .
//PARM.COBOL="CICS("FLAG(I)")"
//COBOL.SYSIN DD *
CBL CICS("DEBUG")
CBL CICS("LINKAGE")
IDENTIFICATION DIVISION.
PROGRAM-ID. COBOL1.
```

コンパイル時に、コンパイラーは次の CICS サブオプション・ストリングを組み込みの CICS 変換プログラムに渡します。

```
"FLAG(I) DEBUG LINKAGE"
```

連結ストリングは、グループの前後に 2 つの引用符 (") または 2 つのアポストロフィ (') を付けてシングル・スペースで区切ります。コンパイラーが同じ CICS 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で最後に指定されたサブオプションが有効になります。コンパイラーでは、連結された CICS サブオプション・ストリングの長さは 4 KB に制限されます。

組み込みの CICS 変換プログラム

COBOL プログラムをコンパイルする場合、CICS コンパイラオプションを使用すると、COBOL コンパイラは統合された CICS ネイティブ COBOL と組み込みの両方を扱うトランスレータ CICS ソース プログラム内のステートメント。

コンパイラは、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。間のすべてのテキスト EXEC CICS そして END-EXEC ステートメントは翻訳者に渡されます。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に制御を戻します。

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。

- 使用できますデバッグ・ツール別個の拡張ソースではなく元のソースをデバッグする CICS 翻訳者が生成します。
- 別途翻訳する必要はありません。EXEC CICS または EXEC DLI コピーブックにある文。
- 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データ・セットが不要です。
- 出力リストは 2 つではなく 1 つだけ作成されます。
- ネストされたプログラムを使用する EXEC CICS ステートメントはよりシンプルになります。最外部のプログラムの GLOBAL 属性を使用して DFHCOMMAREA および DFHEIBLK が生成されます。ネストされたプログラムの呼び出し時に引数として渡したり、USING のフレーズ PROCEDURE DIVISION ネストされたプログラムのヘッダー。
- ネストされたプログラムには、EXEC CICS ステートメントを別々のファイルに記述し、ネストされたプログラムを COPY 声明。
- 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・コンパイル (一連のプログラムのコンパイル) でコンパイルすることができます。
- コンパイラはバイナリフィールドを生成するため、CICS フォーマット付き制御ブロック COMP-5 の代わりに BINARY、の設定に依存しません TRUNC コンパイラオプション。CICS プログラムの TRUNC については、アプリケーション・ロジックおよびユーザー定義の 2 進数フィールドの使用に関する要件に従っている限り、任意の設定値を使用することができます。

注: の CICS 文書には、EXCI トランスレータオプションは統合されたプログラムではサポートされません CICS 翻訳者ですが CICS この立場を逆転させた。現在では、EXCI 変換プログラム・オプションを使用してコンパイルを行うことができますし、警告メッセージ DFH7006I は無視してかまいません。

関連概念

[CICS アプリケーションの開発 \(統合 CICS 変換プログラム\)](#)

関連タスク

[545 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』](#)

[549 ページの『CICS オプションを使用したコンパイル』](#)

関連参照

[384 ページの『CICS』](#)

[454 ページの『TRUNC』](#)

分離型の CICS 変換プログラムの使用

CICS のもとで COBOL プログラムを実行するために、分離型の CICS 変換プログラムを使用して CICS コマンドを COBOL ステートメントに変換してから、プログラムをコンパイルし、リンクして実行可能モジュール

ルを作成することができます。しかし、分離型の変換プログラムよりも、Enterprise COBOL の組み込みの CICS 変換プログラムを使用することをお勧めします。分離型 CICS 変換プログラムは、浮動コメント区切り、JSON GENERATE および JSON PARSE、コンパイラー・ディレクティブなど、より新しい COBOL 言語に対して更新されていません。COBOL コンパイラーの最新機能を使用するには、統合 CICS 変換プログラムを使用してください。

このタスクについて

CICS ステートメントを個別に変換するには、COBOL3 変換プログラム・オプションを使用します。このオプションを使用すると、次の行が挿入されます。

```
CBL RENT,NODYNAM,
```

CICS 変換プログラム・オプション NOCBLCARD を使用すれば、CBL ステートメントの挿入を抑止することができます。

分離型の CICS 変換プログラムを使用したときは、プログラムのコンパイル時に以下のコンパイラー・オプションを使用してください。

表 69. 分離型の CICS 変換プログラムに必要なコンパイラー・オプション	
必要なコンパイラー・オプション	条件
RENT	
NODYNAM	プログラムが CICS 変換プログラムによって変換される。

さらに、IBM では、CICS 下でサポートされない言語エレメントにコンパイラーがフラグを立てるように、コンパイラー・オプション WORD(CICS) を使用することを推奨しています。

TRUNC コンパイラー・オプションに関する以下の推奨は、2 進数データ項目の予期値に基づいています。

表 70. 分離型の CICS 変換プログラムに推奨される TRUNC コンパイラー・オプション	
推奨コンパイラー・オプション	条件
TRUNC(OPT)	すべての 2 進数データ項目が 2 進数データ項目についての PICTURE および USAGE 節に準拠している。
TRUNC(BIN)	すべての 2 進数データ項目が 2 進数データ項目についての PICTURE および USAGE 節に準拠しているわけではない。

たとえば、別の CICS 翻訳者であり、データ項目が次のように定義されている PIC S9(8) BINARY8 桁を超える値を受け取る可能性がある場合は、TRUNC(BIN) コンパイラオプションで、項目を USAGE COMP-5、または変更する PICTURE 句。

また、効果のない以下のオプションの使用を避けることもできます。

- ADV
- FASTSRT
- OUTDD

コンパイラーの入力データ・セットは、変換の結果として受け取ったデータ・セットであり、デフォルトでは SYSPUNCH です。

関連概念

[551 ページの『組み込みの CICS 変換プログラム』](#)

関連タスク

[549 ページの『CICS オプションを使用したコンパイル』](#)

CICS 予約語テーブル

COBOL は、CICS アプリケーション・プログラム用に代替予約語テーブル (IGYCCICS) を用意しています。コンパイラ・オプション WORD(CICS) を使用すると、CICS 下ではサポートされない COBOL ワードにフラグが立てられ、エラー・メッセージが出されます。

IBM 提供のデフォルトの予約語テーブルにより制約を受ける COBOL ワード以外に、IBM 提供の CICS 予約語テーブルも、次の COBOL ワードに制約を与えます。

- CLOSE
- DELETE³
- FACTORY
- FD
- FILE¹
- FILE-CONTROL¹
- INPUT-OUTPUT¹
- INVOKE
- I-O-CONTROL
- MERGE
- METHOD
- OBJECT
- OPEN
- READ
- RERUN
- REWRITE
- SD¹、²
- SELF
- START
- SUPER
- WRITE

注：

1. CICS のもとで SORT ステートメントを使用したい場合は (COBOL は CICS のもとで SORT ステートメントのインターフェースをサポートします)、制限付きとマークされたワードのリストからワードを除去するよう CICS 予約語テーブルを変更する必要があります。
2. SORT キーワードに制限はありませんが、SD キーワードには制限があります。これで、フォーマット 2 (テーブル) の SORT ステートメントは使用できますが、フォーマット 1 (ファイル) の SORT ステートメントは使用できなくなります。
3. DELETE キーワードを制限しても、BASIS 処理の DELETE 機能は使用できます。

関連タスク

[549 ページの『CICS オプションを使用したコンパイル』](#)

[262 ページの『CICS のもとでのソート』](#)

関連参照

[461 ページの『WORD』](#)

CICS HANDLE を使用したエラー処理

CBLPSHPOP ランタイム・オプションの設定は、プログラムが CALL ステートメントを使用して COBOL サブプログラムを呼び出すときの HANDLE 指定の状態に影響を与えます。

このタスクについて

CBLPSHPOP が ON の場合は、CALL ステートメントを使用して COBOL サブプログラム (ネストされたプログラムではないもの) が呼び出されると、以下のことが起こります。

手順

1. プログラム初期化の一部として、実行時に、呼び出し側プログラムの HANDLE 指定が延期されます (EXEC CICS PUSH HANDLE を使用して)。
2. 呼び出されたプログラムが独自の HANDLE コマンドを出すまで、HANDLE のデフォルトのアクションが適用されます。
3. プログラム終了の一部として、実行時に、呼び出し側プログラムの HANDLE 指定が回復されます (EXEC CICS POP HANDLE を使用して)。

タスクの結果

を使用する場合 CICS HANDLE CONDITION または CICS HANDLE AID コマンド、LABEL 指定された CICS HANDLE コマンドは同じである必要があります PROCEDURE DIVISION として CICS 分岐させるコマンド CICS HANDLE ラベル。は使用できません CICS HANDLE コマンドは LABELCOBOL で呼び出された別のプログラムによって引き起こされた条件、補助、または異常終了を処理するオプション CALL 声明。LABEL オプションを指定した CICS HANDLE コマンドを使用してプログラム間分岐を実行しようとする、トランザクションが異常終了します。

ネストされたプログラム内で条件、援助機能、または異常終了が発生する場合、その条件、援助機能、または異常終了の LABEL は、同じネストされたプログラム内に存在しなければなりません。そうでない場合は、予測不能な結果を招きます。

パフォーマンスの考慮事項: CBLPSHPOP が OFF であると、実行時に、COBOL サブプログラムへの CALL に対して CICS PUSH または POP は実行されません。サブプログラムがいずれの EXEC CICS 条件処理コマンドも使用しない場合は、CBLPSHPOP(OFF) を実行することで、PUSH HANDLE コマンドおよび POP HANDLE コマンドのオーバーヘッドを除去できます。その結果、CBLPSHPOP(ON) を指定して実行した場合と比較して、パフォーマンスが向上する場合があります。

アプリケーションを VS COBOL II 実行時間言語環境実行時間については、関連資料を参照してください。CBLPSHPOP 追加の考慮事項のオプション。

554 ページの『例: CICS HANDLE を使用したエラー処理』

関連タスク

850 ページの『CICS、IMS、または VSAM での効率的な実行』

関連参照

Enterprise COBOL for z/OS 移行ガイド

Enterprise COBOL for z/OS パフォーマンス・チューニング・ガイド

例: CICS HANDLE を使用したエラー処理

次の例は、COBOL プログラムでの CICS HANDLE の使用を示しています。

プログラム A には CICS HANDLE CONDITION コマンドがありますが、プログラム B には CICS HANDLE コマンドはありません。プログラム A はプログラム B を呼び出します。プログラム A はネストされたプログラム A1 も呼び出します。条件は 3 つのシナリオのいずれかで取り扱われます。

(1)

CBLPSHPOP(ON): プログラム B の CICS READ コマンドによってある条件が引き起こされても、その条件はプログラム A では処理されません (実行時に CICS PUSH HANDLE が実行されたため、HANDLE 指定が中断されたことが原因です) この条件はトランザクション異常終了となります。

(2)

CBLPSHPOP(OFF): プログラム B の CICS READ コマンドによってある条件が引き起こされても、その条件はプログラム A では処理されません (実行時に、LABEL オプションを指定した CICS HANDLE コマンドを使用して、プログラム間分岐の試みを診断します)。この条件はトランザクション異常終了となります。

(3)

ネストされたプログラム A1 の CICS READ コマンドによって条件が引き起こされた場合、制御のフローはラベル ERR-1 に移り、予測できない結果が生じます。

```
*****
*   Program A                               *
*****
ID DIVISION.
PROGRAM-ID. A.

PROCEDURE DIVISION.
    EXEC CICS HANDLE CONDITION
        ERROR(ERR-1)
    END-EXEC.
    CALL 'B' USING DFHEIBLK DFHCOMMAREA.
    CALL 'A1'.
THE-END.
EXEC CICS RETURN END-EXEC.
ERR-1.
Nested program A1.
ID DIVISION.
PROGRAM-ID. A1.
PROCEDURE DIVISION.
    EXEC CICS READ                               (3)
        FILE('LEDGER')
        INTO(RECORD)
        RIDFLD(ACCTNO)
    END-EXEC.
END PROGRAM A1.
END PROGRAM A.
*
*****
*   Program B                               *
*****
ID DIVISION.
PROGRAM-ID. B.

PROCEDURE DIVISION.
    EXEC CICS READ                               (1) (2)
        FILE('MAINFILE')
        INTO(RECORD)
        RIDFLD(ACCTNO)
    END-EXEC.
END PROGRAM B.
```


第 24 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。

このタスクについて

Db2 と通信するには、以下のステップを実行します。

- 必要な SQL 文を記述し、EXEC SQL そして END-EXEC 声明。
- Db2 独立型プリコンパイラーを使用するか、または SQL コンパイラー・オプションを指定してコンパイルし、Db2 コプロセッサを使用します。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[558 ページの『分離型 Db2 プリコンパイラーの使用』](#)

[558 ページの『SQL ステートメントのコーディング』](#)

[561 ページの『SQL オプションを使用したコンパイル』](#)

[567 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』](#)

関連参照

[565 ページの『Db2 のプリコンパイラーとコプロセッサの動作方法の相違』](#)

Db2 コプロセッサ

Db2 コプロセッサ (Db2 では SQL ステートメント・コプロセッサと呼ばれます) を使用すると、コンパイラーは、別個のプリコンパイル・ステップを使用することなく、組み込み SQL ステートメントを含むソース・プログラムを処理します。

Db2 コプロセッサを使用するには、SQL コンパイラー・オプションを指定してください。

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 コプロセッサとインターフェースします。間のすべてのテキスト EXEC SQL そして END-EXEC ステートメントはコプロセッサに渡されます。コプロセッサが、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。

別個のプリコンパイル・ステップの使用も引き続きサポートされますが、コプロセッサの使用をお勧めします。

- コプロセッサを使用すると、デバッグ・ツールによる対話式デバッグは向上します。リスト中の SQL ステートメント (生成された COBOL ソースではない) が表示されるからです。
- COBOL コンパイラー・リストには、Db2 コプロセッサが生成するエラー診断 (SQL ステートメントの構文エラーなど) が含まれます。
- プリコンパイル・ステップ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 コプロセッサを使用するときには適用されません。コプロセッサを使用する場合:
 - ネストされたプログラムのいずれでも、SQL ステートメントを使用することができます。(プリコンパイラーの場合、SQL ステートメントの使用は最外部のプログラムに制限されます。)
 - コピーブックで SQL ステートメントを使用することができます。
 - REPLACE ステートメントは SQL ステートメントに影響を及ぼします。

Db2 コプロセッサを使用してコンパイルすると、オブジェクト・モジュールやリストのような通常の COBOL コンパイラー出力と一緒に、Db2 データベース要求モジュール (DBRM) が生成されます。DBRM

は、COBOL コンパイル・ステップに関して JCL 内の DBRMLIB DD ステートメントで指定されたデータ・セットに書き込みます。DBRM データ・セットには、Db2 バインド・プロセスへの入力として、プログラム内の SQL ステートメントおよびホスト変数に関する情報が入ります。

関連概念

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[558 ページの『分離型 Db2 プリコンパイラーの使用』](#)

[561 ページの『SQL オプションを使用したコンパイル』](#)

関連参照

[565 ページの『Db2 のプリコンパイラーとコプロセッサの動作方法の相違』](#)

[443 ページの『SQL』](#)

分離型 Db2 プリコンパイラーの使用

SQL ステートメントを含む COBOL プログラムを実行するには、分離型 Db2 プリコンパイラーを使用して SQL ステートメントを COBOL ステートメントに変換してから、プログラムをコンパイルおよびリンクして実行可能モジュールを作成します。

このタスクについて

ただし、IBM は分離型 Db2 プリコンパイラーを機能拡張しなくなったため、Enterprise COBOL と統合された Db2 プリコンパイラーを使用することをお勧めします。特に、別の Db2 プリコンパイラは、浮動小数点区切り文字などの新しい COBOL 言語用に更新されていません。JSON GENERATE そして JSON PARSE、およびコンパイラディレクティブ。COBOL コンパイラーの最新機能を使用するには、統合 Db2 コプロセッサを使用してください。

注：プリコンパイラーで使用される Db2 インターフェースは 31 ビットです。プリコンパイラー出力は LP(64) ではコンパイルできません。Db2 プリコンパイル済みプログラムには、LP(32) を使用する必要があります。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

関連参照

[565 ページの『Db2 のプリコンパイラーとコプロセッサの動作方法の相違』](#)

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。の EXEC SQL そして END-EXEC 区切り文字はそれぞれ 1 行に収まる必要があります。複数行にわたって継続させることはできません。EXEC SQL ステートメント内に COBOL ステートメントをコーディングしないでください。

このタスクについて

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。

- コード化 EXEC SQL INCLUDE 含める声明 SQL 通信領域 (SQLCA) は、WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION 最も外側のプログラムの。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。
- 定義する SQL 文で使用するすべてのホスト変数 WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、または LINKAGE SECTION。しかし、それらを EXEC SQL BEGIN DECLARE SECTION そして EXEC SQL END DECLARE SECTION。

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。

関連タスク

[559 ページの『Db2 コプロセッサを用いた SQL INCLUDE の使用』](#)

[559 ページの『SQL ステートメントでの文字データの使用』](#)

[560 ページの『SQL ステートメントでの国別 10 進数データの使用』](#)

[561 ページの『SQL ステートメントでの国別グループ項目の使用』](#)

[561 ページの『SQL ステートメントでのバイナリー項目の使用』](#)

[561 ページの『SQL ステートメントの成否の判断』](#)

[DB2 アプリケーション・プログラミングおよび SQL ガイド \(COBOL アプリケーションでの SQL ステートメントのコーディング\)](#)

関連参照

[563 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』](#)

[DB2 SQL リファレンス](#)

Db2 コプロセッサを用いた SQL INCLUDE の使用

SQL 文 INCLUDE 文はネイティブ COBOL と同様に扱われる COPY 声明を使用すると SQL コンパイラオプション。

このタスクについて

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* の処理は、REPLACING 句を持たない COPY *literal-1* ステートメント内のリテラルと同じ規則に従います。

SQL のライブラリ検索順序 INCLUDE ステートメントは、コンパイラが COBOL を解決するために使用するのと同じ SYSLIB 連結です。COPY ライブラリ名を指定しないステートメント。

関連参照

[481 ページの『第 21 章 コンパイラ指示ステートメント』](#)

[565 ページの『Db2 のプリコンパイラとコプロセッサの動作方法の相違』](#)

[COPY ステートメント \(Enterprise COBOL for z/OS 言語解説書\)](#)

SQL ステートメントでの文字データの使用

次のいずれかをコーディングできます USAGE 使用する文字データのホスト変数を記述する句 EXEC SQL 声明:USAGE DISPLAY シングルバイトまたは UTF-8 データ、USAGE DISPLAY-1 DBCS データの場合、USAGE NATIONAL のために UTF-16 データ、または USAGE UTF-8 のために UTF-8 データ。

このタスクについて

独立型 Db2 プリコンパイラを使用する場合、USAGE NATIONAL で宣言されたホスト変数に関して、EXEC SQL DECLARE ステートメントでコード・ページ (CCSID) を指定する必要があります。宣言されたホスト変数のコードページを指定する必要があります。USAGE DISPLAY または DISPLAY-1 COBOL に有効な CCSID の場合のみ CODEPAGE コンパイラオプションが、Db2 文字およびグラフィックデータ用。

次のコードを見てください。統合 Db2 コプロセッサを使用する場合には (下記の関連概念に詳述されているように、SQLCCSID コンパイラ・オプションを使用して)、強調表示された 2 つのステートメントは不要です。コード・ページ情報は暗黙的に処理されるからです。

```
CBL CODEPAGE(1140) NSYMBOL(NATIONAL)  
WORKING-STORAGE SECTION.
```



```

EXEC SQL INCLUDE SQLCA END-EXEC.
01 INT1 PIC S9(4) USAGE COMP.
01 C1140.
49 C1140-LEN PIC S9(4) USAGE COMP.
49 C1140-TEXT PIC X(50).
EXEC SQL DECLARE :C1140 VARIABLE CCSID 1140 END-EXEC.
01 G1200.
49 G1200-LEN PIC S9(4) USAGE COMP.
49 G1200-TEXT PIC N(50) USAGE NATIONAL.
EXEC SQL DECLARE :G1200 VARIABLE CCSID 1200 END-EXEC.
.
EXEC SQL FETCH C1 INTO :INT1, :C1140, :G1200 END-EXEC.

```

指定する場合 EXEC SQL DECLARE 変数名 VARIABLE CCSID んんんん END-EXEC、その指定は暗黙の CCSID をオーバーライドします。例えば、次のようなコードを使用すると、Db2 は、C1208-TEXT を、COBOL の CODEPAGE コンパイラー・オプションに対して有効な CCSID としてではなく、UTF-8 (CCSID 1208) でエンコードされているものとして扱います。

```

01 C1208.
49 C1208-LEN PIC S9(4) USAGE COMP.
49 C1208-TEXT PIC X(50).
EXEC SQL DECLARE :C1208 VARIABLE CCSID 1208 END-EXEC.

```

NSYMBOL コンパイラー・オプションは、EXEC SQL ステートメントの内部の文字リテラルには影響を及ぼしません。EXEC SQL ステートメントの文字リテラルは、文字定数についての SQL の規則に従います。

関連概念

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[558 ページの『分離型 Db2 プリコンパイラーの使用』](#)

[DB2 アプリケーション・プログラミングおよび SQL ガイド \(COBOL アプリケーションでの SQL ステートメントのコーディング\)](#)

関連参照

[565 ページの『Db2 のプリコンパイラーとコプロセッサの動作方法の相違』](#)

[385 ページの『CODEPAGE』](#)

[DB2 SQL リファレンス](#)

SQL ステートメントでの国別 10 進数データの使用

統合 Db2 コプロセッサまたは Db2 プリコンパイラーのどちらかを使用すると、国別 10 進数ホスト変数を EXEC SQL ステートメントで使用できます。どちらの場合にも CCSID を EXEC SQL DECLARE ステートメントで指定する必要はありません。CCSID 1200 が自動的に使用されます。

このタスクについて

指定する国別 10 進ホスト変数は、EXEC SQL ステートメントには次の特性が必要です。

- 符号付きでなければなりません。
- 指定する必要があるのは SIGN LEADING SEPARATE 句。
- 暗黙的または明示的に USAGE NATIONAL が有効である必要があります。

関連概念

[53 ページの『数値データの形式』](#)

関連タスク

[141 ページの『国別数値データ項目の定義』](#)

関連参照

[565 ページの『Db2 のプリコンパイラーとコプロセッサの動作方法の相違』](#)

SQL ステートメントでの国別グループ項目の使用

国別グループ項目を EXEC SQL ステートメントでホスト変数として使用することができます。国別グループ項目は、基本項目としてではなく、グループ・セマンティクスで(すなわち、そのグループ項目に従属するホスト変数セットの省略表現として)扱われます。

このタスクについて

国別グループ内のすべての従属項目は USAGE NATIONAL ただし、国別グループ項目では可変長文字列を記述できません。

関連タスク

[145 ページの『国別グループの使用』](#)

SQL ステートメントでのバイナリー項目の使用

バイナリデータ項目を EXEC SQL 声明文では、定義するデータ項目は、USAGE COMP-5 または USAGE BINARY、COMP、または COMP-4。

このタスクについて

バイナリー・データ項目を USAGE BINARY、COMP、または COMP-4 として定義する場合は、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ことがあります。)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

[53 ページの『数値データの形式』](#)

関連参照

[454 ページの『TRUNC』](#)

SQL ステートメントの成否の判断

Db2 が SQL ステートメントの実行を終了すると、Db2 は戻りコードを SQLCA 構造体 () に入れて送り、操作が成功したか失敗したかを示します。プログラムでは、戻りコードをテストし、必要なアクションがあればそれを実行します。

このタスクについて

例外が起こるのは、プログラムが DSN のもとで、TSO バッチ・モード・モジュール IKJEFT01 の代替入り口点の 1 つ (IKJEFT1A または IKJEFT1B) から実行された場合です。この場合、戻りコードはレジスター 15 に入れて渡されます。

SQL ステートメントの実行後、RETURN-CODE 特殊レジスターの内容が有効でなくなる場合があります。このため、SQL ステートメントが正常に実行され、COBOL プログラムが正常に終了した場合でも、ジョブ・ステップが未定義の戻りコードで終了することがあります。終了時に意味のある戻りコードが渡されるようにするには、プログラムを終了する前に、RETURN-CODE 特殊レジスターを設定します。

関連タスク

DB2 アプリケーション・プログラミングおよび SQL ガイド (
COBOL アプリケーションでの SQL ステートメントのコーディング)

SQL オプションを使用したコンパイル

SQL コンパイラ・オプションを使用すると、Db2 coprocessor が使用可能になり、Db2 サブオプションを指定できるようになります。

このタスクについて

指定できるのは SQL コンパイラオプションソースのいずれかのオプション：コンパイラ呼び出し、PROCESS または CBL 声明、OPTFILE、またはインストールのデフォルト。SQL オプションが COBOL インストール・デフォルトであるときは、Db2 サブオプションを指定することはできません。ただし、Db2 プロダクトのインストール・デフォルトをカスタマイズすることによって、デフォルトの Db2 サブオプションを指定することができます。

SQL コンパイラ・オプションで指定された Db2 サブオプション・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見るのは、Db2 コプロセッサだけです。

標準の JCL プロシージャ・ステートメントを使用して、プログラムを Db2 コプロセッサでコンパイルすることができます。上記のコンパイラ・オプションを指定する以外に、以下の項目を JCL に指定する必要があります。

- DBRMLIB DD 生成されたデータベース要求モジュール (DBRM) の場所を指定したステートメント。
- Db2 コプロセッサ・サービスが含まれたデータ・セットを追加する COBOL ステップ用の STEPLIB オーバーライド (これらのサービスが LNKLIST に含まれている場合を除く)。通常、このデータ・セットは xxxxxx.SDSNLOAD と呼ばれます。例えば、Db2 11 の場合は *DSNB10.SDSNLOAD* となりますが、ご使用のインストール済み環境でこの名前が変更されている可能性もあります。

例えば、JCL に次の行があるとします。

```
//DBRMLIB DD DSN=PAYROLL.MONTHLY.DBRMLIB.DATA(MAINFILE),DISP=SHR
//STEPLIB DD DSN=DSN910.SDSNLOAD,DISP=SHR
```

バッチのプログラムのコンパイル: 一連の COBOL プログラム (バッチ・コンパイル・シーケンス) が入ったソース・ファイルをコンパイルする際に SQL オプションを使用する場合、SQL は、シーケンスの最初のプログラムのみ有効である必要があります。コンパイラの呼び出しで SQL を指定できますが、このオプションは最初のプログラムにのみ有効になります。バッチの最初以外のプログラムについて CBL または PROCESS ステートメントに SQL を指定すると、コンパイラ診断メッセージを受け取ります。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[562 ページの『Db2 サブオプションの分離』](#)

[567 ページの『DYNAM または NODYNAM コンパイラ・オプションの選択』](#)

関連参照

[398 ページの『DYNAM』](#)

[443 ページの『SQL』](#)

DB2 コマンド解説書

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。

このタスクについて

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。コンパイラは、複数のソースに含まれている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・ファイルに以下のコードが含まれているとします。

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL='SQL("string1")'
//COBOL.SYSIN DD *
CBL SQL("string2")
```

```
CBL SQL("string3")
IDENTIFICATION DIVISION.
PROGRAM-ID. DRIVER1.
```

コンパイル時、コンパイラーは次のサブオプション・ストリングを Db2 コプロセッサに渡します。

```
"string1 string2 string3"
```

連結される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。

COBOL および Db2 CCSID の決定

すべての Db2 ストリング・データ (BLOB、BINARY、および VARBINARY データを除く) には関連するコード化スキームとコード化文字セット ID (CCSID) があります。このことは、固定長および可変長の文字ストリング、固定長および可変長の図形文字ストリング、CLOB ホスト変数、ならびに DBCLOB ホスト変数にもいえます。

組み込み Db2 コプロセッサを使用すると、SQL ステートメント処理で使用されるストリング・ホスト変数に関連付けられるコード・ページ CCSID の決定は、COBOL SQLCCSID オプションの設定、使用されるプログラミング手法、および各種 Db2 構成オプションによって異なります。

IBM COBOL CCSID 値を Db2DSNHDECP 値および/またはプリコンパイラー CCSID オプションの値。

SQL および SQLCCSID COBOL コンパイラー・オプションを使用すると、CODEPAGE コンパイラー・オプションで指定されたり、ホスト変数の COBOL データ・タイプから決定されたりする CCSID 値 *nnnnn* は自動的に COBOL から Db2 に伝えられます。Db2 はホスト変数に COBOL CCSID を関連付け、そうしなければ Db2 の外部メカニズムおよびデフォルトによって暗黙に指定される CCSID をオーバーライドします。この関連付けられた CCSID は、ホスト変数を参照する SQL ステートメントの処理に使用されます。

SQL および NOSQLCCSID コンパイラー・オプションを使用すると、CODEPAGE コンパイラー・オプションで指定される CCSID 値 *nnnnn* は COBOL プログラム内の COBOL ステートメントの処理にのみ使用されます。この CCSID は SQL ステートメントの処理には使用されません。代わって、Db2 は、SQL ステートメントの処理で、Db2 の外部メカニズムおよびデフォルトによって指定される CCSID に従ってホスト変数のデータ値がエンコードされるものと見なします。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

関連タスク

[564 ページの『SQLCCSID または NOSQLCCSID オプションを使用したプログラミング』](#)

関連参照

[563 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』](#)

[385 ページの『CODEPAGE』](#)

[443 ページの『SQL』](#)

[444 ページの『SQLCCSID』](#)

SQL ステートメントのストリング・ホスト変数のコード・ページ決定

統合 Db2 コプロセッサ (SQL コンパイラー・オプション) を使用すると、SQL ステートメントのストリング・ホスト変数を処理するコード・ページは、優先度の高い順から、次に示すように決定されます。

- USAGE NATIONAL を持つホスト変数は、常に Db2 によって CCSID 1200 (Unicode UTF-16) を使用して処理されます。例えば、次のように指定します。

```
01 hostvariable pic n(10) usage national.
```

- USAGE UTF-8 を持つホスト変数は、常に CCSID 1208 (Unicode UTF-8) を使用して Db2 によって処理される。例:

```
01 hostvariable pic u byte-length 10 usage utf-8.
```

- 明示的な FOR BIT DATA 宣言は Db2CCSID 66535 に、変数がエンコードされた文字を表していないことを示します。例えば、次のように指定します。

```
EXEC SQL DECLARE hostvariable VARIABLE FOR BIT DATA END-EXEC
```

- BLOB、BINARY、または VARBINARY ホスト変数には CCSID 関連がありません。これらのストリング・タイプはエンコードされた文字を表しません。
- SQLDA で明示的な CCSID オーバーライドを指定したホスト変数は、その CCSID で処理されます。
- 明示的な CCSID を使用して宣言で指定したホスト変数は、その CCSID で処理されます。例えば、次のように指定します。

```
EXEC SQL DECLARE hostvariable VARIABLE CCSID nnnnn END-EXEC
```

- 英数字ホスト変数は、SQLCCSID コンパイラー・オプションが有効であると、CODEPAGE コンパイラー・オプションからの CCSID nnnnn で処理されます。
- DBCS ホスト変数 (SQLCCSID オプションが有効の場合、マップされた値で処理されますうーん混合 (MBCS) CCSID の純粋な DBCS CCSID コンポーネントである。んんんから CODEPAGE (nnnnn) コンパイラオプション。
- 英数字または DBCS ホスト変数は、NOSQLCCSID オプションが有効であると、Db2 ENCODING バインド・オプション (指定された場合) からの CCSID か、Db2 インストール・パネル DSNTIPF を通じて DSNHDECP で設定された APPLICATION ENCODING からの CCSID で処理されます。

関連参照

[385 ページの『CODEPAGE』](#)

[444 ページの『SQLCCSID』](#)

SQLCCSID または NOSQLCCSID オプションを使用したプログラミング

一般に、統合 Db2 コプロセッサを使用する新しいアプリケーションの場合は、SQLCCSID オプションが推奨されます。また、既存のアプリケーションの場合は長期的な方向性として、このオプションが推奨されます。の NOSQLCCSID このオプションは、既存のプリコンパイラベースのアプリケーションを統合されたプリコンパイラベースのアプリケーションに移行するためのメカニズムとして推奨されます。Db2 コプロセッサ。

このタスクについて

以下の特性のいずれかを備えた COBOL-Db2 アプリケーションの場合は、SQLCCSID オプションが推奨されます。

- COBOL Unicode サポートを使用する
 - CCSID エンコードに間接に依存する他の COBOL 構文 (Java の相互運用性のための XML サポートやオブジェクト指向構文など) を使用する
 - Db2 が前提とするデフォルトの CCSID とは異なる CCSID でエンコードされる文字データを処理する
- の NOSQLCCSID オプションは、Db2 プリコンパイラ。

COBOL の英数字データ項目をホスト変数として使用し、Db2 で定義された文字列データ FOR BIT DATA サブタイプの場合は、次のいずれかを実行する必要があります。

- NOSQLCCSID コンパイラー・オプションを使用する
- それらのホスト変数に対して、明示的な FOR BIT DATA 宣言を指定する。例えば、次のように指定します。

```
EXEC SQL DECLARE hostvariable VARIABLE FOR BIT DATA END-EXEC
```

使用上の注意

- を使用する場合 Db2DCLGEN テーブルの COBOL 宣言を生成するコマンドでは、オプションで FOR BIT DATA 宣言を自動的に行います。これを行うには、DCLGEN コマンドの DCLBIT (YES) オプションを指定します。
- **パフォーマンスに関する考慮事項:** SQLCCSID コンパイラー・オプションを使用すると、SQL 処理にかなりのパフォーマンス・オーバーヘッドがかかることがあります。これは、SQLCCSID が有効であると、デフォルトの Db2 CCSID 関連メカニズムが、ホスト変数単位で機能するメカニズムによってオーバーライドされるからです。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

関連タスク

[558 ページの『分離型 Db2 プリコンパイラーの使用』](#)

関連参照

[444 ページの『SQLCCSID』](#)

Db2 のプリコンパイラーとコプロセッサの動作方法の相違

以下のセクションでは、独立型 COBOL Db2 プリコンパイラーと統合 COBOL Db2 コプロセッサの動作の相違を列挙します。

Db2 のプリコンパイラーおよびコプロセッサ下での CCSID の決定について詳しくは、[563 ページの『COBOL および Db2 CCSID の決定』](#)を参照してください。

EXEC SQL INCLUDE ステートメントの最後のピリオド

プリコンパイラ: Db2 プリコンパイラは、各ピリオドの終わりを要求しない。EXEC SQL INCLUDE 声明。ピリオドが指定されている場合、プリコンパイラーはそのピリオドをステートメントの一部として処理します。ピリオドが指定されていないと、プリコンパイラーはあたかもピリオドが指定されているかのようにステートメントを受け入れます。

コプロセッサ: Db2 コプロセッサはそれぞれ EXEC SQL INCLUDE のような声明 COPY ステートメントであり、ステートメントの最後にピリオドを付ける必要があります。例:

```
IF A = B THEN
    EXEC SQL INCLUDE some_code_here END-EXEC.
ELSE
    .
END-IF
```

IF ステートメントがピリオドで終了していないことに注意してください。

EXEC SQL と REPLACE または COPY REPLACING

プリコンパイラ:とともに Db2 プリコンパイラ、COBOLREPLACE 声明と REPLACING のフレーズ COPY ステートメントは、拡張ソースに基づいて作成されます EXEC SQL 声明。REPLACE および REPLACING の COBOL 規則が使用されます。

コプロセッサ: Db2 コプロセッサを使用すると、REPLACE ステートメントおよび COPY . . . REPLACING ステートメントは、EXEC SQL ステートメントを含め、元のソース・プログラムに作用します。

次の例のように、異なる動作が起こる可能性があります。

```
REPLACE == ABC == By == XYZ ==.  
01  G.  
   02  ABC PIC X(10).  
.  
EXEC SQL SELECT * INTO :G.ABC FROM TABLE1 END-EXEC
```

プリコンパイラでは、G.ABC 次のように表示されます ABC of G 拡張ソースでは、XYZ of G。コプロセッサでは、ABC は元のソース・ストリング G.ABC で分離文字によって区切られていないため、置換は行われません。

END-EXEC ステートメントの後のソース・コード

プリコンパイラ:の Db2 プリコンパイラは、以下のコードを無視します。END-EXEC 同じ行にステートメントがあります。

コプロセッサ:の Db2 コプロセッサは、次のコードを処理する。END-EXEC 同じ行にステートメントがあります。

ホスト変数の複数定義

プリコンパイラ: Db2 プリコンパイラでは、ホスト変数の参照が固有である必要はありません。有効な Db2 データ型にマップされる最初の定義が使用されます。

コプロセッサ: Db2 コプロセッサでは、各ホスト変数参照が固有でなければなりません。コプロセッサは、ホスト変数に対する固有でない参照を診断します。ホスト変数参照を完全修飾して固有なものにするか、[434 ページの『QUALIFY』コンパイラ・オプション](#)を使用する必要があります。

EXEC SQL ステートメントの継続行

プリコンパイラ:の Db2 プリコンパイラでは、EXEC SQL ステートメントは 12 列目から 72 列目から始まります。ステートメントの継続行は、8 から 72 桁目の間であればどこからでも開始できます。

コプロセッサ:の Db2 コプロセッサは、EXEC SQL 継続行を含むステートメントは、12 列目から 72 列目にコーディングする必要があります。

ビット・データ・ホスト変数

プリコンパイラ:とともに Db2 プリコンパイラでは、COBOL の英数字データ項目をホスト変数として使用して、Db2 サブタイプを持つ文字データ FOR BIT DATA。そのホスト変数を FOR BIT DATA として宣言する明示の EXEC SQL DECLARE VARIABLE ステートメントは必要ありません。

コプロセッサ:とともに Db2 コプロセッサでは、COBOL の英数字データ項目をホスト変数として使用して、Db2 サブタイプを持つ文字データ FOR BIT DATA 明示的な EXEC SQL DECLARE VARIABLE そのホスト変数に対するステートメントは COBOL プログラムで指定されます。例:

```
EXEC SQL DECLARE :HV1 VARIABLE FOR BIT DATA END-EXEC.
```

EXEC SQL DECLARE . . . FOR BIT DATA ステートメントを追加する代わりに、NOSQLCCSID コンパイラー・オプションを使用することもできます。詳細については、下記のコード・ページ決定に関する関連参照を参照してください。

SQL-INIT-FLAG

プリコンパイラ:とともに Db2 プリコンパイラでは、プログラムが複数回呼び出されるときに異なるアドレスにある可能性のあるホスト変数を渡す場合、呼び出されたプログラムはリセットする必要があります。SQL-INIT-FLAG。このフラグをリセットするということは、次の SQL ステートメントの実行時に、ストレージの初期設定が必要であることを Db2 に指示することを意味します。このフラグをリセットするには、呼び出し先プログラムの PROCEDURE DIVISION にステートメント MOVE ZERO TO SQL-INIT-FLAG を挿入します。ただし、当該ホスト変数を使用する実行可能な SQL ステートメントより前に挿入する必要があります。

コプロセッサ:とともに Db2 コプロセッサの場合、呼び出されたプログラムはリセットする必要がない SQL-INIT-FLAG。プログラムの移植を容易にするため、プログラム内で SQL-INIT-FLAG が自動的に定義されます。しかし、SQL-INIT-FLAG、のような MOVE ZERO TO SQL-INIT-FLAG、プログラム内の SQL 処理には影響しません。

FILLER データ項目を含む SQL ホスト変数

プリコンパイラ:とともに Db2 プリコンパイラでは、FILLER データ項目を含む SQL ホスト変数にエラーメッセージがフラグ付けされません。

コプロセッサ:とともに Db2 コプロセッサでは、FILLER データ項目を含む SQL ホスト変数に次のエラーメッセージがフラグ付けされます。これは、FILLER データ項目が COBOL 構造で許可されていないためであり、FILLER データ項目を変更する必要があります。

```
IGYPS0294-S SQL host variable reference "data-item" was defined with a contained "FILLER" data item, which is not valid for an SQL host variable. The statement was discarded.
```

参考：[COBOL のホスト構造でプログラミング Db2 for z/OS](#)。

関連概念

[557 ページの『Db2 コプロセッサ』](#)

[563 ページの『COBOL および Db2 CCSID の決定』](#)

関連タスク

[558 ページの『分離型 Db2 プリコンパイラの使用』](#)

関連参照

[563 ページの『SQL ステートメントのストリング・ホスト変数のコード・ページ決定』](#)

[444 ページの『SQLCCSID』](#)

DYNAM または NODYNAM コンパイラー・オプションの選択

COBOL プログラムの場合、EXEC SQL ステートメント、コンパイラオプションの選択 DYNAM または NODYNAM 動作環境によって異なります。

このタスクについて

稼働環境とそれに対応する処置を以下に示します。

- TSO または IMS: DYNAM または NODYNAM のいずれかのコンパイラー・オプションを使用できます。

IMS と Db2 は、言語インターフェース・モジュールの共通別名 DSNHLI を共用していることに注意してください。ライブラリーは次のように連結する必要があります。

- DYNAM オプションを指定して IMS を使用する場合は、IMS ライブラリーを最初に連結してください。

- Db2 のもとでのみアプリケーションを実行する場合、Db2 ライブラリーを最初に連結してください。
- CICS または Db2 呼び出し接続機能 (CAF): NODYNAM コンパイラー・オプションを使用する必要があります。

ストアド・プロシージャーでは CAF を使用するので、COBOL ストアド・プロシージャーも NODYNAM オプションを指定してコンパイルする必要があります。

関連タスク

[561 ページの『SQL オプションを使用したコンパイル』](#)

[DB2 アプリケーション・プログラミングおよび SQL ガイド \(呼び出し接続機能のプログラミング\)](#)

関連参照

[398 ページの『DYNAM』](#)

第 25 章 COBOL プログラムの開発 (IMS の場合)

COBOL プログラムのコーディングは、IMS のもとで実行する場合とほとんど同じですが、以下の推奨事項と制約事項に留意してください。

このタスクについて

COBOL では、IMS メッセージ処理プログラム (MPP) は、IMS 以外の入力ステートメントまたは出力ステートメント (READ、WRITE、REWRITE、OPEN、CLOSE など) を使用しません。

Enterprise COBOL では、次のインターフェースを使用して IMS 機能呼び出すことができます。

- CBLTDLI 呼び出し
- 言語環境呼び出し可能サービス CEETDLI
- EXEC SQLIMS ステートメント

CEETDLI は基本的に CBLTDLI と同じことを行いますが、CEETDLI では LE 条件処理を使用できる点が異なります。IMS 下で CBLTDLI を使用する際には、Language Environment 条件処理を使用できない場合があります。

Java 従属領域でもオブジェクト指向 COBOL プログラムを実行できます。また、単一のアプリケーションで、オブジェクト指向 COBOL と Java 言語を併用できます。

関連概念

[569 ページの『IMS SQL コプロセッサ』](#)

関連タスク

[570 ページの『SQLIMS ステートメントのコーディング』](#)

[571 ページの『SQLIMS オプションを使用したコンパイル』](#)

[573 ページの『IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク』](#)

[573 ページの『IMS のもとでのオブジェクト指向 COBOL と Java の使用』](#)

[574 ページの『IMS のもとにおける Java アプリケーションからの COBOL メソッドの呼び出し』](#)

[575 ページの『COBOL で開始する COBOL と Java の混合アプリケーションの作成』](#)

[575 ページの『混合言語 IMS アプリケーションの作成』](#)

IMS SQL コプロセッサ

IMS SQL コプロセッサ (IMS では SQL ステートメント・コプロセッサと呼ばれる) を使用すると、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 SQLIMS ステートメントを検出すると、IMS SQL コプロセッサとインターフェースします。間のすべてのテキスト EXEC SQLIMS そして END-EXEC ステートメントはコプロセッサに渡されます。コプロセッサが、SQLIMS 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。

注:

- IMS SQL コプロセッサが処理するのは組み込み SQLIMS ステートメントであって、組み込み SQL ステートメントではありません。
- IMS プログラムに含まれる可能性がある EXEC SQL アクセスするためのステートメント Db2SQL データベース、EXEC SQLIMS アクセスするためのステートメント IMSDLI データベース、またはその両方。SQL オプションを使用すれば EXEC SQL ステートメントが使用可能になり、SQLIMS オプションを使用すれば EXEC SQLIMS ステートメントが使用可能になります。

IMS SQL コプロセッサの場合は、以下のようしてステートメントを使用できます。

- ネストされた任意のプログラムで EXEC SQLIMS ステートメントを使用する。
- COPYBOOKS で EXEC SQLIMS ステートメントを使用する。

- REPLACE ステートメントを SQLIMS ステートメントで使用できる。

関連タスク

[570 ページの『SQLIMS ステートメントのコーディング』](#)

[571 ページの『SQLIMS オプションを使用したコンパイル』](#)

[573 ページの『IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク』](#)

関連参照

[445 ページの『SQLIMS』](#)

SQLIMS ステートメントのコーディング

SQLIMS ステートメントは、EXEC SQLIMS と END-EXEC で区切らなければなりません。EXEC SQLIMS 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。COBOL 文を以下の範囲に記述しないでください。EXEC SQLIMS 声明。

このタスクについて

最外部プログラムの WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION に SQLIMS 通信域 (SQLCA) が組み込まれるように EXEC SQLIMS INCLUDE ステートメントをコーディングします。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 SECTION をお勧めします。

制約事項: オブジェクト指向クラスまたはメソッドで SQLIMS ステートメントを使用することはできません。

関連タスク

[570 ページの『IMS SQL コプロセッサで SQLIMS INCLUDE を使用』](#)

[570 ページの『SQLIMS ステートメントでの文字データの使用』](#)

[571 ページの『SQLIMS ステートメントでのバイナリー項目の使用』](#)

[571 ページの『SQLIMS ステートメントの成否の判断』](#)

IMS SQL コプロセッサで SQLIMS INCLUDE を使用

SQLIMS コンパイラー・オプションを使用すると、ネイティブの COBOL COPY ステートメントとまったく同じように SQLIMS INCLUDE ステートメントが扱われます。

このタスクについて

したがって、次の 2 行は同様に扱われます。EXEC SQLIMS INCLUDE ステートメントの終了を示すピリオドが必要です。

```
EXEC SQLIMS INCLUDE name END-EXEC.  
COPY "name".
```

SQLIMS INCLUDE ステートメント内の *name* の処理は、REPLACING 句を持たない COPY *literal-1* ステートメント内のリテラルと同じ規則に従います。

SQLIMS のライブラリ検索順序 INCLUDE ステートメントは、コンパイラが COBOL を解決するために使用するのと同じ SYSLIB 連結です。COPY ライブラリ名を指定しないステートメント。

関連参照

[481 ページの『第 21 章 コンパイラー指示ステートメント』](#)

COPY ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

SQLIMS ステートメントでの文字データの使用

EXEC SQLIMS ステートメントで使用する英数字ホスト・データ項目 (ホスト変数) は USAGE DISPLAY として定義する必要があります。

このタスクについて

注: USAGE DISPLAY-1 または USAGE NATIONAL を使用して定義された文字データ項目は、SQLIMS ホスト変数として使用しないでください。

関連概念

569 ページの『IMS SQL コプロセッサ』

関連参照

385 ページの『CODEPAGE』

SQLIMS ステートメントでのバイナリー項目の使用

バイナリデータ項目を EXEC SQLIMS 声明文では、定義するデータ項目は、USAGE COMP-5 または USAGE BINARY、COMP、または COMP-4。

このタスクについて

バイナリー・データ項目を USAGE BINARY、COMP、または COMP-4 として定義する場合は、TRUNC(BIN) コンパイラー・オプションを使用します。このオプションを使用すると、個々のデータ項目で USAGE COMP-5 を使用する場合よりもパフォーマンスに対する効果が大きくなる可能性があります。代わりにコンパイラー・オプション TRUNC(OPT) または TRUNC(STD) を使用する場合、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、そのデータは無効になることがあります。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

53 ページの『数値データの形式』

関連参照

454 ページの『TRUNC』

SQLIMS ステートメントの成否の判断

IMS が SQLIMS ステートメントの実行を終了すると、IMS は、戻りコードを SQLIMSCA 構造体に入れて送信し、操作が成功したのか失敗したのかを示します。プログラムでは、戻りコードをテストし、必要なアクションがあればそれを実行します。

このタスクについて

SQLIMS ステートメントの実行後、RETURN-CODE 特殊レジスターの内容が有効でなくなる場合があります。このため、SQLIMS ステートメントが正常に使用された後でプログラムが正常に終了した場合でも、ジョブ・ステップが未定義の戻りコードで終了することがあります。意味のある戻りコードが終了時に渡されるようにするには、プログラムを終了する前に RETURN-CODE 特殊レジスターを設定します。

関連タスク

IMS アプリケーション・プログラミング・ガイド

SQLIMS オプションを使用したコンパイル

SQLIMS コンパイラー・オプションを使用すると、IMS SQL コプロセッサを使用可能にし、IMS サブオプションを指定できるようになります。

このタスクについて

SQLIMS オプションは、コンパイラー・オプション・ソース (すなわち、コンパイラー呼び出し、PROCESS または CBL ステートメント、あるいはインストール・デフォルト) のいずれにでも指定することができます。ただし、SQLIMS オプションが COBOL インストールのデフォルトである場合、IMS サブオプションは指定できません。SQLIMS コンパイラー・オプションにおける IMS サブオプション・ストリングは IMS SQL コプロセッサに対してのみ使用可能です。

IMS SQL コプロセッサを使用するには、SQLIMS オプションを使用してコンパイルを行う必要があります、IMS がコンパイル場所のシステム上で使用可能になっていなければなりません。

標準の JCL プロシージャ・ステートメントを使用して、プログラムを IMS SQL コプロセッサでコンパイルすることができます。上記のコンパイラ・オプションを指定する以外に、以下の項目を JCL に指定する必要があります。

Db2 コプロセッサ・サービスが含まれたデータ・セットを追加する COBOL ステップ用の STEPLIB オーバーライド (これらのサービスが LNKLIST に含まれている場合を除く)。通常、このデータ・セットは xxxxxx.SDSNLOAD です。例えば、Db2 11 の場合は DSNB10.SDSNLOAD となりますが、ご使用のインストール済み環境でこの名前が変更されている可能性もあります。

例えば、以下の行が JCL に含まれる場合があります。

```
//STEPLIB DD DSN=IMS.SDFSRESL,DISP=SHR
```

プログラムの一括コンパイル:

一連の COBOL プログラム (バッチのコンパイル・シーケンス) を含むソース・ファイルをコンパイルするときに SQLIMS オプションを使用する場合は、そのシーケンスの最初のプログラムに対してのみ SQLIMS が有効になります。コンパイラの呼び出しで SQLIMS を指定できますが、このオプションは最初のプログラムにのみ有効になります。バッチの最初のプログラム以外のプログラムに対して CBL ステートメントまたは PROCESS ステートメントで SQLIMS を指定すると、コンパイラ診断メッセージが発行されます。

関連概念

[569 ページの『IMS SQL コプロセッサ』](#)

関連タスク

[572 ページの『IMS サブオプションの分離』](#)

関連参照

[443 ページの『SQL』](#)

IMS サブオプションの分離

複数の SQLIMS 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 IMS 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。

このタスクについて

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。コンパイラは、複数のソースに含まれている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・ファイルに以下のコードが含まれているとします。

```
//STEP1 EXEC IGYWC, . . .
//  PARM.COBOL='SQLIMS("string1")'
//COBOL.SYSIN DD *
  CBL SQLIMS("string2")
  CBL SQLIMS("string3")
  IDENTIFICATION DIVISION.
  PROGRAM-ID. DRIVER1.
```

コンパイル時、コンパイラは次のサブオプション・ストリングを IMS SQL コプロセッサに渡します。

```
"string1 string2 string3"
```

連結されるストリングはシングル・スペースで区切られます。コンパイラが同じ SQLIMS 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。コンパイラは、連結 IMS 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。

関連概念

[569 ページの『IMS SQL コプロセッサ』](#)

関連タスク

[571 ページの『SQLIMS オプションを使用したコンパイル』](#)

IMS のもとで実行するための COBOL プログラムのコンパイルおよびリンク

IMS 環境で最高のパフォーマンスを得るためには、RENT コンパイラー・オプションを使用してください。RENT を使用すると、COBOL で再入可能コードが生成されます。その後、アプリケーション・プログラムをプリロード・モード (プログラムが常にストレージにある) または非プリロード・モードのいずれでも実行することができます。別のオプションを使用して再コンパイルする必要はありません。

このタスクについて

プリロードによってパフォーマンスが向上します。これは、プログラムが既にストレージにあると (必要が生じるたびにライブラリーから取り出すよりも) プログラムに対する後続の要求をより速く処理できるためです。

IMS プログラムについては、RENT コンパイラー・オプションの使用が推奨されます。プリロードして実行するプログラム、またはプリロードおよび非プリロードの両方で実行するプログラムには、RENT コンパイラー・オプションを使用する必要があります。プリロードするとプログラムオブジェクト COBOL プログラムを含むファイルの場合、その中のすべての COBOL プログラムはプログラムオブジェクトコンパイルには RENT オプション。

RENT オプションを指定してコンパイルされたプログラムは、z/OS リンク・パック域に入れることができます。そこでは、それらのプログラムを複数の IMS 従属領域で共有できます。

16MB ラインを超えて実行するには、アプリケーションプログラムを RENT。データ IMS アプリケーションプログラムは 16MB ラインより上に配置することができ、DATA(31) RENT 使用するプログラムの場合 IMS サービス。

IMS のもとで COBOL プログラムを正しく実行するには、リンク・エディット属性に関する次のガイドラインに従ってください。

- RENT コンパイラー・オプションを指定してコンパイルされた COBOL プログラムのみを含むプログラム・オブジェクトをリンクするには、RENT としてリンクする。
- COBOL RENT プログラムとその他のプログラムが混在するプログラム・オブジェクトをリンクするために、他のプログラムについて推奨されるリンク・エディット属性を使用する。

関連概念

[40 ページの『ストレージとそのアドレス可能性』](#)

関連タスク

[567 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』](#)
[言語環境プログラミングガイド \(IMS での条件処理\)](#)

関連参照

[391 ページの『データ』](#)

[435 ページの『RENT』](#)

[Enterprise COBOL for z/OS 移行ガイド \(IMS の考慮事項\)](#)

IMS のもとでのオブジェクト指向 COBOL と Java の使用

Java 従属領域で稼働するアプリケーションで、オブジェクト指向 COBOL と Java を併用できます。

このタスクについて

例えば、以下のことを行うことができます。

- Java アプリケーションから COBOL のメソッドを呼び出します。アプリケーションのメッセージ処理部分を Java で作成し、COBOL のメソッドを呼び出して IMS データベースにアクセスします。
- COBOL クラスの main メソッドで開始して Java のルーチンを呼び出す、COBOL と Java の混合アプリケーションを作成します。

このようなアプリケーションは、Java メッセージ処理 (JMP) 従属領域または Java バッチ処理 (JBP) 従属領域で実行する必要があります。メッセージ・キューから読み取るプログラムは、言語に関係なく、JMP 従属領域で実行する必要があります。

関連タスク

[774 ページの『ファクトリー・セクションの定義』](#)

[743 ページの『第 40 章 オブジェクト指向プログラムの作成』](#)

[785 ページの『第 41 章 Java メソッドとの通信』](#)

[321 ページの『第 17 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』](#)

IMS Application Programming Guide

IMS のもとにおける Java アプリケーションからの COBOL メソッドの呼び出し

Enterprise COBOL のオブジェクト指向言語サポートを利用すれば、IMS のもとで Java プログラムから呼び出すことができる COBOL メソッドを作成できます。

このタスクについて

COBOL クラスを定義し、Enterprise COBOL を使用してコンパイルすると、コンパイラーによって、ネイティブ・メソッドを含んだ Java クラス定義およびそれらのネイティブ・メソッドを実装するオブジェクト・コードが生成されます。これらを使用すれば、ほかのクラスを使用する場合とまったく同じように、このクラスのインスタンスを作成し、Java 従属領域で稼働する Java プログラムからクラスのメソッドを呼び出すことができます。

例えば、適切な DL/I 呼び出しを使用して IMS データベースにアクセスする COBOL クラスを定義できます。このクラスの実装を Java プログラムから利用できるようにするためには、以下のステップを実行します。

手順

1. Enterprise COBOL を使用して COBOL クラスをコンパイルします。
コンパイラーによって、クラス定義を含む Java ソース・ファイル (.java) と、ネイティブ・メソッドのインプリメンテーションを含むオブジェクト・モジュール (.o) が生成されます。
2. 生成された Java ソース・ファイルを Java コンパイラーでコンパイルします。
Java コンパイラーによって、クラス・ファイル (.class) が作成されます。
3. オブジェクト・コードをリンクして、z/OS UNIX ファイルシステム (.so) のダイナミック・リンク・ライブラリー (DLL) を生成します。COBOL DLL を含むディレクトリーは、IMS 領域プロシージャの ENVIRON= パラメーターで示されている IMS.PROCLIB メンバーで指定されているように、LIBPATH でリストされなければなりません。
4. メイン JVM オプション・メンバーの共有可能アプリケーション・クラス・パス (IMS 領域プロシージャの JVMOPMAS= パラメーターで指定されている IMS.PROCLIB メンバー内の `ibm.jvm.sharable.application.class.path`) を更新し、JVM が Java クラス・ファイルにアクセスできるようにします。

タスクの結果

Java プログラムからプロシージャ型 COBOL プログラムを直接呼び出すことはできません。既存の COBOL IMS コードを再利用するには、次のいずれかの技法を使用します。

- COBOL のコードを COBOL クラスのメソッドとして再構築します。

- 既存のプロシージャ型コードに対するラッパーとして機能する COBOL のクラス定義とメソッドを記述します。ラッパーのコードでは、COBOL の CALL ステートメントを使用して、プロシージャ型 COBOL プログラムにアクセスできます。

関連タスク

[321 ページの『第 17 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』](#)

[782 ページの『OO アプリケーションの構造化』](#)

[782 ページの『プロシージャ指向 COBOL プログラムのラッピング』](#)

[IMS Application Programming Guide](#)

COBOL で開始する COBOL と Java の混合アプリケーションの作成

アプリケーションはジャワ従属領域は、main クラスのメソッド。

このタスクについて

main ファクトリー・メソッドを持つ COBOL クラス定義は、この要件を満たしています。したがって、IMS のもとでの COBOL と Java の混合アプリケーションの先頭のルーチンとして、main ファクトリー・メソッドを使用できます。

Enterprise COBOL 生成するジャワクラスで main 方法、ジャワ従属領域は検索、インスタンス化、および呼び出すことができます。アプリケーション全体を COBOL でコーディングできますが、通常は、Java ルーチン呼び出すためにこの種のアプリケーションを作成します。COBOL ランタイムは、Java 従属領域の JVM 内で稼働しているときは、この JVM を自動的に検出して使用し、Java クラスのメソッドを呼び出します。

COBOL アプリケーションは、メッセージを処理するために DL/I 呼び出しを使用する必要があります (GU そして GN) とトランザクションの同期 (CHKP)。

関連タスク

[782 ページの『OO アプリケーションの構造化』](#)

[IMS Application Programming Guide](#)

[IBM SDK for Java - Tools Documentation](#)

混合言語 IMS アプリケーションの作成

混合言語 IMS アプリケーションを作成する場合には、STOP RUN ステートメントの影響に注意する必要があります。また、メッセージの処理およびトランザクションの同期、データベースのアクセス、アプリケーション・インターフェース・ブロック (AIB) の使用の方法を理解する必要があります。

このタスクについて

関連タスク

[575 ページの『STOP RUN ステートメントの使用』](#)

[576 ページの『メッセージの処理とトランザクションの同期』](#)

[576 ページの『データベースへのアクセス』](#)

[576 ページの『アプリケーション・インターフェース・ブロックの使用』](#)

STOP RUN ステートメントの使用

アプリケーションの COBOL の部分で STOP RUN ステートメントを使用すると、COBOL と Java のすべてのルーチン (JVM を含む) が終了します。

このタスクについて

制御は即座に IMS に返されます。プログラムとトランザクションは、停止状態のままになります。

メッセージの処理とトランザクションの同期

IMS メッセージ処理アプリケーションは、すべてのメッセージ処理およびトランザクション同期化を、COBOL または Java のいずれかで行う必要があります。このロジックを COBOL と Java の両方の言語で書かれたアプリケーション・コンポーネント間で分散させないでください。

このタスクについて

COBOL コンポーネントの使用 CALLDL/I サービスにメッセージを処理するためのステートメント (GU そして GN) とトランザクションを同期する (CHKP)。ジャワコンポーネントの使用ジャワクラス IMS これらの機能を実行します。IMSFieldMessage から派生したクラスのオブジェクト・インスタンスを使用して、アプリケーションの COBOL コンポーネントと Java コンポーネントの間のすべての IMS のメッセージ通信を行うことができます。

関連タスク

IMS Application Programming Guide

関連参照

IMS Application Programming API Reference

データベースへのアクセス

IMS データベースへのアクセスには、Java または COBOL のどちらかだけを使用することも、両方の言語を併用することもできます。

このタスクについて

制限: EXEC SQL の声明 Db2 データベースアクセスは、COBOL ルーチンではサポートされていません。ジャワ従属地域。

推奨: Java と COBOL の両方から同じデータベース・プログラム連絡ブロック (PCB) にアクセスしないでください。アプリケーションの Java 部分と COBOL 部分は、同じデータベース位置を共用します。アプリケーションのある部分での呼び出しでデータベース位置が変化すると、アプリケーションの別の部分でのデータベース位置に影響があります。(アプリケーションの影響を受ける部分が同じ言語で記述されていても、異なる言語で記述されていても、この問題は発生します。)

混合アプリケーションの Java コンポーネントが SQL SELECT 節を作成し、Java Database Connectivity (JDBC) を使用して、IMS データベースを照会して結果を取り出すものとします。IMS 用の Java クラス・ライブラリーは、IMS に対する適切な要求を構成し、データベースの正しい位置を確立します。その後、セグメント検索索引数 (SSA) を作成し、同じデータベース PCB に対する GU (Get Unique) 要求を IMS に対して発行する COBOL メソッドを呼び出すと、おそらく、その PCB に対するデータベース中の位置がその要求によって変化します。この場合、最初の SQL SELECT 節を使用してさらにレコードを取り出そうとする後続の JDBC 要求は、データベース位置が変化しているため誤りになります。複数の言語から同じ PCB にアクセスする必要がある場合は、言語間呼び出しの後で、データベースのレコードにさらにアクセスする前に、データベース位置を確立し直してください。

関連タスク

IMS Application Programming Guide

アプリケーション・インターフェース・ブロックの使用

Java 従属領域で実行される COBOL アプリケーションは通常、AIB インターフェースを使用する必要があります。これは、Java 従属領域が、PCB アドレスをアプリケーションに提供しないからです。

このタスクについて

AIB インターフェースを使用するには、PCB 名 (PSBGEN の一部として定義されていなければならない) を AIB のリソース名フィールドに設定することで、呼び出しに対して要求された PCB を指定します。(AIB では、プログラム仕様ブロック (PSB) 定義のすべての PCB に名前がある必要があります。) PCB アドレスを直接指定することではなく、アプリケーションは PCB リスト内での PCB の相対的な位置を知る必要はありません。

ん。呼び出しが完了すると、AIB は、アプリケーションが渡した PCB 名に対応する PCB アドレスを戻します。

または、リソース名としてサブ関数 FIND と PCB 名を使用して、IMS INQY 呼び出しを行うことによって、PCB アドレスを取得することができます。この呼び出しでは、PCB のアドレスが返され、このアドレスを COBOL プログラムに渡すことができます。(この方法でも、PCB 名を PSBGEN の一部として定義する必要がありますが、アプリケーションで AIB インターフェースを使用する必要はなくなります。)

577 ページの『例: アプリケーション・インターフェース・ブロックの使用』

関連タスク

IMS Application Programming Guide

例: アプリケーション・インターフェース・ブロックの使用

次の例は、COBOL アプリケーションで AIB インターフェースを使用する方法を示しています。

```
Local-storage section.
  copy AIB.
  .
Linkage section.
01 IOPCB.
   05 logtterm      pic x(08).
   05               pic x(02).
   05 tpstat        pic x(02).
   05 iodate        pic s9(7)   comp-3.
   05 iotime        pic s9(7)   comp-3.
   05               pic x(02).
   05 seqnum        pic x(02).
   05 mod           pic x(08).
Procedure division.
  Move spaces to input-area
  Move spaces to AIB
  Move "DFS AIB" to AIBRID
  Move length of AIB to AIBRLEN
  Move "IOPCB" to AIBRSNM1
  Move length of input-area to AIBOALEN
  Call "CEETDLI" using GU, AIB, input-area
  Set address of IOPCB to AIBRESA1
  If tpstat = spaces
* . . process input message
```


第 26 章 z/OS ユニックスのもとでの COBOL プログラムの実行

COBOL プログラムを z/OS ユニックス 環境で実行するには、Enterprise COBOL または COBOL for OS/390 & VM を使用してプログラムをコンパイルします。これらのプログラムは再入可能でなければなりません。したがって、コンパイラーおよびリンカー・オプション RENT を使用してください。

このタスクについて

プログラムを z/OS UNIX ファイルシステムリンカーオプションを使用する AMODE 31。どれでも AMODE 24 内部から呼び出すプログラム z/OS ユニックスアプリケーションは MVs PDSE。

制約事項: z/OS ユニックスのもとでの実行には、以下の制約事項が適用されます。

- SORT および MERGE ステートメントはサポートされません。
- 事前初期設定用として旧バージョンの COBOL インターフェース (ランタイム・オプション RTEREUS) を使用した場合、再使用可能な環境を確立することはできません。
- NOSTHREAD オプションを指定してコンパイルした COBOL プログラムを複数のスレッドで実行することはできません。2 番目のスレッドで COBOL アプリケーションを開始すると、COBOL 実行時に、ソフトウェア条件を受け取ります。NOSTHREAD を指定してコンパイルした COBOL プログラムは、初期プロセス・スレッド (IPT) か、C または PL/I のルーチンから作成する 1 つの非 IPT 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 COBOL プログラムが THREAD オプションを指定してコンパイルされている場合は、複数のスレッドで COBOL プログラムを実行できます。

デバッグ・ツールを使用すれば、IBM 開発者 z/OS のデバッグ・パースペクティブを使用するなどしてリモート・デバッグ・モードで、または VTAM® 端末を使用してフルスクリーン・モード (MFI) で、z/OS ユニックス プログラムをデバッグできます。

関連タスク

[311 ページの『第 16 章 z/OS ユニックスのもとでのコンパイル』](#)

[324 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行』](#)

[579 ページの『z/OS ユニックス 環境での実行』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

[582 ページの『UNIX/POSIX API の呼び出し』](#)

[584 ページの『z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス』](#)

[言語環境プログラミングガイド](#)

関連参照

[435 ページの『RENT』](#)

z/OS ユニックス 環境での実行

COBOL プログラムは、z/OS ユニックス シェルまたはシェル以外から、z/OS ユニックス 実行環境のいずれかで実行できます。

このタスクについて

- プログラムは、OMVS シェル (OMVS) または ISPF シェル (ISHELL) で実行することができます。

シェル・プロンプトでプログラム名を入力してください。プログラムは、現行ディレクトリーまたは検索パスに入っている必要があります。

プログラムを開始する前に環境変数 _CEE_RUNOPTS を設定することによってのみ、ランタイム・オプションを指定することができます。

カタログ式 MVS データ・セットに常駐するプログラムは、tso ユーティリティを使用して、シェルから実行することができます。例:

```
tso "call 'my.loadlib(myprog)'"
```

ISPF シェルは、stdout および stderr のみを、端末ではなく z/OS ユニックスファイルに送信します。

- シェルの外部から、TSO/E 下で、またはバッチ内でプログラムを実行できます。

z/OS ユニックスファイルに常駐する COBOL プログラムを TSO/E プロンプトから呼び出すには、BPXBATCH ユーティリティを使用するか、REXX exec で spawn() syscall を使用してください。

COBOL プログラムを呼び出すには z/OS ユニックスファイルとともに EXECJCL ステートメントでは、BPXBATCH ユーティリティを使用します。

関連タスク

[324 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

[582 ページの『UNIX/POSIX API の呼び出し』](#)

[584 ページの『z/OS ユニックス下でのメインプログラム・パラメーターへのアクセス』](#)

[191 ページの『QSAM ファイルの定義および割り振り』](#)

[238 ページの『行順次ファイルの割り振り』](#)

[222 ページの『VSAM ファイルの割り振り』](#)

[36 ページの『画面上またはファイル内での値の表示 \(DISPLAY\)』](#)

言語環境プログラミングガイド (POSIX 対応プログラムの実行)

関連参照

[448 ページの『TEST』](#)

UNIX システム・サービス ユーザーズ・ガイド (BPXBATCH ユーティリティ)

言語環境プログラミング・リファレンス

環境変数の設定およびアクセス

環境変数を設定することができます z/OS ユニックス COBOL プログラムはシェルからコマンドを使って export そして set、またはプログラムから。

このタスクについて

プログラムの実行を開始する前にシェルから環境変数を設定およびリセットするのが一般的な手順ですが、実行中にプログラムから環境変数を設定、リセット、およびアクセスすることができます。

BPXBATCH を使用してプログラムを実行している場合は、STDENV DD ステートメントを使用して環境変数を設定することができます。

環境変数を設定されていないかのようにリセットするには、z/OS ユニックスシェルコマンド unset。

COBOL プログラムから環境変数をリセットするには、setenv() 関数を呼び出してください。

すべての環境変数の値を調べるには、export コマンドをパラメーターを付けずに使用します。COBOL プログラムから環境変数の値にアクセスするには、getenv() 関数を呼び出してください。

[582 ページの『例: 環境変数の設定とアクセス』](#)

関連タスク

[579 ページの『z/OS ユニックス 環境での実行』](#)

[581 ページの『実行に影響を与える環境変数の設定』](#)

[584 ページの『z/OS ユニックス下でのメインプログラム・パラメーターへのアクセス』](#)

[324 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションの実行』](#)

[311 ページの『z/OS ユニックスのもとでの環境変数の設定』](#)

関連参照

[581 ページの『ランタイム環境変数』](#)

実行に影響を与える環境変数の設定

環境変数を設定するには z/OS ユニックスシェルから COBOL プログラムを実行するには、`export` または `set` 指示。プログラム内から環境変数を設定するには、POSIX 関数 `setenv()` または `putenv()` を呼び出してください。

このタスクについて

例えば、環境変数 MYFILE を設定するには、次のように指定します。

```
export MYFILE=/usr/mystuff/notes.txt
```

[582 ページの『例: 環境変数の設定とアクセス』](#)

関連タスク

[582 ページの『UNIX/POSIX API の呼び出し』](#)

[311 ページの『z/OS ユニックスのもとでの環境変数の設定』](#)

関連参照

[581 ページの『ランタイム環境変数』](#)

ランタイム環境変数

COBOL プログラムでは、いくつかのランタイム変数に関心があります。

それらのランタイム環境変数を次に示します。

_CEE_ENVFILE

環境変数を読み取るファイルの指定。

_CEE_RUNOPTS

ランタイム・オプションを指定します。

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。

COBJVMINITOPTIONS

COBOL による JVM の初期化時に使用する Java 仮想マシン (JVM) オプションを指定します。

_IGZ_SYSOUT

DISPLAY 出力の宛先を指定します。使用できる値は `stdout` と `stderr` だけです。

LIBPATH

ダイナミック・リンク・ライブラリーのディレクトリー・パスを指定します。

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。

STEPLIB

LNKLST に含まれていないプログラムの場所を指定します。

関連タスク

[37 ページの『システム論理出力装置上でのデータの表示』](#)

関連参照

[XL C/C++ プログラミング・ガイド \(_CEE_ENVFILE\)](#)

言語環境プログラミング・リファレンス

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。文字列を渡す BY VALUE ヌル終端文字列を指すポインタ。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=ABCDEFGH".
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 ABCDEFGH
  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.
```

UNIX/POSIX API の呼び出し

標準の UNIX/POSIX 関数は、CALL *literal* ステートメントを使用して、z/OS ユニックス COBOL プログラムおよび従来の z/OS COBOL プログラムから呼び出すことができます。これらの関数は、言語環境の一部です。

このタスクについて

これらは C 関数なので、引数を渡す必要があります BY VALUE。文字ストリングは、ヌル終了ストリングを指す BY VALUE ポインタとして渡します。これらの関数を呼び出すプログラムをコンパイルするときは、コンパイラ・オプション NODYNAM および PGMNAME (LONGMIXED) を使用する必要があります。

制約事項: は使用できません>>CALLINTERFACE DYNAM これらの API でディレクティブを使用します。

`fork()`、`exec()`、および `spawn()` 関数は、COBOL プログラムから、または COBOL プログラムと同じプロセスに含まれている非 COBOL プログラムから呼び出すことができます。ただし、以下の制約事項に留意しなければなりません。

- `fork` を発行したときに開かれていた COBOL の順次ファイル、索引付きファイル、または相対ファイルに、`fork` されたプロセスからアクセスすることはできません。このようなアクセス (CLOSE、READ、WRITE、REWRITE、DELETE、または START) を試みた場合、ファイル状況コード 92 が返されます。`fork` を出したときにオープンされた行順次ファイルにはアクセスできます。
- 以下の状況の中から当てはまるものがあるプロセスでは、`fork()` 関数を使用できません。

- COBOL の SORT または MERGE が実行されている。
- 宣言が実行されている。
- プロセスに、複数の 言語環境・エンクレーブ (COBOL 実行単位) が含まれている。
- プロセスで、いずれかの COBOL 再使用可能環境インターフェースが使用されている。
- プロセスが実行されたことがある 1 つの VS COBOL II プログラム。
- 1 つの例外を除いて、DD 割り振りは親プロセスから子プロセスへ継承されません。例外はローカル spawn で、この場合は、親プロセスと同じアドレス・スペース内に子プロセスが作成されます。ローカル spawn は、spawn() 関数を呼び出す前に、環境変数 _BPX_ SHAREAS=YES を設定することによって要求します。

exec() 関数と spawn() 関数は、新規 UNIX プロセス内で新規の 言語環境・エンクレーブを開始します。したがって、exec() または spawn() 関数のターゲット・プログラムはメインプログラムであり、プロセス内のすべての COBOL プログラムは、すべてのファイルをクローズした初期状態で開始します。

SIGYSAMP データ・セットには、一部の POSIX ルーチンを呼び出すサンプル・コードが用意されています。

表 71. POSIX 関数呼び出しのサンプル		
目的	サンプルの	使用される関数
一部のファイルおよびディレクトリー・ルーチンの使用法を示す	IGYTFL1	• getcwd() • mkdir() • rmdir() • アクセス ()
iconv ルーチンを使用してデータを変換する方法を示す	IGYTENV	• iconv_open() • iconv() • iconv_close()
exec() ルーチンを使用して、他のプロセス関連ルーチンと一緒に新規プログラムを実行する方法を示す	IGYTEXC、IGYTEXC1	• fork() • getpid() • getppid() • execl() • perror() • wait()
errno 値の入手方法を示す	IGYTERNO、IGYTGETE	• perror() • fopen()
プロセス間通信メッセージ・ルーチンの使用法を示す	IGYTMSQ、IGYTMSQ2	• ftok() • msgget() • msgsnd() • perror() • fopen() • fclose() • msgrcv() • msgctl() • perror()

関連タスク

[579 ページの『z/OS ユニックス 環境での実行』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

[584 ページの『z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス』](#)
言語環境プログラミングガイド

関連参照

XL C/C++ ランタイム・ライブラリー・リファレンス

UNIX システム・サービス・プログラミング: アセンブラー呼び出し可能サービス 解説書

z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス

z/OS ユニックス シェル・コマンド行から、あるいは `exec()` または `spawn()` 関数を使用して COBOL プログラムを実行する場合、パラメーター・リストは参照によって渡される 3 つのパラメーターから構成されます。標準の COBOL コーディングを使用してこれらのパラメーターにアクセスすることができます。

このタスクについて

引数カウント

2 番目および 3 番目のパラメーターで渡されるそれぞれの配列のエレメントの数が含まれている 2 進数のフルワード整数。

引数の長さのリスト

ポインターの配列。配列内の n 番目の記入項目は、引数リスト内の n 番目の記入項目の長さを含むフルワードの 2 進整数のアドレスです。

引数リスト

ポインターの配列。配列内の n 番目の記入項目は、`spawn()` 関数、`exec()` 関数、またはコマンド呼び出しで引数として渡される n 番目の文字ストリングのアドレスです。各文字ストリングは、ヌル終了文字ストリングです。

この配列は空になることはありません。最初の引数は、開始するプロセスと関連付けられたファイルの名前を表す文字ストリングです。

[584 ページの『例: z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス』](#)

関連タスク

[579 ページの『z/OS ユニックス 環境での実行』](#)

[580 ページの『環境変数の設定およびアクセス』](#)

[582 ページの『UNIX/POSIX API の呼び出し』](#)

[637 ページの『z/OS 下でのメインプログラム・パラメーターへのアクセス』](#)

例: z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス

次の例は、参照によって渡される 3 つのパラメーターと、それらにアクセスするために使用可能なコーディングを示しています。

```
Identification division.
Program-id. "EXECED".
*****
* This sample program displays arguments received via exec() *
* function of z/OS ユニックス *
*****
Data division.
Working-storage section.
01 curr-arg-count pic 9(9) binary value zero.
Linkage section.
01 arg-count pic 9(9) binary. (1)
01 arg-length-list. (2)
    05 arg-length-addr pointer occurs 1 to 99999
        depending on curr-arg-count.
01 arg-list. (3)
    05 arg-addr pointer occurs 1 to 99999
        depending on curr-arg-count.
01 arg-length pic 9(9) binary.
01 arg pic X(65536).
```

```

Procedure division using arg-count arg-length-list arg-list.
*****
* Display number of arguments received          *
*****
    Display "Number of arguments received: " arg-count
*****
* Display each argument passed to this program  *
*****
    Perform arg-count times
        Add 1 to curr-arg-count
    * *****
    * * Set address of arg-length to address of current *
    * * argument length and display                    *
    * *****
        Set Address of arg-length
        to arg-length-addr(curr-arg-count)
        Display
        "Length of Arg " curr-arg-count " = " arg-length
    * *****
    * * Set address of arg to address of current argument *
    * * and display                                     *
    * *****
        Set Address of arg to arg-addr(curr-arg-count)
        Display "Arg " curr-arg-count " = " arg (1:arg-length - 1)
    End-Perform
    Display "Display of arguments complete."
    Goback.

```

- (1) このカウントには、2 番目および 3 番目のパラメーターで渡される配列の要素の数が含まれています。
- (2) この配列には、引数リスト内の n 番目の項目の長さへのポインターが含まれています。
- (3) この配列には、`spawn()` 関数、`exec()` 関数、またはコマンド呼び出しで引数として渡される n 番目の文字列へのポインターが含まれています。

第 27 章 AMODE 64 プログラムの開発

一般に、AMODE 64 アプリケーションに関する設計上の考慮事項は AMODE 31 と同じです。いくつかの例外を除き、COBOL 言語機能の構文とセマンティクスに関して、LP(64) と LP(32) に違いはありません。主な違いと考慮事項については、関連タスクを参照してください。

このタスクについて

関連タスク

587 ページの『AMODE 64 プログラムでのポインター・データ項目』

588 ページの『AMODE 64 サポート付きコンパイラー制限』

588 ページの『AMODE 64 サポート付き CALL ステートメント』

588 ページの『コンパイラー・オプションを使用した、AMODE 64 プログラムのコンパイル』

590 ページの『AMODE 64 に関する他の考慮事項』

AMODE 64 プログラムでのポインター・データ項目

LP(64) と LP(32) の基本的な違いの 1 つは、基礎となる生成済みプログラム・コードにおけるアドレス・フィールドのサイズです。一部のデータ項目に対するストレージ割り振りは、LP コンパイラー・オプションの設定に依存します。

このタスクについて

より大きなアドレス空間にアクセスするには、USAGE POINTER、USAGE FUNCTION-POINTER、そして USAGE INDEX8 バイトの場合 LP(64) 有効の場合 4 バイト、LP(32) 有効です。データ項目 USAGE PROCEDURE-POINTER LP コンパイラ オプションの設定に関係なく、8 バイトの基本項目です。

LENGTH OF 特殊レジスタ

LENGTH OF 特殊レジスタには、LP(64) に関する以下の暗黙的な定義があります。

```
PICTURE 9(18) USAGE IS BINARY.
```

対応する LP(32) に関する暗黙的な定義は次のようなものです。

```
PICTURE 9(9) USAGE IS BINARY.
```

これは、当該データ項目を含むグループ項目のサイズに影響する可能性があります。このグループ項目が COBOL コピーブックで定義されている場合、およびそのコピーブックが LP(64) のコンパイルと LP(32) のコンパイルの両方で使用されている場合は、サイズの違いがプログラム・ロジックに影響することがあることを理解しておく必要があります。

USAGE POINTER-32 データ項目

AMODE 64 サポートを導入する際、コンパイラに 'USAGE POINTER-32 という新しいデータ項目が追加される。これは LP(64) コンパイラー・オプションや LP(32) コンパイラー・オプションとともに使用できます。また、これを使用すれば、データ項目が AMODE 64 プログラムで使用されているのか AMODE 31 プログラムで使用されているのかに関係なく、2 GB 境界より下のアドレスを含むポインター・データ項目を定義できます。LP(64) プログラムと LP(32) プログラムの間で同一コピーブックを共用できます。

関連タスク

415 ページの『LP』

FUNCTION-POINTER 句 (*Enterprise COBOL for z/OS* 言語解説書)

INDEX 句 (*Enterprise COBOL for z/OS* 言語解説書)

の (*Enterprise COBOL for z/OS* 言語解説書)

POINTER 句 (*Enterprise COBOL for z/OS* 言語解説書)

AMODE 64 サポート付きコンパイラー制限

通常、LP(64)におけるテーブルおよび基本英数字データ項目の最大サイズは 2,147,483,646 バイトです。詳細については、コンパイラー限界値(Enterprise COBOL for z/OS 言語解説書)を参照してください。

このタスクについて

LOCAL-STORAGE SECTION は、言語環境 スタックに割り振られます。その合計サイズは、言語環境 の設定およびコンパイラーが使用する内部変数の要件によって制限されます。COBOL プログラムで使用できる実際の限度は 2,147,483,646 バイト未満です。

言語環境 STACK64 ランタイム・オプションは、AMODE 64 アプリケーションに対するスタック・ストレージの割り振りを制御します。デフォルト値は STACK64(1M,1M,128M)です。このオプションは、ご使用のプログラムで必要となるスタックの最大サイズを指定する場合に使用します。

WORKING-STORAGE SECTION は、言語環境 ヒープに割り振られます。その合計サイズは、マシンの 64 ビット・ストレージ容量によって制限されます。

関連タスク

415 ページの『LP』

AMODE 64 サポート付き CALL ステートメント

CALLCALL ステートメントは、あるオブジェクト・プログラムからその実行単位内の別のオブジェクト・プログラムに制御を移します。

このタスクについて

既存の構文規則に加えて、LP(64)では以下の追加規則が適用されます。

1. 静的呼び出しと動的呼び出しは、他の AMODE 64 言語環境規格適合プログラムの呼び出しをサポートします。
2. AMODE 64 COBOL プログラムは、非言語環境規格適合プログラムからは呼び出せません。
LOAD の後でサブプログラムのエントリー・ポイントに分岐するアセンブラー・プログラムは機能しません。代わりに、LE マクロ CEEFETCH を使用して AMODE 64 COBOL プログラムをフェッチしたり呼び出したりしてください。
3. パラメーターを渡すときの規則は XPLINK です。

AMODE 64 において言語環境でサポートされる呼び出し規則は XPLINK のみです。XPLINK ランタイム・オプションを明示的に設定する必要はありません。

関連タスク

CALL ステートメント (Enterprise COBOL for z/OS 言語解説書)
415 ページの『LP』

コンパイラー・オプションを使用した、AMODE 64 プログラムのコンパイル

プログラムをコンパイルして AMODE 64 にするには、LP(64)コンパイラー・オプションを使用する必要があります。LP(64)を使用した場合はコンパイル時に一部のコンパイラー・オプション/サブオプションが破棄されることに注意してください。

このタスクについて

LP(64) でコンパイルを行うときは、以下のコンパイラー・オプションに注意してください。

DLL そして RENT

LP(64) 用のコンパイラー生成コードは再入可能であり、DLL サポートに対応しています。 の LP(64) オプションは RENT。 もし NORENT が指定されている場合、コンパイラは警告メッセージとともにこのオプションを破棄します。 いつ LP(64) が有効であれば、DLL そして NODLL オプションは警告なしに無視され、NODLL リストには常に表示されます。 コンパイラーによって作成されたオブジェクト・ファイルは、DLL アプリケーションまたは非 DLL アプリケーションにおいてリンクできます。 DLL アプリケーションにおいてリンクを行うときは、DYNAM(DLL) バインダー・オプションおよび RENT バインダー・オプションを指定します。 非 DLL アプリケーションにおいてリンクを行うときは、DYNAM(NO) オプションおよび RENT オプションを指定します。

436 ページの『RMODE』

LP(64) オプションは RMODE(ANY) を暗黙指定します。 RMODE(24) オプションが指定されている場合、コンパイラーはオプションを破棄して警告メッセージを発行します。 AMODE 64 COBOL プログラムは 2 GB 境界より下でロードされます。

DATA

DATA(24) オプションは、LP(64) が有効になっている場合は破棄されます。 WORKING-STORAGE SECTION にあるデータ項目は 2 GB 境界より上で割り振られます。 LOCAL セクションにあるデータ項目は言語環境・スタック (これも 2 GB 境界より上にある) 上で割り振られます。

AFP そして HGPR

これらの 2 つのオプションはコンパイラーのレジスター使用を制御するものであり、LP(64) が有効なときにはコンパイラーによって破棄されます。 生成されたコードは言語環境での動作が保証されています。 また、生成されたコードは、他の言語環境 AMODE 64 高水準言語を呼び出したりそのような高水準言語から呼び出されたりすることが可能です。 ユーザー・アプリケーションは COBOL コンパイラーのレジスター使用を制御できません。

384 ページの『CICS』, 443 ページの『SQL』, SQLIMS, 444 ページの『SQLCCSID』, 453 ページの『THREAD』

これらのオプションは LP(64) では破棄されます。 また、SQL プリコンパイラーの出力は AMODE 64 と互換性がありません。

ランタイム・オプション

LP(64) では以下の COBOL 専用ランタイム・オプションがサポートされています。

- AIXBLD
- CBLQDA
- DEBUG
- SIMVRD
- UPSI

以下のランタイム・オプションはサポートされていません。

- ALL31

このオプションは AMODE 31 でのみ有効です。

- CBLOPTS
- CBLPSHPOP

このオプションは、CICS の下で OS VS プログラムに対してのみ適用されます。

- MSGFILE

このオプションは AMODE 64 の言語環境ではサポートされません。

- **RTEREUS**

このオプションは、レガシー設計を使用する COBOL ランタイム再使用環境をサポートします。また、このオプションは LP(64) ではサポートされません。再使用可能ランタイム環境を設定するには、LE 事前初期設定環境機能を使用します。

関連参照

[415 ページの『LP』](#)

AMODE 64 に関する他の考慮事項

プログラムをコンパイルして AMODE 64 にするときは、一部のコンパイラー機能がサポートされていないことを理解しておいてください。

このタスクについて

LP(64)を使用してプログラムをコンパイルする場合は、以下の機能に注意してください。

ALTER ステートメント

ALTER ステートメントは、非構造化プログラミング手法の使用を推奨します。EVALUATE ステートメントは同じ機能を提供するため、代わりに使用されるべきです。

GO TO ステートメント

LP(64) ではサポートされません。

オブジェクト指向 **COBOL** ステートメント

LP(64) ではサポートされません。JAVA JNI インターフェースを使用すれば、JAVA プログラムとのインターオペラビリティを直接実行できます。

XML PARSE および **XML GENERATE** ステートメント

LP(64) ではサポートされません。

JSON PARSE および **JSON GENERATE** ステートメント

LP(64) ではサポートされません。

POSIX ランタイム・オプション

LE ランタイム・オプション **POSIX(ON)** および **POSIX(OFF)** がサポートされています。Pure AMODE 64 COBOL 自体に、POSIX オプションに関する要件はありません。

LE をサポートする最新のオープン環境で動作するように設計されたツールと API ライブラリーは、**POSIX(ON)** を必要とする傾向があります。AMODE 64 COBOL アプリケーションを実行する場合は、**POSIX(ON)** オプションを使用することを強くお勧めします。

これは、アプリケーションに他のプログラミング言語のプログラムが含まれている場合、アプリケーションに信号処理が必要な場合、またはアプリケーションが他の API ライブラリーを使用する場合に特に当てはまります。**POSIX** オプションおよびオプション全般について、追加の要件および制約事項がある場合があります。これらのオプションを適切に設定するのは、アプリケーションの責任です。

サブシステム

LP(64)を使用する COBOL コンパイラーは、統合 CICS 変換プログラム、独立 CICS 変換プログラム、IMS SQL コプロセッサ、および Db2 プリコンパイラーをサポートしません。

必須 **APAR**

AMODE 64 COBOL プログラムをコンパイルしたり実行したりするには、以下の APAR が必要となります。以下の APAR に適したレベルの LE PTF がシステムにインストールされていることを確認してください。

- PH10606、PH10740 および PH13196 (z/OS 2.2 言語環境)
- PH07107、PH09543 追加します PH13761 (z/OS 2.3 言語環境)

注：AMODE 64 COBOL アプリケーションは z/OS 2.2 以上でコンパイルできますが、AMODE 64 COBOL アプリケーションを実行するには z/OS 2.3 以上でなければなりません。

関連参照

[415 ページの『LP』](#)

変更の要約 (*Enterprise COBOL for z/OS* 言語解説書)

第 28 章 混合 AMODE 言語環境での COBOL プログラムの開発

このタスクについて

この章では、混合 AMODE 言語環境でのプログラムの開発について説明します。

AMODE 31 の COBOL プログラムと AMODE 64 の Java プログラムとの対話

このトピックでは、AMODE 64 の Java プログラムと対話するために AMODE 31 で実行される COBOL プログラムを作成する方法について説明します。

このタスクについて

注：このトピックは、OO 機能を含む COBOL プログラムには適用されません。例として、INVOKE ステートメント、OBJECT-REFERENCE データ項目、およびクラス定義を含むプログラムがあります。例外は JNIEnvPtr です。これは、OO プログラムで使用される COBOL 特殊レジスターですが、非 OO COBOL プログラムで Java Native Interface (JNI) 呼び出しを行うために使用できます。

AMODE 31 の COBOL プログラムと、AMODE 64 の Java プログラムとの間のアプリケーション呼び出し

このセクションでは、混合 31/64 ビット環境で COBOL/Java のプログラム呼び出しを構成して使用可能にする方法について説明します。

このタスクについて

AMODE 31 の COBOL から開始されたアプリケーションの AMODE 64 の Java メソッドの呼び出し

COBOL から開始され、JNI 呼び出しを行う必要があるアプリケーションの場合、COBOL 特殊レジスター JNIEnvPtr を使用して JNI 環境ポインターを取得する必要があります。この特別なレジスタは JNIEnv へのアドレス可能性を確立するために使用でき、JNINativeInterface コピーブックで定義された構造 JNI.cpy、これは、コピーブック データセット SIGYLIB のメンバー JNI としても提供されます。

COBOL プログラムは Java 仮想マシン (JVM) を開始するので、環境変数 *COBJVMINIT* オプションに必要な JVM 初期化オプションに設定することができます。混合 AMODE JVM サポートを活用するには、JVM オプション *-XX:+Enable3164Interoperability* を指定する必要があります。

例:

```
export COBJVMINITOPTIONS="-XX:+Enable3164Interoperability -Djava.library.path=."
```

また、*CLASSPATH* 環境変数を適切に設定してください。

AMODE 64 の Java から開始されたアプリケーションによる、AMODE 31 の COBOL ネイティブ・メソッドの呼び出し

Java から開始されるアプリケーションの場合、JNI 環境ポインターはパラメーターとして COBOL ネイティブ・プログラムに渡されるため、JNIEnvPtr 特殊レジスターを使用する必要はありません。コーディングの便宜のために、JNIEnvPtr 特殊レジスターは引き続き使用可能です。AMODE 31 でネイティブ関数 (COBOL プログラム) を呼び出すには、JVM オプション *XX:+Enable3164Interoperability* が必要です。

プログラミングに関する考慮事項:

31/64 ビット混合環境では、Java タイプへの参照は、COBOL の PIC 9(18) COMP-5、つまり 8 バイトのデータ項目として宣言する必要があります。OBJECT-REFERENCE タイプは、このデータ項目が AMODE 31 COBOL プログラムで正しくない 4 バイト・サイズにマップされているため、使用できません。これは、JNI タイプ (jobject、jclass、jfieldID、jmethodID、jarray、jarray<type> など) との対話に使用される COBOL データ項目に適用されます。JNI 呼び出しでパラメーターとして使用する場合。

注: EXTERNAL データ項目および EXTERNAL ファイルへのアクセスはシリアライズされません。

混合 AMODE COBOL/Java インターオペラビリティ・アプリケーションの構築

このセクションでは、混合 AMODE COBOL/Java インターオペラビリティ・アプリケーションを構築する方法について説明します。

このタスクについて

COBOL JNI アプリケーションを z/OS UNIX で構築して実行することをお勧めします。

コンパイル

COBOL プログラムをコンパイルするには、z/OS UNIX シェルで cob2 コマンドを使用します。RENT、DLL、および PGMNAME (LONGMIXED) コンパイラー・オプションを指定します。THREAD オプションは、実行単位内で常に 1 つの COBOL プログラムのみがアクティブである場合には、必須ではありません。

COBOL プログラムに COPY ステートメントを使用してファイル JNI.cpy が含まれている場合は、コピーブックの検索順序で、COBOL インストール・ディレクトリーの組み込みサブディレクトリー (通常は /usr/lpp/cobol/include) を指定します。組み込みサブディレクトリーは、cob2 コマンドの -I オプションを使用するか、または SYSLIB 環境変数を設定することによって、指定することができます。

リンク

COBOL プログラムを実行できるよう準備するには、オブジェクト・ファイルを次の 2 つの DLL サイド・ファイルにリンクして、実行可能モジュールを作成します。

- libjvm31.x は、64 ビットの IBM Java Software Development Kit (\$JAVA_HOME/lib/s390x/j9vm) に付属しています。
- igzxjni2.x これは、COBOL インストールディレクトリーの lib サブディレクトリーに提供されています。z/OS ユニックスファイルシステム、通常は /usr/lpp/cobol/lib。この DLL サイド・ファイルは、SCEELIB PDS 内のメンバー IGZXJNI2 としても使用可能です。

サイド・デック igzcjava.x にはリンクしないでください。これは、混合 AMODE 環境ではサポートされていない OO 機能によってのみ使用されます。igzxjni2.x と igzcjava.x の両方を同じプログラム内でリンクすると、ABEND が発生する可能性があります。

注: \$JAVA_HOME は、64 ビット IBM Java Software Development Kit のホーム・ディレクトリーです。

混合 AMODE COBOL/Java インターオペラビリティ・プログラムの実行

このセクションでは、z/OS UNIX のもとでプログラムを実行するステップを示します。

このタスクについて

z/OS UNIX のもとでプログラムを実行するには、以下のように環境変数をセットアップします。

- STEPLIB: 言語環境プログラム データ・セット SCEERUN および SCEERUN2 を、STEPLIB 連結に組み込みます。

例:

```
export STEPLIB=CEE.SCEERUN2:CEE.SCEERUN
```

または、システムにインストールされているデフォルトの言語環境プログラムを使用します。

- Java 環境変数: LIBPATH 環境変数と CLASSPATH 環境変数を設定します。\$JAVAHOME は、インストールされている Java JDK のホーム・ディレクトリーです。

例:

```
export LIBPATH=${$JAVAHOME}/lib/s390x/j9vm:./
export CLASSPATH=.
```

- COBJVMINITOPTIONS: この環境変数を Java JDK 初期化オプションに設定します。混合 AMODE 環境を初期化するには、JVM オプション -XX:+Enable3164Interoperability が必要です。

例:

```
export COBJVMINITOPTIONS="-Djava.library.path=. -XX:+Enable3164Interoperability"
```

混合 AMODE COBOL/Java 相互相互運用性アプリケーションの例

このセクションでは、混合 AMODE COBOL/Java 相互運用性アプリケーションの例を示します。

このタスクについて

AMODE 31 で開始し、AMODE 64 で Java の main メソッドを呼び出す COBOL プログラムの例を以下に示します。

COBOL プログラムの例:

プログラム・テキストがソース・ファイル prog.cbl にある cob2 で、次のプログラムをコンパイルしてリンクします。

```
cob2 -o prog.cbl -I/usr/lpp/cobol/include -q"dll" -q"pgmname(longmixed)" prog.cbl
/usr/lpp/cobol/lib/igzxjni2.x $JAVA_HOME/lib/s390x/j9vm/libjvm31.x
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. "PROG" recursive.

*
ENVIRONMENT DIVISION.

*
DATA DIVISION.
WORKING-STORAGE SECTION.

01 WS-CLASS-NAME PIC X(50) VALUE LOW-VALUES.
01 WS-METHOD-NAME PIC X(50).
01 WS-METHOD-SX PIC X(50).

01 WS-MAIN-METHOD-ID PIC S9(18) COMP-5.
01 WS-CLASS-ID PIC S9(18) COMP-5.
01 WS-NULL PIC S9(18) COMP-5 VALUE ZERO.

01 WS-JNI-ENV USAGE POINTER.

*
LINKAGE SECTION.

COPY JNI.

PROCEDURE DIVISION.
0010-MAIN.

SET ADDRESS OF JNIEnv TO JNIEnvPtr.
SET ADDRESS OF JNINativeInterface TO JNIEnv.

DISPLAY "set class name".
MOVE z"Greetings" TO WS-CLASS-NAME.
CALL "__etoa" USING WS-CLASS-NAME.

* *****
* FIND CLASS
* *****
CALL FindClass USING BY VALUE
JNIEnvPtr
ADDRESS OF WS-CLASS-NAME
RETURNING WS-CLASS-ID .
```

```

IF WS-CLASS-ID = ZERO
    DISPLAY "CANT LOAD CLASS " WS-CLASS-NAME
    GOBACK
END-IF.

* *****
* FIND METHOD (main)
* *****

MOVE z"main" TO WS-METHOD-NAME.
CALL "__etoa" USING WS-METHOD-NAME.

MOVE z"([Ljava/lang/String;)V" TO WS-METHOD-SX.
CALL "__etoa" USING WS-METHOD-SX.

CALL GetStaticMethodId USING BY VALUE
                           JNIEnvPtr
                           WS-CLASS-ID
                           ADDRESS OF WS-METHOD-NAME
                           ADDRESS OF WS-METHOD-SX
                           RETURNING
                           WS-MAIN-METHOD-ID.

* *****
* CALL METHOD (main)
* *****

CALL CallStaticVoidMethod USING BY VALUE
                           JNIEnvPTR
                           WS-CLASS-ID
                           WS-MAIN-METHOD-ID.

GOBACK.
0100-INIT.
EXIT.

```

Java プログラムの例:

静的 main メソッドを使用して、Java Class Hello World インスタンスを作成します。

```

public class Greetings {
    public static void main(String[] args){
        System.out.println("Greetings from AMODE 64");
    }
}

```

その他の考慮事項と制約事項

このトピックでは、混合 AMODE COBOL/Java インターオペラビリティ・プログラムを実行する際の考慮事項および制約事項について説明します。

このタスクについて

AMODE 31 および AMODE 64 の混合プログラムを使用する COBOL アプリケーションに関する考慮事項は、Java プログラムとの対話にも適用されます。AMODE 31 と AMODE 64 の考慮事項と制約事項の間の相互作用については、[598 ページの『その他の考慮事項と制約事項』](#)を参照してください。

Java 9 仮想マシン・リファレンスの詳細については、[64 ビット Java VM を使用した 31 ビットのネイティブ C または C++ コードの使用 \(z/OS のみ\)](#) を参照してください。

AMODE 31 および AMODE 64 の混合プログラムを使用する COBOL アプリケーション

このトピックでは、AMODE 31 および AMODE 64 の混合プログラムを使用する COBOL アプリケーションについて説明します。

AMODE 31 プログラムと AMODE 64 プログラムの間の動的呼び出し

AMODE 64 プログラムと AMODE 31 プログラムの間の呼び出し (サブプログラムが呼び出し側と異なる AMODE を持つ場合) は、動的呼び出しを介してのみ実行できます。呼び出し側は AMODE 31 または 64 です。CALL ステートメントには、標識は必要ありません。COBOL ランタイムの動的呼び出しハンドラーは、それに応じてサブプログラムおよびプロセスの AMODE を決定します。

プログラムは COBOL 6.3 以上でコンパイルする必要があります。CALL ステートメント内の RETURNING 句が使用されない場合、AMODE 31 プログラムは COBOL 5.1 以降のリリースでコンパイルすることができます。非 COBOL プログラムは、混合 AMODE 31/64 COBOL 動的呼び出しではサポートされません。

ユーザー・プログラムは、パラメーターで渡されるデータ項目を、直接的または間接的に、サブプログラムからアクセスできるようにする必要があります。必要に応じて、DATA ディレクティブを使用して保管場所を指定することができます。

パラメーターの受け渡し

両方の CALL ... USING BY REFERENCE および CALL ... USING BY CONTENT がサポートされています。

電話 ... USING BY VALUE パラメーターはサポートされていません。

CALL の場合 ... RETURNING 句が使用されている場合、戻りデータ項目は 24 バイトより大きくすることはできません。

RETURNING 句は、以下のデータ項目をサポートします。

- 英数字
- ゼーン 10 進数
- COMP-3 (パック 10 進数)
- COMP-4
- COMP-5
- COMP (2 進)

注: 2 つの AMODE 設定にまたがってデータ項目を戻すために必要な追加のコピーがあります。データ項目を CALL ... USING BY REFERENCE で受け渡すことにより、要件を回避することができます。この方法を使用すれば、特に大規模なデータ項目の場合にパフォーマンスが向上します。

動的呼び出しのサブプログラム

呼び出し側プログラムをコンパイルする場合、DLL オプションは必要ありません。これは、既存の純粋な AMODE 31 アプリケーションのサブプログラムが LP(64) で再コンパイルされ、呼び出し側プログラムが AMODE 31 に残っている場合に、AMODE の混合使用を容易にするためです。このサブプログラムは、呼び出し側プログラムを DLL コンパイラー・オプションで再コンパイルすることなく、呼び出し側プログラムで呼び出すことができます。

サブプログラムは DLL でなければなりません。

除外: この制限は AMODE 31 サブプログラムでは解除されました。LE APAR の PTF の場合 PH56800 および COBOL ランタイム APARPH59864 インストールされている z/OS2.5 COBOL プログラムが実行される以降のシステムでは、AMODE 31 サブプログラムは DLL または非 DLL のいずれかになります。

DLL サブプログラムを作成するには、以下を実行します。

- DLL の作成時には、コンパイラー・オプション EXPORTALL および DLL、ならびにバインダー・オプション RENT および DYNAM=DLL を指定してください。
- DLL プログラムのプログラム ID は、DLL モジュール名と同じでなければなりません。そうしないと、動的呼び出しを使用して DLL を呼び出すことはできません。

注: LP(64) での DLL の使用を単純化するための作業として、AMODE 64 プログラムを作成する場合には DLL コンパイラー・オプションは必要ありません。これを指定することはできますが、既存の LP(64) コンパイルで削除する必要はありません。コンパイラーは、これを受け入れ、無視します。EXPORTALL コンパイラー・オプションは、依然として必要です。DLL のバインダー・オプション設定は未変更のままです。

外部ファイル

AMODE 設定が異なるプログラムでは、同じ外部ファイルにアクセスすることはできません。

関連タスク

[607 ページの『動的呼び出しの作成』](#)

関連参照

データ (*Enterprise COBOL for z/OS 言語解説書*)

CALL ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

COBOL ランタイム・オプション AMODE3164

混合 AMODE COBOL プログラムを実行する場合、COBOL ランタイム・オプション AMODE3164 を指定します。このオプションは、IGZOPTS DD ステートメントを使用して JCL 内で指定できます。

例:

```
//IGZOPTS DD *  
    AMODE3164  
/*
```

UNIX で実行する場合は z/OS UNIX で実行する場合は、`_IGZ_RUNOPTS` 環境変数を使用してオプションを指定する。例:

```
export _IGZ_RUNOPTS="AMODE3164"
```

COBOL ランタイムオプションの指定方法の詳細については、以下を参照してください。[470 ページの『COBOL ランタイム オプションの指定』](#)。

関連タスク

[597 ページの『AMODE 31 プログラムと AMODE 64 プログラムの間の動的呼び出し』](#)

その他の考慮事項と制約事項

このトピックでは、AMODE 31 および AMODE 64 の考慮事項と制約事項をリストします。

パフォーマンス

AMODE 31 と 64 の間の遷移には余分なオーバーヘッドがあります。パフォーマンスを重視する領域およびグループのプログラムで AMODE を交差させることは避けてください。同じ AMODE に頻繁に互いを呼び出す必要が生じます。

CALL ステートメントでは、可能な限り `CALL ... BY REFERENCE` を使用してください。これには、他の AMODE でパラメーターがアクセスされる場合に処理を減らす必要があります。RETURNING 句を使用する代わりに、データ項目を BY REFERENCE パラメーターとして返します。これにより、呼び出しが戻された場合の追加コピーが回避されます。CALL ...BY VALUE サポートされていません。

マルチスレッド化

THREAD コンパイラー・オプションは、LP (64) でコンパイルされた COBOL プログラム (AMODE 64 で実行) ではサポートされないという既存の制限があります。この制限は、混合 AMODE 31/64 機能が使用されている場合に、LP (32) でコンパイルされた COBOL プログラム (AMODE 31 で実行) に拡張されます。つまり、LE ランタイム環境に任意の LE エンクレーブ内の AMODE 31 プログラムと AMODE 64 プログラムの両方が含まれている場合、1つのスレッドのみに COBOL プログラムを含めることができます。

注: AMODE 31 プログラムと AMODE 64 プログラムは、混合 LE 環境の異なる LE エンクレーブで実行されています。

条件処理

条件処理サポートは、AMODE 31/64 境界を超えて拡張することはありません。条件を処理する場合、言語環境プログラムは、AMODE 31 および AMODE 64 の境界に達するまで、スタック・フレームをトラバースして条件処理ルーチンを検出することはありません。この時点でハンドラーが見つからない場合、条件は未処理として扱われます。条件処理に依存するアプリケーションは、変更が必要になる場合があります。

追加情報

AMODE 31/64 の相互運用について詳しくは、AMODE 31 と AMODE 64 インターオペラビリティのための言語環境プログラムベンダー・インターフェースの [AMODE 31 および AMODE 64 プログラムのインターオペラビリティの概要](#)を参照してください。

関連タスク

[597 ページの『AMODE 31 プログラムと AMODE 64 プログラムの間の動的呼び出し』](#)

第 4 部 複雑なアプリケーションの構造化

第 29 章 サブプログラムの使用

多くのアプリケーションは、一緒にリンクされた別々にコンパイルされた複数のプログラムから構成されます。実行単位 (言語環境用語のエンクレーブと同義の COBOL 用語) には 1 つ以上のオブジェクト・プログラムが含まれており、他の言語環境のメンバー言語で書かれたオブジェクト・プログラムが含まれることもあります。

このタスクについて

言語環境には言語間サポートが用意されており、これを使用すると、Enterprise COBOL プログラムは、言語環境の要件に適合するプログラムとの間で呼び出しを行うことができます。

名前の接頭部に関する注意事項: IBM 製品が使用している接頭部で始まるプログラム名は使用しないでください。そのような接頭部で始まる名前を持つプログラムを使用すると、CALL ステートメントは意図されたプログラムではなく、IBM ライブラリーまたはコンパイラ・ルーチンを呼び出してしまう可能性があります。避けるべき接頭部のリストについては、プログラムの識別についての関連タスクを参照してください。

関連概念

[603 ページの『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

[3 ページの『プログラムの識別』](#)

[604 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』](#)

[605 ページの『別のプログラムへの制御権移動』](#)

[617 ページの『再帰呼び出しの実行』](#)

[617 ページの『オブジェクト指向プログラムとの間での呼び出し』](#)

[617 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』](#)

[620 ページの『プログラムを再入可能にする』](#)

[656 ページの『マルチスレッド化による COBOL 制限の処理』](#)

s

関連参照

言語環境プログラミングガイド (レジスター規則)

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

の中に PROCEDURE DIVISION プログラムは別のプログラム (一般的にはサブプログラム)、呼び出されたプログラム自体が他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。

呼び出された COBOL プログラムは、PROCEDURE DIVISION。

関連タスク

[604 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』](#)

関連参照

言語環境プログラミングガイド

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。

このタスクについて

メインプログラムまたはサブプログラムで3つの終了ステートメントのいずれを使用することもできますが、次の表に示すように、その影響がそれぞれ異なります。

表 72. 終了ステートメントの影響		
終了 ステートメント	メインプログラム	サブプログラム
EXIT PROGRAM	アクションはとられません。	実行単位を終了せずに呼び出し側プログラムに戻ります。呼び出し先プログラムに、次の実行可能なステートメントがない場合は、暗黙の EXIT PROGRAM ステートメントが生成されます。 スレッド化された環境では、そのプログラムがスレッド内の最初の (最も古い) プログラムでない限り、スレッドは終了しません。
STOP RUN	呼び出し側プログラムに戻ります。 ¹ (オペレーティング・システムに戻る可能性があり、アプリケーションは終了します。) STOP RUN は実行単位 (言語環境 用語ではエンクレーブとも呼ばれます) を終了し、実行単位内で動的に呼び出されたすべてのプログラムと、それらのプログラムにリンク・エディットされているすべてのプログラムを削除します。(メインプログラムは削除しません。) スレッド化された環境では、言語環境のエンクレーブ全体 (そのエンクレーブ内で実行中のすべてのスレッドを含む) が終了します。 CICS 環境では、トランザクション内で実行されているすべてのプログラムを含め、トランザクション全体が終了します。	メインプログラムを呼び出したプログラムに直接戻ります。 ¹ (オペレーティング・システムに戻る可能性があり、アプリケーションは終了します。) STOP RUN は実行単位を終了して、実行単位内で動的に呼び出されたプログラムと、それらとリンク・エディットされたすべてのプログラムを削除します。(メインプログラムは削除しません。) スレッド化された環境では、言語環境のエンクレーブ全体 (そのエンクレーブ内で実行中のすべてのスレッドを含む) が終了します。 CICS 環境では、トランザクション内で実行されているすべてのプログラムを含め、トランザクション全体が終了します。

表 72. 終了ステートメントの影響 (続き)

終了ステートメント	メインプログラム	サブプログラム
GOBACK	呼び出し側プログラムに戻ります。¹ (オペレーティング・システムに戻る可能性があり、アプリケーションは終了します。) GOBACK は実行単位を終了して、実行単位内で動的に呼び出されたプログラムと、それらとリンク・エディットされたすべてのプログラムを削除します。(メインプログラムは削除しません。) スレッド化された環境では、スレッドが終了します。²	呼び出し側プログラムに戻ります。 スレッド化された環境では、そのプログラムがスレッド内の最初のプログラムである場合、スレッドが終了します。²
1. メインプログラムが、言語環境のリンケージ規約に従っていない別の言語で書かれたプログラムによって呼び出された場合は、この呼び出し側プログラムに戻ります。 2. スレッドがエンクレーブ内の最初の実行スレッドである場合は、エンクレーブが終了します。		

サブプログラムは通常、EXIT PROGRAM または GOBACK で終了するときに、最後に使用された状態に残されます。サブプログラムが実行単位の中で次に呼び出されると、PERFORMed パラグラフおよびセクションの戻り点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。
- PROGRAM-ID 段落に INITIAL 節を持つプログラム、または INITIAL コンパイラー・オプションでコンパイルされたプログラムは、呼び出されるたびに初期状態になります。
- LOCAL-STORAGE SECTION で定義されたデータ項目は、プログラムが呼び出されるたびに、VALUE 節で指定された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』

言語環境プログラミングガイド (終了中に起こる事象:

スレッド終了)

関連タスク

613 ページの『ネストされた COBOL プログラムの呼び出し』

617 ページの『再帰呼び出しの実行』

別のプログラムへの制御権移動

別のプログラムへの制御権移動を行うのに、幾つかの異なる方法 (静的呼び出し、動的呼び出し、ネストされたプログラムの呼び出し、およびダイナミック・リンク・ライブラリー (DLL) の呼び出し) を使用できます。

このタスクについて

すべての環境 (CICS を含む) において、Enterprise COBOL プログラム相互間での呼び出しだけでなく、Enterprise COBOL プログラムとその他の古いバージョンのコンパイラーでコンパイルされたプログラムとの間で、静的呼び出しおよび動的呼び出しを行うことができます。

古いレベルのプログラムで通話を行う場合の制限については、旧レベルとの相互運用性 IBM COBOL プログラムの中に Enterprise COBOL for z/OS 移行ガイド。

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て、アプリケーションを作成することができます。また、データ項目を不注意で変更しないようにするためには、**PERFORM** プロシーチャーの代わりに、ネストされたプログラムを使用することができます。ネストされたプログラムは、**CALL literal** または **CALL identifier** ステートメントを使用して呼び出します。

ダイナミック・リンク・ライブラリー (DLL) の呼び出しは、COBOL の動的 **CALL** の代替方法であり、オブジェクト指向 COBOL アプリケーション、z/OS ユニックス プログラム、および C/C++ と相互運用するアプリケーションでの使用に適しています。

z/OS では、2 つのプログラム・オブジェクトをリンクすると、論理的には単一のプログラムになり、そのプログラムには 1 次入り口点と代替入り口点があり、その入り口点にはそれぞれ独自の名前があります。サブプログラムを動的に呼び出すのに使われる個々の名前は、システムに認識されていなければなりません。それぞれの名前をバインダー (リンケージエディター) 制御文として **NAME** または **ALIAS** のプログラムオブジェクトサブプログラムが含まれています。

関連概念

[609 ページの『AMODE 切り替え』](#)

[611 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』](#)

[614 ページの『ネストされたプログラム』](#)

関連タスク

[606 ページの『静的呼び出しの作成』](#)

[607 ページの『動的呼び出しの作成』](#)

[611 ページの『静的呼び出しと動的呼び出しの両方の作成』](#)

[613 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

Enterprise COBOL for z/OS 移行ガイド

(古いレベルの IBM COBOL プログラムとのインターオペラビリティー)

静的呼び出しの作成

NODYNAM および NODLL コンパイラー・オプションを使用して コンパイルされたプログラム内で **CALL literal** ステートメントを使用すると、静的呼び出しが行われます。これらのオプションが使用されていると、**CALL literal** の呼び出しはすべて、静的呼び出しとして処理されます。

このタスクについて

静的呼び出しステートメントを使用すると、COBOL プログラムと呼び出されるプログラムはすべて同一のプログラム・オブジェクトの一部になります。呼び出されるプログラムに制御権が移動すると、そのプログラムは既にストレージにあるので、そのプログラムに対する分岐が行われます。それ以後 **CALL** ステートメントを実行すると、呼び出されるプログラムは最後に使われた状態で使えます。ただし、呼び出されるプログラムに **INITIAL** 属性がある場合は除きます。その場合、呼び出し先プログラムおよびその中に直接的または間接的に含まれているそれぞれのプログラムは、実行単位内で呼び出し先プログラムが呼び出されるたびに、その初期状態にされます。

代替入り口点を指定すると、静的 **CALL** ステートメントはいずれかの代替入り口点を使用して、呼び出されるサブプログラムに入ることができます。

[612 ページの『例: 静的および動的 CALL ステートメント』](#)

関連概念

[611 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』](#)

関連タスク

[607 ページの『動的呼び出しの作成』](#)

[611 ページの『静的呼び出しと動的呼び出しの両方の作成』](#)

[617 ページの『オブジェクト指向プログラムとの間での呼び出し』](#)

関連参照

[396 ページの『DLL』](#)

[398 ページの『DYNAM』](#)

CALL ステートメント (Enterprise COBOL for z/OS 言語解説書)

動的呼び出しの作成

DYNAM および NODLL コンパイラー・オプションを使用してコンパイルしたプログラムで *CALL literal* ステートメントを使用した場合、または NODLL コンパイラー・オプションを使用してコンパイルしたプログラムで *CALL identifier* ステートメントを使用した場合には、動的呼び出しが発生します。

このタスクについて

これらの形式の CALL ステートメントでは、呼び出される COBOL サブプログラムは、メインプログラムとはリンク・エディットされません。代わりに、それは独立したプログラム・オブジェクトにリンク・エディットされ、実行時にロードされます。ロードされるのは必要とされたとき (すなわち、呼び出されたとき) のみです。PROGRAM-ID 段落または ENTRY ステートメント内のプログラム名は、プログラムを含んでいるプログラム・オブジェクトの、対応するプログラム・オブジェクト名またはプログラム・オブジェクト別名と同一でなければなりません。メッセージが届く一般的な理由の 1 つは CEE3593I、CEE3541S、CEE3574I、そして IGZ0178S ダイナミックコール中に program-name の中に PROGRAM-ID 段落は、対応するプログラム オブジェクト名またはエイリアスと同一ではありません。

動的に呼び出すサブプログラムはそれぞれ CALL ステートメントは別のプログラムオブジェクトシステムリンク ライブラリまたはユーザーが提供するプライベート ライブラリのメンバーです。その場合、サブプログラムは MVS ロード・ライブラリーに置かれていなければならない、z/OS UNIX ファイルシステムに常駐させることはできません。動的 CALL ステートメントが、ストレージに常駐していないサブプログラムを呼び出すと、そのサブプログラムは、2 次ストレージから、メインプログラムを含む領域または区画にロードされ、そのサブプログラムへの分岐が実行されます。

実行単位内でサブプログラムへの最初の動的呼び出しが行われると、そのサブプログラムの新規コピーが獲得されます。それ以後この同じサブプログラムを (元の呼び出し側または同じ実行単位内の他のサブプログラムによって) 呼び出すと、サブプログラムの同一コピーに最後に使われた状態で分岐されます (このサブプログラムが INITIAL 属性を処理する場合を除きます)。したがって、次の項目のどちらかを再初期化する必要があります。

- 更新済みの GO TO ステートメント
- データ項目

同じ COBOL プログラムを別の実行単位で呼び出す場合、それぞれの実行単位ごとに WORKING-STORAGE の別個のコピーが割り振られます。

制約事項: 次のものを動的呼び出しすることはできません。

- COBOL DLL プログラム
- PGMNAME (LONGMIXED) オプションを指定してコンパイルされた COBOL プログラム (プログラム名が 8 バイト以下ですべて大文字の場合は可能)。
- PGMNAME (LONGUPPER) オプションを指定してコンパイルされた COBOL プログラム (プログラム名が 8 バイト以下の場合は可能)。
- 同じ COBOL プログラム内の複数の入り口点 (その間で CANCEL ステートメントが実行された場合は可能)

[612 ページの『例: 静的および動的 CALL ステートメント』](#)

関連概念

[608 ページの『サブプログラムに関連した動的呼び出しの使用時期』](#)

[611 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』](#)

関連タスク

[608 ページの『サブプログラムの取り消し』](#)

[606 ページの『静的呼び出しの作成』](#)

[611 ページの『静的呼び出しと動的呼び出しの両方の作成』](#)

関連参照

[396 ページの『DLL』](#)

[398 ページの『DYNAM』](#)

ENTRY ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

CALL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

言語環境プログラミング・リファレンス

サブプログラムの取り消し

サブプログラムに対して CANCEL ステートメントを発行すると、そのサブプログラムが占有していたストレージが解放されます。そのサブプログラムの後続の呼び出しは、最初の呼び出しと同じように機能します。当初の呼び出し元以外のプログラムから、サブプログラムの取り消しを行うことができます。

このタスクについて

呼び出されるサブプログラムに複数の入り口点がある場合は、その間で CANCEL ステートメントが実行されない限り、別個の入り口点をそのサブプログラムに対する動的 CALL ステートメントに指定することはできません。

含まれているプログラムが動的に呼び出され、それに対して CANCEL ステートメントが処理されると、そのプログラムは最初に使われた状態になります。しかし、そのプログラムは初期呼び出しでロードされず、プログラムが取り消されてもストレージは解放されません。

[612 ページの『例: 静的および動的 CALL ステートメント』](#)

関連概念

[611 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』](#)

サブプログラムに関連した動的呼び出しの使用時期

サブプログラムに関連して動的呼び出しを使用するかどうかの判断は、プログラム・オブジェクトの場所、サブプログラムの呼び出しの頻度、サブプログラムのサイズ、保守の容易性、未使用状態のサブプログラムを呼び出す必要があるかどうか、AMODE 切り替えが必要かどうか、およびプログラム名がいつ認識されるか、といった要因によって左右されます。

動的に呼び出したいプログラム・オブジェクトは、z/OS UNIX ファイルシステムではなく MVS ロード・ライブラリーに入っている必要があります。

ほんの少しの条件でサブプログラムが呼び出される場合には、動的呼び出しを使用して、必要時にのみサブプログラムを取り込むことができます。

サブプログラムが非常に大きいか、あるいはその数が多い場合、静的呼び出しを使用すると、あまりにも多くの主記憶域を必要とする場合があります。使用する合計ストレージをより少なくするためには、2つのサブプログラムを両方とも静的に呼び出すよりも、あるサブプログラムを呼び出してから取り消し、次に別のサブプログラムを呼び出してそれを取り消す必要があります。

保守を容易にすることに関心がある場合、動的呼び出しは役立ちます。動的に呼び出されたサブプログラムが変更されても、アプリケーションを再度リンク・エディットする必要はありません。

INITIAL 属性を使用してサブプログラムが呼び出されるときに必ず未使用状態にすることができない場合は、動的 CALL と CANCEL ステートメントの組み合わせを使用して未使用状態を設定できます。最初に COBOL プログラムが呼び出したサブプログラムを取り消した場合、次に呼び出したとき、サブプログラムはその未使用の状態に再初期化されます。

CANCEL ステートメントを使用して、非 COBOL プログラムによって動的にロードして分岐したサブプログラムを明示的に取り消しても、そのサブプログラムのストレージを解放したり、そのサブプログラムを削除するための処置は取られません。

例えば、AMODE 24 同じ実行単位内のプログラム Enterprise COBOL31 ビット アドレッシング モードで実行するプログラム。COBOL 動的呼び出し処理には、AMODE 31 プログラムを呼び出す AMODE 24 プログラムの AMODE 切り替え (逆の場合も同じ) が含まれます。この 暗黙の AMODE 切り替えを行うには、言語環境の ランタイム・オプション ALL31(OFF) および STACK(, ,BELOW) を有効にする必要があります。

動的呼び出しが行われると、制御が呼び出し側から言語環境のライブラリー・ルーチンに渡されます。切り替えの実行後、制御権は呼び出されたプログラムに移動します。ライブラリー・ルーチンの保管域は、呼び出し側プログラムの保管域と呼び出されたプログラムの保管域の間に位置付けられます。

呼び出されるプログラム名が実行時まで分からない場合、CALL *identifier* の形式を使用してください。ここで、*identifier* は、実行時に呼び出し先プログラムの名前が入れられるデータ項目です。例えば、プログラム内での条件付き処理に応じて呼び出されるプログラムが異なるような場合には、CALL *identifier* を使用できます。NODYNAM コンパイラー・オプションを使用する場合でも、CALL *identifier* は常に動的です。

[612 ページの『例: 静的および動的 CALL ステートメント』](#)

関連概念

[609 ページの『AMODE 切り替え』](#)

[611 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』](#)

関連タスク

[607 ページの『動的呼び出しの作成』](#)

関連参照

[398 ページの『DYNAM』](#)

CALL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)
言語環境プログラミング・リファレンス

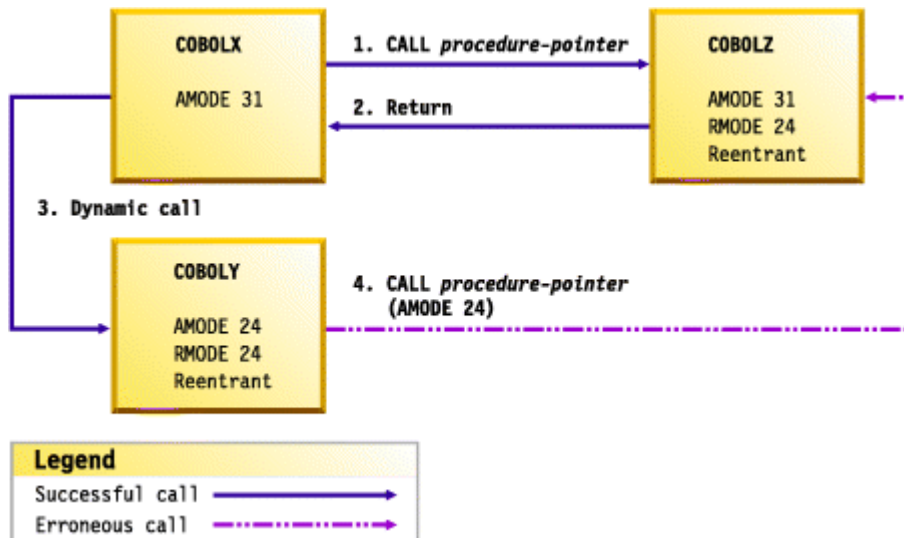
AMODE 切り替え

COBOL サブプログラムを含むアプリケーションの場合、一部の COBOL サブプログラムは AMODE 31 そしていくつかは AMODE 24。この混合 AMODE をサポートするには、コールがダイナミックで、言語環境プログラム ランタイムオプション ALL31(OFF) と STACK(, ,BELOW) が有効でなければならない。

アプリケーションが COBOL プログラムのみで構成され、動的呼び出しを使用する場合、各 COBOL サブプログラムは、常に適切な AMODE になります。たとえば、AMODE 31 COBOL プログラムを AMODE 24 COBOL プログラム、AMODE 自動的に切り替わります。

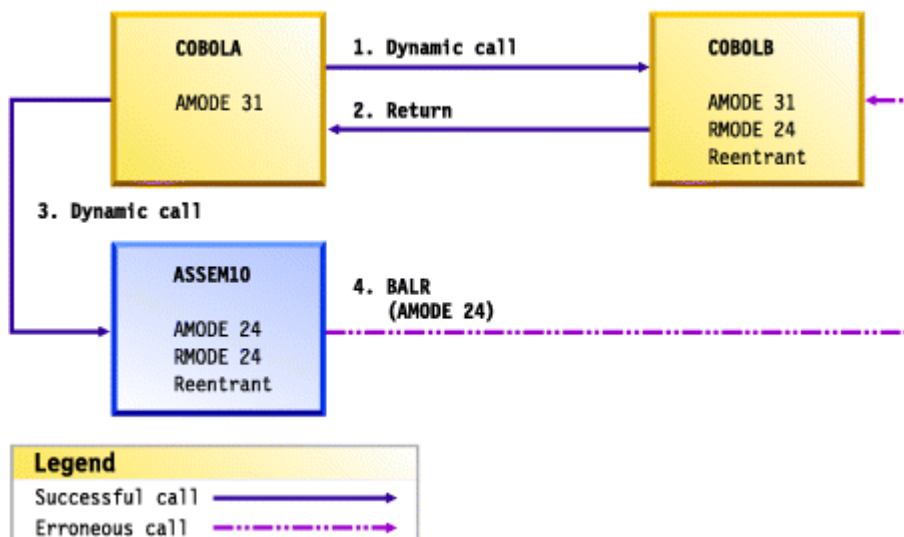
ただし、プロシージャ・ポインター、関数 ポインター、または COBOL サブプログラムを呼び出す他の言語を使用する場合は、COBOL プログラムがエンクレープで複数回呼び出されたときに、呼び出されるたびに同じ AMODE になることを確認する必要があります。この場合、AMODE は自動的に切り替えられません。

次のシナリオは、プロシージャ・ポインターが COBOL サブプログラムの呼び出しに使用される場合に、AMODE の問題が生じる可能性を示しています。このシナリオは、COBOL プログラム COBOLY が呼び出されるときに毎回同じ AMODE にならないため、サポートされません。



1. COBOLX は AMODE 31 です。これは SET ステートメントを使用して、プロシージャー・ポインターを COBOLZ に設定します。COBOLZ は再入可能プログラムオブジェクトそして AMODE 31 そして RMODE 24。COBOLX は、プロシージャー・ポインターを使用して COBOLZ を呼び出します。COBOLZ は AMODE 31 になります。
2. COBOLZ は COBOLX に戻ります。
3. COBOLX は、COBOLY を動的に呼び出して、プロシージャー・ポインターを COBOLZ に渡します。COBOLY は再入可能プログラム・オブジェクトであり、AMODE 24 と RMODE 24 です。COBOLY は AMODE 24 になります。
4. COBOLY はプロシージャー・ポインターを使用して COBOLZ を呼び出します。この呼び出しによって COBOLZ は AMODE 24 になりますが、これは COBOLZ が最初に呼び出されたときの AMODE と同じではありません。

次のシナリオは、COBOL とアセンブラー言語の混合を使用しています。このシナリオは、COBOL プログラム COBOLB が呼び出されるときに毎回同じ AMODE にならないため、サポートされません。



1. COBOLA は AMODE 31 です。COBOLA は COBOLB を動的に呼び出します。COBOLB は再入可能プログラム・オブジェクトであり、AMODE 31 と RMODE 24 です。COBOLB は AMODE 31 になります。
2. COBOLB は COBOLA に戻ります。
3. COBOLA はアセンブラー言語である ASSEM10 を動的に呼び出します。ASSEM10 再入可能プログラムオブジェクト、そして AMODE 24 そして RMODE 24。ASSEM10 入力されている AMODE 24。

4. ASSEM10 は COBOLB をロードします。ASSEM10 は COBOLB に対して BALR 命令を実行します。
COBOLB は AMODE 24 になりますが、COBOLB が最初に呼び出されたときの AMODE と同じではありません。

関連概念

[40 ページの『ストレージとそのアドレス可能性』](#)

[608 ページの『サブプログラムに関連した動的呼び出しの使用時期』](#)

関連タスク

[607 ページの『動的呼び出しの作成』](#)

関連参照

言語環境プログラミング・リファレンス (ALL31)

静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項

静的に呼び出されるプログラムは、呼び出し側プログラムと同じプログラム・オブジェクトにリンク・エディットされるので、静的呼び出しは動的呼び出しより高速です。動的呼び出しのサービスがアプリケーションで必要ない場合は、静的呼び出しが推奨されるメソッドです。

注：同じまたは類似の COBOL バージョンでコンパイルされたプログラム間の静的呼び出しは、異なる COBOL バージョンでコンパイルされたプログラム間の呼び出しよりパフォーマンスが高くなります。例えば、COBOL 6 プログラム間の静的呼び出しは、COBOL 6 プログラムと 4 プログラムの間の静的呼び出しより速くなります。COBOL 6 プログラムと COBOL 4 プログラムなどの混合ケースでは、後方互換性を維持するための余分な処理が必要になるので、静的呼び出しよりも動的呼び出しのほうがパフォーマンスが高くなります。

この説明では、以下の COBOL バージョンを類似のものと見なしています。

- COBOL 4.2 およびそれ以前のリリース
- COBOL 5.1 およびそれ以降のリリース

静的に呼び出されたプログラムは CANCEL を使用して削除できないので、静的呼び出しを使用すると、より多くの主記憶域を使用することになる場合があります。ストレージを重要視する場合には、動的呼び出しの使用を考えてください。呼び出しで使用するストレージは、次の要因によって異なります。

- サブプログラムが 2、3 回しか呼び出されないかどうか。サブプログラムが呼び出されるかどうかに関係なく、静的に呼び出されるプログラムはストレージにロードされます。動的に呼び出されるプログラムは、サブプログラムが呼び出される場合に限りロードされます。
- 後で CANCEL ステートメントを使用して、動的に呼び出されたサブプログラムを取り消すかどうか。

静的に呼び出されたプログラムは削除できませんが、動的に呼び出されたプログラムは削除することができます。動的呼び出しを行った後で CANCEL ステートメントを使用して、動的に呼び出されたプログラムを (呼び出し後ではなく、アプリケーションで不要になった後で) 削除すると、必要なストレージは静的呼び出しを使用する場合よりも少なくて済みます。

関連概念

[608 ページの『サブプログラムに関連した動的呼び出しの使用時期』](#)

関連タスク

[606 ページの『静的呼び出しの作成』](#)

[607 ページの『動的呼び出しの作成』](#)

静的呼び出しと動的呼び出しの両方の作成

NODYNAM コンパイラー・オプションを使用してコンパイルされたプログラムでは、静的と動的の両方の CALL ステートメントを同一のプログラムに使用することができます。

このタスクについて

この場合、CALL *literal* ステートメントを使用すると、呼び出されるサブプログラムはメインプログラムとリンク・エディットされ、1つのプログラム・オブジェクトになります。CALL *identifier* ステートメントを実行すると、別個のプログラム・オブジェクトが動的に呼び出されます。

1つのプログラム内から同一のサブプログラムに対して動的 CALL ステートメントと静的 CALL ステートメントが出されると、そのサブプログラムの2つ目のコピーがストレージにロードされます。この構造では、サブプログラムが最後に使われた状態のままになるとは限らないので、結果が予測できなくなることがあります。

関連参照

[398 ページの『DYNAM』](#)

例: 静的および動的 CALL ステートメント

この例は、静的呼び出しおよび動的呼び出しのコーディング方法を示しています。

この例は次の3つの部分から構成されます。

- 静的呼び出しを使用してサブプログラムを呼び出すコード
- 動的呼び出しを使用して同じサブプログラムを呼び出すコード
- この2種類の呼び出しによって呼び出されるサブプログラム

次の例は、静的呼び出しをコーディングする方法を示しています。

```
PROCESS NODYNAM NODLL
IDENTIFICATION DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 RECORD-2                PIC X.                (6)
01 RECORD-1.               (2)
   05 PAY                   PICTURE S9(5)V99.
   05 HOURLY-RATE           PICTURE S9V99.
   05 HOURS                 PICTURE S99V9.

PROCEDURE DIVISION.
   CALL "SUBPROG" USING RECORD-1.                (1)
   CALL "PAYMASTR" USING RECORD-1 RECORD-2.      (5)
   STOP RUN.
```

次の例は、動的呼び出しをコーディングする方法を示しています。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
77 PGM-NAME                 PICTURE X(8).          (6)
01 RECORD-2                PIC x.                 (2)
01 RECORD-1.
   05 PAY                   PICTURE S9(5)V99.
   05 HOURLY-RATE           PICTURE S9V99.
   05 HOURS                 PICTURE S99V9.

PROCEDURE DIVISION.
   . . .
   MOVE "SUBPROG" TO PGM-NAME.
   CALL PGM-NAME USING RECORD-1.                (1)
   CANCEL PGM-NAME.
   MOVE "PAYMASTR" TO PGM-NAME.                 (4)
   CALL PGM-NAME USING RECORD-1 RECORD-2.      (5)
   STOP RUN.
```

次の例は、呼び出されるサブプログラムです。これは、前の2つの呼び出し側プログラムのそれぞれによって呼び出されます。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROG.
DATA DIVISION.
LINKAGE SECTION.
```


01	PAYREC.		(2)
10	PAY	PICTURE S9(5)V99.	
10	HOURLY-RATE	PICTURE S9V99.	
10	HOURS	PICTURE S99V9.	
77	PAY-CODE	PICTURE 9.	(6)
	PROCEDURE DIVISION USING PAYREC.		(1)
.	.	.	
.	EXIT PROGRAM.		(3)
.	ENTRY "PAYMASTR" USING PAYREC PAY-CODE.		(5)
.	.	.	
.	GOBACK.		(7)

(1)

処理は呼び出し側プログラムで開始されます。最初の CALL 文が実行されると、制御は最初の文に移ります。PROCEDURE DIVISION で SUBPROG、呼び出されるプログラムです。

各 CALL ステートメントでは、最初の USING オプションのオペランドが RECORD-1 として識別されます。

(2)

SUBPROG が制御権を受け取ると、RECORD-1 内の値が SUBPROG で使用可能になります。ただし、SUBPROG では、この値は PAYREC として参照されます。

PAYREC および PAY-CODE 内の PICTURE 文字ストリングの文字数は、記述は同一ではありませんが、RECORD-1 および RECORD-2 の文字数と同じです。

(3)

処理する場合 SUBPROG 到達する EXIT PROGRAM ステートメントでは、制御は呼び出し元のプログラムに戻されます。このプログラムでの処理は、2 番目の CALL ステートメントが実行されるまで継続されます。

(4)

動的に呼び出されるプログラムの例では、2 番目の CALL ステートメントは SUBPROG 内の別の入り口点を参照するため、2 番目の CALL ステートメントの前に CANCEL ステートメントが実行されます。

(5)

呼び出し側プログラム中の 2 番目の CALL ステートメントによって制御権は再び SUBPROG に移動しますが、今回の処理は SUBPROG 中の ENTRY ステートメントの次のステートメントから開始されます。

(6)

RECORD-1 内の値が再び PAYREC で使用可能になります。さらに、SUBPROG は対応する USING オペランド PAY-CODE を使用して、RECORD-2 中の値も使えるようになります。

静的にリンクされたプログラムから 2 度目に制御権が移動すると、SUBPROG は最後に使われた状態で使えるようになります(したがって、最初の実行時に SUBPROG ストレージ内の任意の値が変更されていると、その変更された値は依然として有効です)。しかし、動的にリンクされたプログラムから制御権が移動すると、CANCEL ステートメントが実行されているため、SUBPROG は初期状態で使用可能になります。

(7)

処理が GOBACK ステートメントに達すると、呼び出し側プログラム中の 2 番目の CALL ステートメントの直後のステートメントに制御権が戻されます。

呼び出されたプログラムと 2 つの呼び出しプログラムのいずれかの実行において、RECORD-1 最初の時間から CALL 2 番目は、2 番目の時点で渡された値 CALL ステートメントは元の値ではなく変更された値になります。元の値を使用する場合は、その値を保管しなければなりません。

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。

このタスクについて

どちらかを使用する CALL リテラルまたは CALL 識別子ネストされたプログラムを呼び出すためのステートメント。

電話することができます含まれている直接含まれているプログラムからのみプログラムが参照可能。含まれているプログラムとして COMMON その PROGRAM-ID 段落。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) 含まれている プログラムから呼び出すことができます。のみ含まれているプログラムは次のように識別できます COMMON。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場合は、次のガイドラインに従ってください。

- 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 含まれている プログラムの名前を固有にしてください。含まれている 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、アプリケーションの保守が一層容易になります。含まれている 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。含まれている プログラムが CONFIGURATION SECTION を持つことはできません。
- それぞれの 含まれている 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。
- 使用 END PROGRAM 終了マーカー含まれているプログラムを含みます。

ネストされたプログラムを含むプログラムをコンパイルするときは、THREAD オプションを使用できません。

関連概念

[614 ページの『ネストされたプログラム』](#)

関連参照

[616 ページの『名前のスコープ』](#)

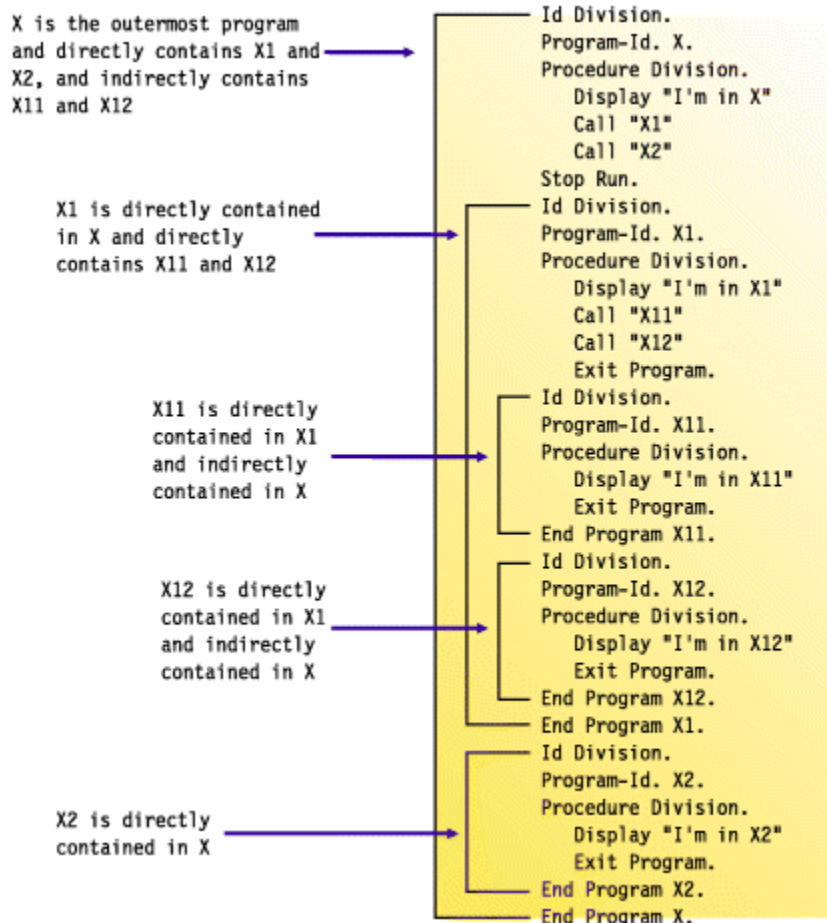
ネストされたプログラム

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。



615 ページの『例: ネストされたプログラムの構造』

関連タスク

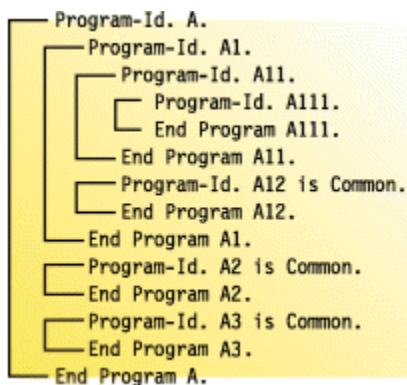
613 ページの『ネストされた COBOL プログラムの呼び出し』

関連参照

616 ページの『名前のスコープ』

例: ネストされたプログラムの構造

次の例は、一部の含まれているプログラムが 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。

名前のスコープ

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。検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、検索は終了します。一致が検出されない場合、エラーが存在します。

検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

かると探索は停止します。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。

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。このタイプの呼び出しを再帰的と言います。

このタスクについて

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必要があります。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、条件が通知されます。この条件を未処理のままにしておくと、実行単位が終了します。

関連タスク

[4 ページの『プログラムを再帰的として識別する』](#)

関連参照

PROGRAM-ID 段落 (*Enterprise COBOL for z/OS 言語解説書*)

オブジェクト指向プログラムとの間での呼び出し

オブジェクト指向 (OO) プログラムを含むアプリケーションを作成する場合、OO COBOL プログラムは DLL プログラムなので、1 つ以上のダイナミック・リンク・ライブラリー (DLL) に置くことができます。ただし、各クラス定義は個別の DLL に置かれている必要があります。

このタスクについて

COBOL DLL プログラムとの間での呼び出しでは、DLL リンケージを使用するか、または静的呼び出しのいずれかでなければなりません。COBOL DLL プログラムとの間での COBOL 動的呼び出しはサポートされていません。

COBOL の非 DLL プログラムから COBOL DLL プログラムを呼び出す必要がある場合は、DLL リンケージ・メカニズムに従っていることを保証する別の方法を使用することができます。

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・データ項目および関数ポインター・データ項目の設定には、形式 6 の SET ステートメントのみを使用することができます。

このタスクについて

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・データ項目のいずれかを指します。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。例えば、例外条件が発生したときに、ユーザー作成エラー処理ルーチンに制御権を渡すためには、まずそのルーチンの入り口アドレスを CEEHDLR (条件管理 言語環境呼び出し可能サービス) に渡して、そのルーチンを登録させなければなりません。
- 別の言語で作成されているプログラム。たとえば、C 関数のエンタリアドレスを取得するには、次のように関数を呼び出します。CALL RETURNING 声明。この結果、SET ステートメントの形式を使用して関数ポインターとして使用したりプロシージャ・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)。

SET ステートメントは、コンパイラー・オプション DYNAM|NODYNAM および DLL|NODLL の設定に応じて、プログラムと同じプログラム・オブジェクト内の入り口点、個別のプログラム・オブジェクト、または DLL からエクスポートされた入り口点を参照するように、プロシージャ・ポインターを設定します。したがって、これらのポインター・データ項目を使用する場合は、以下の点を考慮する必要があります。

- プログラムをコンパイルすると、NODYNAM そして NODLL オプションを使用してポインタ項目をリテラル値（エントリポイントの実際の名前）に設定する場合、その値は同じエントリポイントを参照する必要があります。プログラムオブジェクト。そうしないと、参照は解決できません。
- NODLL オプションを指定してプログラムをコンパイルするときに、ポインター項目を実行時に入り口点の名前が入る ID に設定する場合、またはポインター項目をリテラルに設定し DYNAM オプションを指定してコンパイルする場合、ポインター項目は（リテラルであっても変数であっても）別個のプログラム・オブジェクト・リンク内の入り口点を指している必要があります。入り口点は、1 次入り口点か、バインダー（リンケージ・エディター）の ALIAS ステートメントで指定した代替入り口点になります。
- コンパイルする場合は NODYNAM そして DLL オプションを使用してポインタ項目をリテラル値（エントリポイントの実際の名前）に設定する場合、その値は同じエントリポイントを参照する必要があります。プログラムオブジェクトまたは、DLL モジュールからエクスポートされるエントリポイント名になります。後者の場合は、プログラム・オブジェクトのリンク・エディットに、ターゲット DLL モジュールの DLL サイド・ファイルを組み込む必要があります。
- NODYNAM オプションおよび DLL オプションを指定してコンパイルするときに、ポインター項目を ID（実行時に入り口点名が入るデータ項目）に設定する場合、その ID 値は、DLL モジュールからエクスポートされた入り口点名を参照している必要があります。この場合、DLL モジュール名は、エクスポートされた入り口点の名前と一致しなければなりません。

ポインター項目を、動的に呼び出されるプログラム・オブジェクトの入り口アドレスに設定し、その後、動的に呼び出されたモジュールをプログラムで取り消すと、ポインター項目は未定義になります。その後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。

AMODE 24 アプリケーションに対しては、プロシージャ・ポインター呼び出しと関数ポインター呼び出しがサポートされています。ただし、これらの呼び出しに関してはアドレッシング・モードを切り替えることができないため、実行時には、呼び出される側のプログラムと呼び出す側のプログラムに同じアドレッシング・モードが必要となります。

COBOL のエントリポイント AMODE ANY 属性は次のいずれかで入力できます AMODE 31 または AMODE 24。ただし、初めてプログラムに入ったときに有効だった AMODE 値は、現在の Language Environment エンクレープ中は後続のすべてのプログラム再入に関しても有効でなければなりません。

COBOL 5 以降では、プロシージャおよび関数ポインターは、エントリー・ポイントを直接指すのではなく、関数記述子を指します。データ専用モジュール（テーブルなど）がある場合は、次の手法を使用できなくなりました。

```
77 DATA-FUNCTION-PTR USAGE FUNCTION-POINTER.  
77 DATA-PTR REDEFINES DATA-FUNCTION-PTR USAGE POINTER.  
  
SET DATA-FUNCTION-PTR TO ENTRY "DATAONLY"  
SET ADDRESS OF DATA TO DATA_PTR
```

しかし、以下の変更を行う必要があります。

```
77 DATA-PTR USAGE POINTER.  
  
CALL "DATAONLY" RETURNING DATA-PTR  
SET ADDRESS OF DATA TO DATA-PTR
```

データ専用モジュールは、データのアドレスを返すように改定されています。以下は LP(32) の例です。

```
DATAONLY CSECT  
        USING  *,15  
        LA     15,DATA  
        BR     14  
  
*  
DATA    DC     ...
```

...
END

この改訂された方法は、以前の COBOL でも機能します。

プロシージャと関数のポインターに対する呼び出しは、言語環境のスタック・フレームのあるモジュールから行わなければなりません。これは高水準プログラミング言語の場合にも当てはまります。この種の呼び出しをアセンブラー・モジュールから行う場合は、CEEENTRY マクロと CEETERM マクロを使用して LE スタック・フレームを提供し、必要とされる関連するレジスターの内容も提供する必要があります。

関連タスク

[619 ページの『使用するポインター・タイプの決定』](#)

[620 ページの『代替入り口点の呼び出し』](#)

[645 ページの『DLL でのプロシージャ・ポインターまたは関数ポインターの使用』](#)

関連参照

[396 ページの『DLL』](#)

[398 ページの『DYNAM』](#)

CANCEL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

フォーマット 6: プロシージャ・ポインターおよび関数ポインターのデータ項目用の SET

(*Enterprise COBOL for z/OS* 言語解説書)

ENTRY ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

[MVS プログラム管理: ユーザーズ・ガイドおよび解説書](#)

使用するポインター・タイプの決定

プロシージャ・ポインターを使用して、他の COBOL プログラムを呼び出したり、言語環境呼び出し可能サービスを呼び出したりします。関数ポインターを使用して、C/C++プログラムや Java Native Interface が提供するサービスと通信します。

このタスクについて

COBOL-COBOL 間呼び出しの場合、プロシージャ・ポインターは関数ポインターよりも一層効率的であり、言語環境条件処理サービスの呼び出しに必要です。

C で書かれた多くの呼び出し可能サービスは、関数ポインターを戻します。以下に示すように COBOL 関数ポインターを使用して、COBOL プログラムからそのような C 関数ポインターを呼び出すことができます。

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. DEMO.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
*  
WORKING-STORAGE SECTION.  
01 FP USAGE FUNCTION-POINTER.  
*  
PROCEDURE DIVISION.  
    CALL "c-function" RETURNING FP.  
    CALL FP.
```

関連タスク

[645 ページの『DLL でのプロシージャ・ポインターまたは関数ポインターの使用』](#)

[785 ページの『JNI サービスへのアクセス』](#)

代替入り口点の呼び出し

このタスクについて

呼び出し先プログラムの ENTRY ラベルを使用して、プログラムが実行を開始する別のエントリー・ポイントを指定できます。

代替入り口点への静的呼び出しは制限なしで作動します。

代替入り口点に対する動的呼び出しには、明示的に指定された NAME および/または ALIAS バインダー (リンケージ・エディター) 制御ステートメント、またはそれらを自動的に生成する NAME コンパイラー・オプションを使用する必要があります。

プライマリ エントリ ポイントを含む複数のエントリー ポイントでコンパイルユニットを動的に呼び出すと、個別のコンパイルユニットが呼び出されているかのように、各エントリーに対して WORKING-STORAGE の一意のインスタンスが生成されます。

コンパイル単位を呼び出し側と静的にリンクせずに 1 つの WORKING-STORAGE インスタンスを複数の入り口点の間で共有する場合は、そのインスタンスを DLL にして、EXPORTALL を使用して 2 次入り口点を公開し、動的呼び出しではなく DLL 呼び出しを使用してコンパイル単位を呼び出す必要があります。

代替エントリー・ポイントに対する動的呼び出しおよびプロシージャ・ポインター呼び出しは、AMODE 31 の下で実行する必要があります。これらの呼び出しは、LP (64) でコンパイルされたプログラムではサポートされません。

612 ページの『例: 静的および動的 CALL ステートメント』

関連参照

421 ページの『名前』

CANCEL ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

ENTRY ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

MVS プログラム管理: ユーザーズ・ガイドおよび解説書

プログラムを再入可能にする

複数のユーザーが同時にアプリケーション・プログラムを実行する場合 (例えば、異なるアドレス・スペースのユーザーがリンク・パック域にある 1 つのプログラムにアクセスする場合)、RENT オプションを使用してコンパイルすることによって、プログラムを再入可能にする必要があります。

このタスクについて

プログラマーとしては、変数の複数コピーについて心配する必要はありません。コンパイラーが、オブジェクト・モジュールの中で必要な再入可能制御コードを作成します。

以下の Enterprise COBOL プログラムは再入可能でなければなりません。

- CICS で使用されるプログラム
- IMS でプリロードされるプログラム
- Db2 ストアード・プロシージャとして使用されるプログラム
- z/OS ユニックス 環境で実行されるプログラム
- DLL サポートで使用可能なプログラム
- オブジェクト指向構文を使用するプログラム

再入可能プログラムの場合は、DATA コンパイラー・オプションと HEAP および ALL31 ランタイム・オプションを使用して、WORKING-STORAGE などの動的データ域を 16 MB 境界より下のストレージから獲得するか、または上のストレージから獲得するかを制御します。

関連概念

40 ページの『ストレージとそのアドレス可能性』

関連タスク

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

[321 ページの『第 17 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

[435 ページの『RENT』](#)

[391 ページの『データ』](#)

言語環境プログラミング・リファレンス (ALL31、HEAP)

第 30 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、通常は共通データにアクセスする必要があります。

このタスクについて

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

[16 ページの『別のプログラムからのデータの使用』](#)

[789 ページの『Java とのデータ共用』](#)

[623 ページの『データの受け渡し』](#)

[627 ページの『LINKAGE SECTION のコーディング』](#)

[628 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』](#)

[632 ページの『戻りコード情報の引き渡し』](#)

[633 ページの『EXTERNAL 節によるデータの共用』](#)

[634 ページの『プログラム間でのファイルの共用 \(外部ファイル\)』](#)

[637 ページの『z/OS 下でのメインプログラム・パラメーターへのアクセス』](#)

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。

このタスクについて

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、パラメータに対して 3 つの方法のどれかが指定されていない場合、あるいは暗黙のうちに指定されている場合に、そのパラメータを渡すメカニズムとして想定されるものである。BY REFERENCE パラメータはオプションであり、その存在は OMITTED クラステストを使うことでテストできる。さらに、CALL USING パラメータリストの最後には、OMITTED という余分なパラメータが想定されている。

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

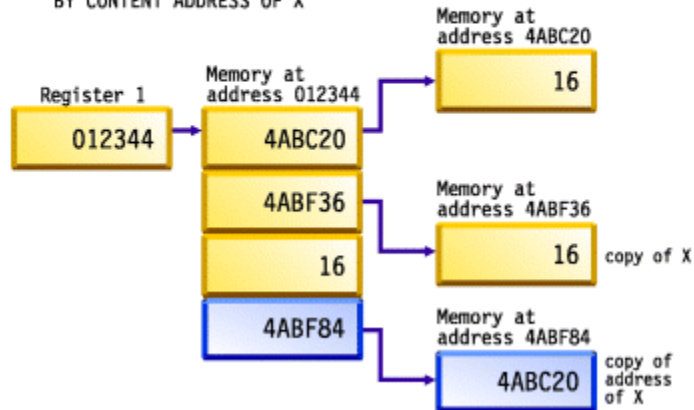
呼び出し側プログラムまたはメソッドは、送り出しデータ項目を参照せず、*literal* または *identifier* の値を渡します。呼び出されるプログラムまたはメソッドは、その中のパラメーターを変更できます。しかし、サブプログラムまたはメソッドは送り出しデータ項目の一時コピーへのアクセス権しか持っていないので、どの変更内容も呼び出し側プログラムの引数に影響を与えません。

次の図は渡された値の違いを示しています BY REFERENCE、BY CONTENT、そして BY VALUE:


```

MOVE 16 TO X.
CALL ABC USING
  BY REFERENCE X
  BY CONTENT X
  BY VALUE X
  BY CONTENT ADDRESS OF X

```



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 73. 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 識別子	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す。
CALL . . . BY REFERENCE ADDRESS OF 識別子	<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す。
CALL . . . BY REFERENCE ファイル名	データ制御ブロック (DCB) をアセンブラー・プログラムに渡す。	ファイル名は QSAM 順次ファイルを参照しなければなりません。 ¹
CALL . . . BY CONTENT ADDRESS OF 識別子	<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 識別子	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 リテラル	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 識別子	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。
の組み合わせ BY REFERENCE そして BY CONTENT のような： 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。	

表 73. CALL ステートメントでデータを渡す方法 (続き)		
コード	目的	コメント
CALL . . . BY VALUE 識別子	プログラムにデータを渡すには、例えば C/C++ プログラム、使用する BY VALUE パラメータリンク規約	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 リテラル	プログラムにデータを渡すには、例えば C/C++ プログラム、使用する BY VALUE パラメータリンク規約	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 識別子	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。	
1. ファイル名は CALL オペランドは、IBM COBOL の拡張。拡張部分の使用は一般的に、コンパイラーの特定の内部インプリメンテーションによって異なります。制御ブロック・フィールドの設定は、今後のリリースで変更される場合があります。制御ブロックに対して行われる変更は、各自の責任であって、IBM がサポートするものではありません。		

関連概念

40 ページの『ストレージとそのアドレス可能性』

関連タスク

625 ページの『呼び出し側プログラムの中での引数の記述』

626 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』

627 ページの『OMITTED 引数』

633 ページの『CALL ... の指定 RETURNING』

633 ページの『EXTERNAL 節によるデータの共用』

634 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)』

789 ページの『Java とのデータ共用』

関連参照

CALL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

USING 句 (*Enterprise COBOL for z/OS* 言語解説書)

INVOKE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

呼び出し側プログラムの中での引数の記述

呼び出しプログラムでは、引数を DATA DIVISION 他のデータ項目と同様に DATA DIVISION。

このタスクについて

引数のためのストレージは、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 で割り振られます。これらのデータ項目は、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ければなりません。

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。

CALL 引数を AMODE 31 プログラムから AMODE 24 プログラムに渡すには、AMODE 24 サブプログラムによってアドレッシングされるように、その引数を 16 MB 境界より下のストレージに置いておく必要があります。

- 再入可能 AMODE 31 プログラムの場合、DATA(24) オプションを使用してプログラムをコンパイルするか、WORKING-STORAGE がヒープ・ストレージから割り振られている場合は Language Environment のランタイム・オプション HEAP(, ,BELOW) を指定します。WORKING-STORAGE がヒープ・ストレージからいつ割り振られるのかについて詳しくは、[40 ページの『ストレージとそのアドレス可能性』](#)を参照してください。
- NORENT オプションを使用してコンパイルされた 再入不可プログラムについては、RMODE(24) オプションまたは RMODE(AUTO) オプションを使用してコンパイルを行います。その結果として、以下の項目が 16 MB 境界より下で割り振られます。これらの項目は AMODE 24 プログラムに引数として渡すことができます。
 - WORKING-STORAGE データ項目 (EXTERNAL 節なし)
 - FD レコード域
 - QSAM バッファ
- 混合 AMODE アプリケーションの場合、Language Environment のランタイム・オプション ALL31(OFF) および STACK(, ,BELOW) が必要です。その結果として、LOCAL-STORAGE SECTION データ項目と、EXTERNAL 属性を持つデータ項目が 16 MB 境界より下で割り振られます。これらの項目は AMODE 24 プログラムに引数として渡すことができます。

関連概念

[40 ページの『ストレージとそのアドレス可能性』](#)

関連タスク

[627 ページの『LINKAGE SECTION のコーディング』](#)

[628 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 (*Enterprise COBOL for z/OS 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。

このタスクについて

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

[627 ページの『LINKAGE SECTION のコーディング』](#)

関連参照

USING 句 (*Enterprise COBOL for z/OS 言語解説書*)

OMITTED 引数

1 つまたは複数の BY REFERENCE 引数を呼び出されるプログラムに渡さないように指定するには、CALL 文の引数の代わりに OMITTED キーワードを記述するか、CALL 文の最後の USING 引数の後に OMITTED 引数があると仮定する機能を使用します。

このタスクについて

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。

コールされたプログラムでは、引数が OMITTED :

- OMITTED クラス・テストを使用すると、パラメータが OPTIONAL と指定されたときに、引数が OMITTED であったかどうかを検証することができます：

```
PROCEDURE DIVISION USING A OPTIONAL B BY REFERENCE C
    . . .
    IF B is OMITTED then ...
```

- 対応するパラメータのアドレスを NULL と比較することができる。例:

```
Program-ID. sub1.
    . . .
    Procedure Division Using RPARM1, RPARM2, RPARM3.
        If Address Of RPARM2 = 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 (*Enterprise COBOL for z/OS 言語解説書*)

USING 句 (*Enterprise COBOL for z/OS 言語解説書*)

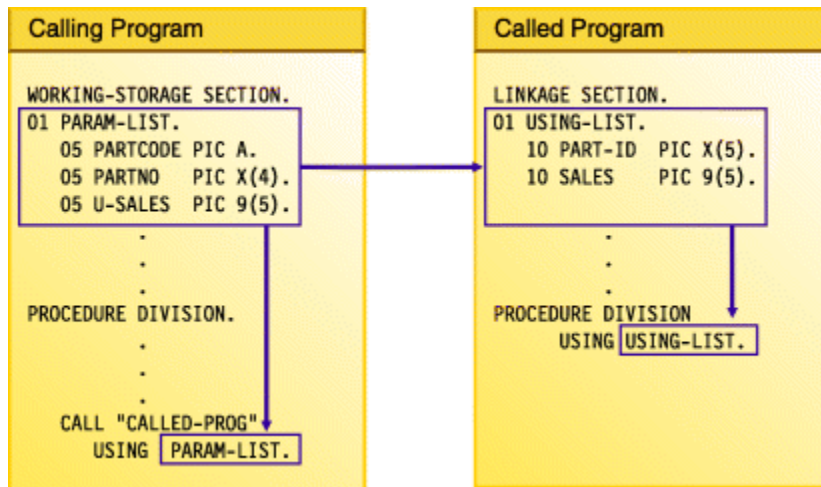
LINKAGE SECTION のコーディング

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。

このタスクについて

呼び出し先プログラムの identifier リストのデータ名の方が、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、データ項目が 1 つのプログラムから別のプログラムに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。対照的に、呼び出されたプログラムでは、部品のコードと部品番号が 1 つのデータ項目に結合されます (PART-ID)。呼び出されたプログラムでは、PART-ID これらの項目への唯一の有効な参照です。

関連タスク

637 ページの『z/OS 下でのメインプログラム・パラメーターへのアクセス』

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。

このタスクについて

```

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 (*Enterprise COBOL for z/OS 言語解説書*)

USING 句 (*Enterprise COBOL for z/OS 言語解説書*)

CALL ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

受け渡されるデータのグループ化

このタスクについて

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。つまり、WORKING-STORAGE SECTION 呼び出し側のプログラムと LINKAGE SECTION 呼び出されたプログラムの。

関連タスク

627 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 (*Enterprise COBOL for z/OS 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' と一緒にストリング処理ステートメントを使用する場合、COBOL はヌル終了ストリングをサポートします。

このタスクについて

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

[116 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
(*Enterprise COBOL for z/OS 言語解説書*)

ポインターによるチェーン・リストの処理

レコード領域のアドレスを渡したり受け取ったりする必要がある場合は、ポインタデータ項目を使用できます。これは、USAGE IS POINTER 節または ADDRESS OF 特殊レジスタ。

このタスクについて

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。

630 ページの『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

627 ページの『LINKAGE SECTION のコーディング』

628 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

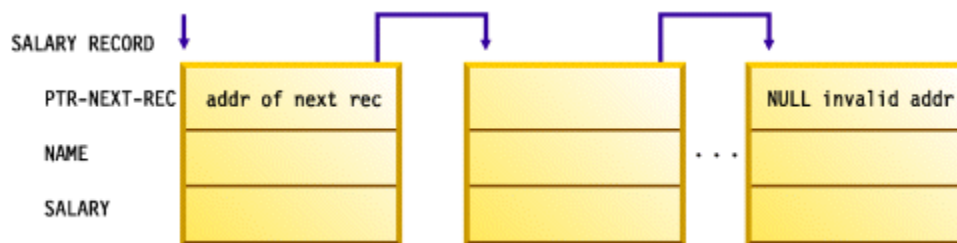
関連参照

SET ステートメント (*Enterprise COBOL for z/OS* 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・リストについて説明します。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.          (2)
   02 PTR-NEXT-REC      POINTER.
   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 ACCUMULATE 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、初期値 NULL を持つポインター・データ項目として定義されています。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・リスト内のレコードの記述が含まれています。さらに、CALL ステートメントの USING 節で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。

他のプログラムから受信したアドレスを増やすには、LINKAGE SECTION そして PROCEDURE DIVISION
このような：

```
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

629 ページの『[ポインターによるチェーン・リストの処理](#)』

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が、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。)

このタスクについて

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。 使用する場合は PROCEDURE DIVISION . . . RETURNING と
CALL . . . RETURNING、RETURN-CODE レジスタは設定されません。

RETURN-CODE 特殊レジスターの使用

このタスクについて

COBOL プログラムがその呼び出し側に戻ると、RETURN-CODE 特殊レジスターの内容が、レジスター 15
に保管されます。

制御が呼び出しから COBOL プログラムまたはメソッドに戻されると、レジスター 15 の内容は、呼び出し
側プログラムまたはメソッドの RETURN-CODE 特殊レジスターに保管されます。 COBOL プログラムから
オペレーティング・システムに制御が戻されると、特殊レジスターの内容はユーザー戻りコードとして戻
されます。

非 COBOL プログラムから COBOL プログラムに制御が戻る場合には、このような RETURN-CODE 特殊レジ
スターの処理を考慮する必要が生じることがあります。 非 COBOL プログラムが、戻りコードを戻すために
レジスター 15 を使用しない場合、COBOL プログラムの RETURN-CODE 特殊レジスターが無効値で更新さ
れることがあります。 Enterprise COBOL プログラムがオペレーティング・システムに戻る前に、この特殊
レジスターを意味のある値に設定しない限り、無効な戻りコードがシステムに渡されます。

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 特殊レジスター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。

このタスクについて

```
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
```

CEEPIPI: 言語環境の事前初期設定サービス (CEEPIPI) を使用して呼び出されるプログラムで PROCEDURE DIVISION RETURNING を指定すると結果は不定となります。

CALL ... の指定 RETURNING

指定できるのは RETURNING のフレーズ CALL 声明文 C/C++ 関数または COBOL サブルーチンに適用されます。

このタスクについて

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。

このタスクについて

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体が 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。
- コーディングすることはできません VALUE いかなる条項についてもグループアイテムまたは従属アイテムは EXTERNAL。

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。

このタスクについて

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。

634 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 (Enterprise COBOL for z/OS 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	関数
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。
ef1openo	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。
ef1write	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。
ef1openi	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。
ef1read	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。
ef1close	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 段落に入れられます。

```
Select ef1
Assign To ef1
```

```
File Status Is efs1
Organization Is Sequential.
```

- efile 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 efile.
WORKING-STORAGE SECTION.
    Copy efwrkstg.
PROCEDURE DIVISION.
    Call "eflopeno"
    Call "ef1write"
    Call "ef1close"
    Call "ef1openi"
    Call "ef1read"
    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 "ef1close"
    Goback.
End Program ef1.
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 efile.
WORKING-STORAGE SECTION.
    Copy efwrkstg.
PROCEDURE DIVISION.
    Open Output ef1
    If efs1 Not = 0
        Display "file status " efs1 " on open output"
        Stop Run
    End-If
    Goback.
End Program eflopeno.
IDENTIFICATION DIVISION.
Program-Id.
    ef1write.
*
```

* 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 efs1
If efs1 Not = 0
    Display "file status " efs1 " on close"
    Stop Run
End-If
Goback.
End Program efs1close.

```

z/OS 下でのメインプログラム・パラメーターへのアクセス

z/OS 下で Enterprise COBOL プログラムを実行し、そのプログラムにパラメーター・ストリングを (例えば JCL または TSO コマンドを使用して) 渡す場合、パラメーター・リストは、ストリング長を含むハーフワード接頭部を持つ文字ストリングで構成されます。

このタスクについて

パラメータ文字列にアクセスするには、LINKAGE SECTION および、以下の例に示すような標準 COBOL コーディング:

637 ページの『例: z/OS 下でのメインプログラム・パラメーターへのアクセス』

また、以下の 言語環境 呼び出し可能サービス (下の関連参照で説明されています) のいずれかを呼び出すことにより、パラメーター・ストリングを取得できます。

- CEE3PRM (パラメーター・ストリングの照会): パラメーター・ストリングを取得します (80 文字以下の場合)
- CEE3PR2 (パラメーター・ストリング長の照会): パラメーター・ストリングとその長さを取得します

どちらの場合も、パラメーター・ストリングには、プログラム引数、ランタイム・オプション、またはその両方が含まれる場合があります。CBLOPTS ランタイム・オプションの設定によって、期待されるプログラム実引数およびランタイム・オプションの相対順序が決まります。CBLOPTS(ON) (デフォルト) が有効で、プログラム実引数とランタイム・オプションの両方がパラメーター・ストリングで渡される場合は、それらを次の順序で配置し、スラッシュで区切る必要があります。

```
program_arguments/runtime_options
```

注: AMODE 64 では、CBLOPTS ランタイム・オプションはサポートされなくなり、プログラム引数とランタイム・オプションは以下の順序で渡される:

```
runtime_options/program_arguments
```

詳細については、下の関連情報を参照してください。

関連タスク

627 ページの『LINKAGE SECTION のコーディング』

584 ページの『z/OS ユニックス 下でのメインプログラム・パラメーターへのアクセス』

1025 ページの『JCL パラメーターへのアクセス』

言語環境プログラミングガイド (実行時オプションとプログラム引数の指定、メインルーチンにパラメータを受け取る準備)

関連参照

言語環境のカスタマイズ (CBLOPTS(COBOL のみ))

言語環境プログラミング・リファレンス (CEE3PRM, CEE3PR2)

言語環境プログラミングガイド (COBOL 互換性に関する考慮事項、z/OS UNIX 下での COBOL プログラムの実行)

言語環境プログラミングガイド for 64 ビット (仮想アドレス指定モード (実行時オプションとプログラム引数の指定))

例: z/OS 下でのメインプログラム・パラメーターへのアクセス

次の例は、z/OS 下で実行される COBOL プログラムに渡されるパラメーター・ストリングを受け取る方法と、パラメーター・ストリングにアクセスするために使用可能なコーディングを示しています。

```

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.

```

CBLOPTS(ON) ランタイム・オプションが有効で、プログラムを実行するために使用する JCL または TSO コマンドで次の引数を渡すとしてます。

```
'95/'
```

結果の出力は、次のとおりです。

```

parm-len=002
parm-string='95'
case ninety-five

```

第 31 章 DLL または DLL アプリケーションの作成

ダイナミック・リンク・ライブラリー (DLL) または DLL アプリケーションの作成は、通常の COBOL アプリケーションの作成と似ています。すなわち、ソース・コードを書き、コンパイルし、リンクします。

このタスクについて

DLL または DLL アプリケーションを作成するには、以下の特別な考慮事項が適用されます。

- プログラム・オブジェクトまたはアプリケーションの各部分を互いに関連付ける方法、または他の DLL と関連付ける方法を決定する。
- どんなリンクまたは呼び出しのメカニズムを使用するかを決定する。

DLL プログラム・オブジェクトを作成するか、または別個の DLL を参照するプログラム・オブジェクトを作成するかによって、使用する必要があるコンパイラおよびバインダー (リンケージ・エディター) オプションは若干異なります。

関連概念

[639 ページの『ダイナミック・リンク・ライブラリー \(DLL\)』](#)

関連タスク

[314 ページの『z/OS ユニックスのもとでの DLL の作成』](#)

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

[641 ページの『DLL のリンク』](#)

[643 ページの『DLL での CALL ID の使用』](#)

[645 ページの『DLL リンケージと動的呼び出しの併用』](#)

[648 ページの『C/C++ プログラムでの COBOL DLL の使用』](#)

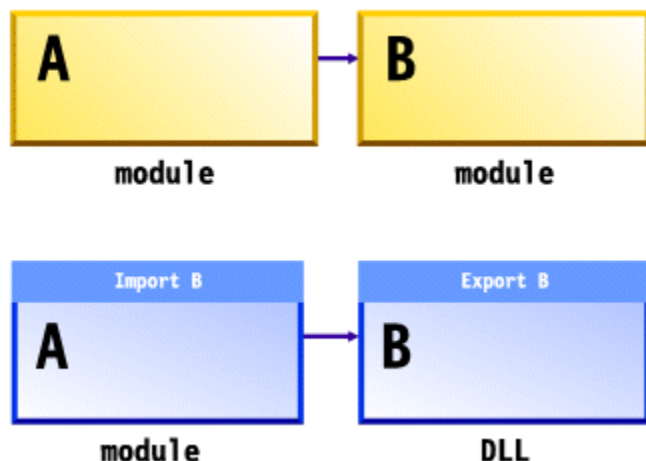
[648 ページの『OO COBOL アプリケーションでの DLL の使用』](#)

[645 ページの『DLL でのプロシージャ・ポインターまたは関数ポインターの使用』](#)

ダイナミック・リンク・ライブラリー (DLL)

DLL は、プログラム・オブジェクトであり、他の別個のプログラム・オブジェクトからアクセスできます。

DLL は、プログラム、関数、または変数の定義を DLL、DLL アプリケーション、または非 DLL ヘクスポートするという点で、従来のプログラム・オブジェクトとは異なります。このため、ターゲット・ルーチンを、参照ルーチンと同じプログラム・オブジェクトにリンクする必要はありません。アプリケーションが初めて別個の DLL を参照すると、システムは自動的にその DLL をメモリーにロードします。すなわち、DLL 内でプログラムを呼び出すことは、動的 CALL を使用してプログラム・オブジェクトを呼び出すことに似ています。



DLL アプリケーションは、プログラム、関数、または変数のインポートされた定義を参照するアプリケーションです。

64 ビット COBOL は、DLL アプリケーションと DLL 以外のアプリケーションをどちらもサポートします。このコンパイラによって生成されるオブジェクト・ファイルには、DLL 使用とも DLL 以外の使用とも互換性があるため、DLL コンパイル・オプションは必要ありません。また、すべての 64 ビット COBOL プログラムは再入可能であるため、RENT も必要ありません。NORENT コンパイル・オプションはサポートされていません。シンボルを DLL からエクスポートするためには、EXPORTALL コンパイル・オプションが必要です。

一部の機能は z/OSDLL は COBOL ダイナミックライブラリが提供する機能と重複している。CALL ステートメントでは、DLL は通常の z/OS プログラムオブジェクト動的呼び出し:

- DLL は COBOL、および C/C++ 間で共通なので、複数のプログラム言語を使用するアプリケーションの相互協調処理が向上します。再入可能な COBOL および C/C++ DLL も、スムーズに相互協調処理することができます。
- 長いプログラム名を持つ、別個の DLL モジュールに入っているプログラムへの呼び出しを行うことができます。(動的呼び出し解決は、プログラム名を 8 文字に切り捨てます。) COBOL オプションの PGMNAME (LONGUPPER) または PGMNAME (LONGMIXED) と COBOL DLL サポートを使用すると、最高 160 文字までの名前を持つプログラム・オブジェクト間で呼び出しを行うことができます。

DLL は、z/OS プログラム管理バインダーによって提供される機能に基づいて、IBM z/OS 言語環境によってサポートされます。DLL サポートは、z/OS のもとでバッチ環境で実行中のアプリケーションで、あるいは TSO、CICS、z/OS ユニックス、または IMS 環境で使用可能です。

関連参照

432 ページの『PGMNAME』

MVS プログラム管理: ユーザーズ・ガイドおよび解説書 (DLL のためのバインダー・サポート)

DLL を作成するためのプログラムのコンパイル

DLL 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、そのプログラムは DLL サポートに使用可能になります。DLL サポートを使用するアプリケーションは再入可能でなければなりません。このため、RENT コンパイラ・オプションを使用してコンパイルし、それらを RENT バインダー・オプションとリンクしなければなりません。

このタスクについて

AMODE 64 に関する考慮事項

LP (64) コンパイラ・オプションを指定してコンパイルされた COBOL プログラムは、既に DLL サポートに対応していて再入可能です。LP (64) オプションは DLL と RENT を暗黙指定します。NODLL や NORENT が指定されている場合、コンパイラはこれらのオプションを破棄して警告メッセージを発行します。LP (64) を使用してコンパイルされたプログラムは、DLL または非 DLL アプリケーションにリンクできます。

LP (64) オプションは RMODE (ANY) を暗黙指定します。RMODE (24) が指定されている場合、コンパイラはオプションを破棄して警告メッセージを発行します。AMODE 64 COBOL プログラムは 2 GB 境界より下でロードされます。

DLL サポートのあるアプリケーションでは、プログラムまたはクラスの場所に応じて以下のコンパイラ・オプションを使用してください。

表 74. DLL アプリケーションのコンパイラ・オプション		
プログラムまたはクラスの場合	で LP (32) コンパイル:	LP (64) でコンパイルに使用するもの:
ルート・プログラム・オブジェクト	DLL, RENT, NOEXPORTALL	NOEXPORTALL

表 74. DLL アプリケーションのコンパイラー・オプション (続き)		
プログラムまたはクラスの場合	で LP (32) コンパイル:	LP (64) でコンパイルに使用するもの:
他のプログラム・オブジェクトによって使用される DLL プログラム・オブジェクト	DLL, RENT, EXPORTALL	EXPORTALL

DLL プログラム・オブジェクトに、DLL モジュール内からだけ使用されるプログラムが含まれている場合は、NOEXPORTALL を使用してコンパイルすることによって、これらのルーチンを隠すことができます。

642 ページの『例: プロシージャ型 DLL アプリケーションのサンプル JCL』

関連タスク

314 ページの『z/OS ユニックスのもとでの DLL の作成』

641 ページの『DLL のリンク』

639 ページの『第 31 章 DLL または DLL アプリケーションの作成』

関連参照

396 ページの『DLL』

401 ページの『EXPORTALL』

415 ページの『LP』

435 ページの『RENT』

DLL のリンク

DLL 可能オブジェクト・モジュールを別個の DLL プログラム・オブジェクトにリンクするか、またはそれらを静的にリンクすることができます。リンク時に、アプリケーションを 1 つのモジュールとしてパッケージするか、いくつかの DLL モジュールとしてパッケージするかを決定することができます。

このタスクについて

DLL アプリケーションをリンクする場合は、z/OS バインダーの DLL サポートをお勧めします。バインダーは COBOL コンパイラーからの出力を直接受け取ることができます。

バインダー・ベースの DLL は、PDSE または z/OS ユニックスファイル内に常駐させなければなりません。

バインダーを使用して DLL アプリケーションをリンクする場合は、次のオプションを使用します。

表 75. DLL アプリケーションのバインダー・オプション	
コードのタイプ	リンクに使用するバインダー・パラメーター
DLL アプリケーション	DYNAM(DLL), RENT
大/小文字混合のエクスポート済みプログラム名を使用するアプリケーション クラス定義または INVOKE ステートメント	CASE(MIXED)

SYSDEFSD DD ステートメントを指定して、バインダーが DLL 定義サイド・ファイルを作成するデータ・セットを指示してください。このサイド・ファイルには、DLL によってエクスポートされた各記号についての IMPORT 制御ステートメントが入っています。バインダー SYSLIN 入力 (DLL コードを参照するバインディング・コード) には、リンクされるモジュールから参照される DLL のための DLL 定義サイド・ファイルを含めなければなりません。

DLL リンケージに使用させたくないプログラムがモジュールに含まれている場合には、定義サイド・ファイルを編集してそれらのプログラムを除去することができます。

642 ページの『例: プロシージャ型 DLL アプリケーションのサンプル JCL』

関連タスク

314 ページの『z/OS ユニックスのもとでの DLL の作成』

639 ページの『第 31 章 DLL または DLL アプリケーションの作成』

640 ページの『DLL を作成するためのプログラムのコンパイル』

関連参照

MVS プログラム管理: ユーザーズ・ガイドおよび解説書 (DLL のためのバインダー・サポート)

例: プロシージャ型 DLL アプリケーションのサンプル JCL

COBOL DLL サブプログラムを呼び出す COBOL メインプログラム

次の例は、COBOL DLL サブプログラムを呼び出す COBOL メイン プログラムで構成されるアプリケーションを作成する方法を示しています。

最初のステップは DLL を作成するプログラムオブジェクト COBOL サブプログラムを含む DemoDLLSubprogram そして DemoDLLSub2。2 番目のステップではメインを作成しますプログラムオブジェクト COBOL プログラムを含む MainProgram。3 番目のステップでアプリケーションを実行します。

```
//DILLSAMP JOB ,
// TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,
// NOTIFY=&SYSUID,USER=&SYSUID
// SET LEPMX='SYS1'
//*-----
//* Compile COBOL subprogram, bind to form a DLL.
//*-----
//STEP1 EXEC IGYWCL,REGION=80M,GOPGM=DEMOMLL,
// PARM.COBL='RENT,PGMN(LM),DLL,EXPORTALL',
// PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBL.SYSIN DD *
  Identification division.
  Program-id. "DemoDLLSubprogram".
  Procedure division.
    Display "Hello from DemoDLLSubprogram!".
  End program "DemoDLLSubprogram".

  Identification division.
  Program-id. "DemoDLLSub2".
  Procedure division.
    Display "Hello from DemoDLLSub2!!".
  End program "DemoDLLSub2".
/*
//LKED.SYSDEFSD DD DSN=&&SIDEDECK,UNIT=SYSDA,DISP=(NEW,PASS),
// SPACE=(TRK,(1,1))
//LKED.SYSLMOD DD DSN=&&GOSET(&GOPGM),DSNTYPE=LIBRARY,DISP=(MOD,PASS)
//LKED.SYSIN DD DUMMY
//*-----
//* Compile and bind COBOL main program
//*-----
//STEP2 EXEC IGYWCL,REGION=80M,GOPGM=MAINPGM,
// PARM.COBL='RENT,PGMNAME(LM),DLL',
// PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBL.SYSIN DD *
  Identification division.
  Program-id. "MainProgram".
  Procedure division.
    Call "DemoDLLSubprogram"
    Call "DemoDLLSub2"
    Stop Run.
  End program "MainProgram".
/*
//LKED.SYSIN DD DSN=&&SIDEDECK,DISP=(OLD,DELETE)
//*-----
//* Execute the main program, calling the subprogram DLL.
//*-----
//STEP3 EXEC PGM=MAINPGM,REGION=80M
//STEPLIB DD DSN=&&GOSET,DISP=(OLD,DELETE)
// DD DSN=&LEPMX..SCEERUN,DISP=SHR
// DD DSN=&LEPMX..SCEERUN2,DISP=SHR
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
```

COBOL メインプログラムが C DLL サブプログラムを呼び出す

次の例は、C DLL サブプログラムを呼び出す COBOL メイン プログラムで構成されるアプリケーションを作成する方法を示しています。

最初のステップは DLL を作成するプログラムオブジェクト C サブプログラムを含む DemoDLLSubprogram そして DemoDLLSub2。2 番目のステップではメインを作成しますプログラムオブジェクト COBOL プログラムを含む MainProgram。3 番目のステップでアプリケーションを実行します。

```
//DILLSAMP JOB ,
// TIME=(1),MSGLEVEL=(1,1),MSGCLASS=H,CLASS=A,
// NOTIFY=&SYSUID,USER=&SYSUID
// SET LEPFX='SYS1'
//*
// JCLLIB ORDER=(CBC.SCCNPRC)          TO FIND EDCCL
//*-----
// * Compile C subprogram, bind to form a DLL.
//*-----
//STEP1 EXEC EDCCL,REGION=200M,GOPGM=DEMODLL,
//      PARM.COMPILE='DLL,EXPORTALL,LIST',
//      PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COMPILE.SYSIN DD *
#include<stdio.h>
void DemoDLLSubprogram(){
    printf("Hello from DemoDLLSubprogram!\n");
}

void DemoDLLSub2(){
    printf("Hello from DemoDLLSub2!!\n");
}
/*
//LKED.SYSDEFSD DD DSN=&&SIDEDECK,UNIT=SYSDA,DISP=(NEW,PASS),
//      SPACE=(TRK,(1,1))
//LKED.SYSLMOD DD DSN=&&GOSET(&GOPGM),DSNTYPE=LIBRARY,DISP=(MOD,PASS)
//LKED.SYSIN DD DUMMY
//*-----
// * Compile and bind COBOL main program
//*-----
//STEP2 EXEC IGYWCL,REGION=80M,GOPGM=MAINPGM,
//      PARM.COBOLE='RENT,PGMNAME(LM),DLL',
//      PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBOL.SYSIN DD *
Identification division.
Program-id. "MainProgram".
Procedure division.
    Call "DemoDLLSubprogram"
    Call "DemoDLLSub2"
    Stop Run.
End program "MainProgram".
/*
//LKED.SYSIN DD DSN=&&SIDEDECK,DISP=(OLD,DELETE)
//*-----
// * Execute the main program, calling the subprogram DLL.
//*-----
//STEP3 EXEC PGM=MAINPGM,REGION=80M
//STEPLIB DD DSN=&&GOSET,DISP=(OLD,DELETE)
//      DD DSN=&LEPFX..SCEERUN,DISP=SHR
//      DD DSN=&LEPFX..SCEERUN2,DISP=SHR
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
```

DLL での CALL ID の使用

DLL オプションを指定してコンパイルされた COBOL プログラムの場合、CALL *identifier* および CALL *literal* ステートメントを使用して DLL を呼び出すことができます。ただし、CALL *identifier* の場合には、追加の考慮事項があります。

このタスクについて

identifier の内容または *literal* の場合は、次のいずれかのプログラムの名前を使用してください。

- CALL *identifier* ステートメントを含んでいるプログラムからの呼び出しに適格な、同じコンパイル単位内のネストされたプログラム。
- 別個のバインドされた DLL モジュール内のプログラム。ターゲット・プログラム名は DLL からエクスポートされていなければなりません。また、DLL モジュール名は、ターゲット・プログラムのエクスポートされた名前と一致していなければなりません。

ネストされていない場合、ランタイム環境は、CALL ステートメントを含むプログラムの PGMNAME コンパイラー・オプションの設定に従って、*identifier* 内のプログラム名を解釈します。また、ターゲット DLL からエクスポートされたプログラム名は、ターゲット・プログラムのコンパイル時に使用された PGMNAME オプションの設定に従って解釈されます。

z/OS UNIX ファイルシステムにおけるターゲット DLL の探索には、大/小文字の区別があります。ターゲット DLL が PDSE メンバーである場合、DLL メンバー名は 8 文字以下でなければなりません。PDSE メンバーとして DLL を探索するために、ランタイム環境は名前を自動的に大文字に変換します。

ランタイム環境が解決できない場合は、CALL どちらの場合も、制御は ON EXCEPTION または ON OVERFLOW のフレーズ CALL 声明。この状況で CALL ステートメントにこれらの句の 1 つが指定されていないと、言語環境は重大度 3 の条件を発生させます。

関連タスク

[645 ページの『DLL リンケージと動的呼び出しの併用』](#)

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

[641 ページの『DLL のリンク』](#)

関連参照

[396 ページの『DLL』](#)

[432 ページの『PGMNAME』](#)

CALL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

[644 ページの『z/OS UNIX ファイルシステムでの DLL の探索順序』](#)

z/OS UNIX ファイルシステムでの DLL の探索順序

z/OS UNIX ファイルシステムを使用する場合は、CALL ステートメントの DLL 参照を解決するための探索順序は、言語環境の POSIX ランタイム・オプションの設定によって異なります。

POSIX ランタイム・オプションが ON の場合の探索順序は、次のとおりです。

1. ランタイム環境は、z/OS UNIX ファイルシステムに DLL がないかどうか探します。LIBPATH 環境変数が設定されていると、リストされたそれぞれのディレクトリーが探索されます。これが設定されていない場合は、現行ディレクトリーだけが探索されます。z/OS UNIX ファイルシステムでの DLL の探索には、大/小文字の区別があります。
2. z/OS UNIX ファイルシステムで DLL が見つからないと、ランタイム環境は、呼び出し側の MVS ロード・ライブラリー探索順序から DLL をロードすることを試みます。この場合、DLL 名は 8 文字以下でなければなりません。この探索では、ランタイム環境は自動的に DLL 名を大文字に変換します。

POSIX ランタイム・オプションが OFF に設定されていると、探索順序は逆になります。

1. ランタイム環境は、呼び出し側のロード・ライブラリー用の探索順序から DLL をロードすることを試みます。
2. このロード・ライブラリーから DLL をロードできない場合、ランタイム環境は z/OS UNIX ファイルシステムから DLL をロードすることを試みます。

関連タスク

[643 ページの『DLL での CALL ID の使用』](#)

関連参照

言語環境プログラミング・リファレンス (POSIX)

DLL リンケージと動的呼び出しの併用

複数の別個にバインドされたモジュールとして構築されるアプリケーション (つまり 言語環境 エンクレーブ) の場合、各モジュールは、動的呼び出しリンケージまたは DLL リンケージを使用して呼び出すことができます。ある 1 つのモジュールに対しては、そのモジュールに入るために 1 つの形式のリンケージのみを使用してください。ただし、呼び出し元には CALL 両方のリンク タイプを持つステートメントで、異なるモジュールを呼び出します。

このタスクについて

DLL リンケージとは、DLL オプションおよび NODYNAM 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呼び出し、LP (64) オプションおよび NODYNAM 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呼び出し、または DLL を指定する CALLINTERFACE コンパイラ・ディレクティブを使用した呼び出しを指します。このような呼び出しでは、呼び出し先サブプログラムは、別個モジュール内のエクスポートされた名前に解決されます。DLL リンケージが、別個のモジュール内に定義されているメソッドの呼び出しを指すこともあります。

コンパイル単位内では、特定のプログラムを呼び出す際に、「動的」、「DLL」、「静的」のうちいずれか 1 つの呼び出し規約のみを使用する必要があります。1 つのプログラムが複数の異なる呼び出し規約を使用して呼び出された場合、コンパイラはこの状況を診断し、そのプログラムで検出された最初の呼び出しステートメントと同じ規約をすべての呼び出しに強制します。

プログラムでは、動的呼び出しリンケージと DLL リンケージの両方を持つ CALL ステートメントを使用できます。これは、CALLINTERFACE コンパイラ指示を使用して当該呼び出しのリンケージ・タイプを指定することによって行うことができます。DLL アプリケーションのコンポーネントはすべて同じ AMODE でなければなりません。COBOL 動的呼び出しによって通常提供される自動 AMODE 切り替えは、DLL リンケージでは利用できません。DLL リンケージを使用して呼び出されるプログラムを取り消すことはできません。

DLL アプリケーションのコンポーネントはすべて同じ AMODE でなければなりません。COBOL 動的呼び出しによって通常提供される自動 AMODE 切り替えは、DLL リンケージでは利用できません。

関連概念

[639 ページの『ダイナミック・リンク・ライブラリー \(DLL\)』](#)

関連タスク

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

[641 ページの『DLL のリンク』](#)

[645 ページの『DLL でのプロシージャ・ポインターまたは関数ポインターの使用』](#)

[646 ページの『非 DLL からの DLL の呼び出し』](#)

関連参照

[396 ページの『DLL』](#)

[401 ページの『EXPORTALL』](#)

CALLINTERFACE (Enterprise COBOL for z/OS 言語解説書)

DLL でのプロシージャ・ポインターまたは関数ポインターの使用

DLL と非 DLL の両方を含む実行単位では、プロシージャ・ポインターおよび関数ポインター・データ項目の使用には注意してください。

このタスクについて

の SETprocedure-pointer-1TO ENTRY エントリー名声明、SETfunction-pointer-1TO ENTRY エントリー名声明、そして CALL ステートメントには、それに関連付けられた呼び出しリンク タイプがあります。呼び出しリンケージ・タイプは、そのステートメントで有効なコンパイラ・オプションおよび CALLINTERFACE 指示によって決まります。DLL オプションを使用してコンパイルされたプログラムで

は、デフォルトの呼び出しリンケージ・タイプは DLL です。そうでない場合、リンケージ・タイプは非 DLL です。このデフォルトは、CALLINTERFACE 指示によってオーバーライドできます。

リンケージ・タイプが非 DLL の SET ステートメントによって設定されたプロシージャ・ポインター・データ項目または関数ポインター・データ項目を、リンケージリンケージ・タイプが DLL の CALL ステートメントで使用してはいけません。リンケージ・タイプが DLL で *entry-name* が ID である SET ステートメントでは、NODYNAM オプションが有効な場合、*entry-name* ID 値は、DLL モジュールからエクスポートされたエントリ・ポイント名を参照していなければなりません。DLL モジュール名は、エクスポートされる入り口点の名前と一致していなければなりません。この場合、次の点にも注意してください。

- ID に含まれているプログラム名は、CALL ステートメントを含むプログラムの PGMNAME (COMPAT | LONGUPPER | LONGMIXED) コンパイラ・オプションの設定に従って解釈されます。
- ターゲット DLL からエクスポートされるプログラム名は、ターゲット・プログラムのコンパイル時に使用された PGMNAME オプションの設定に従って解釈されます。
- z/OS UNIX ファイルシステムにおけるターゲット DLL の探索には、大/小文字の区別があります。
- ターゲット DLL が PDSE メンバーである場合、DLL メンバー名は 8 文字以下でなければなりません。PDSE メンバーとして DLL を探索するために、名前は自動的に大文字に変換されます。

関連タスク

[643 ページの『DLL での CALL ID の使用』](#)

[617 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』](#)

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

[641 ページの『DLL のリンク』](#)

関連参照

[396 ページの『DLL』](#)

[401 ページの『EXPORTALL』](#)

CALLINTERFACE (Enterprise COBOL for z/OS 言語解説書)

非 DLL からの DLL の呼び出し

NODLL オプションを使用してコンパイルされた COBOL プログラムから DLL を呼び出すことができますが、制限があります。

このタスクについて

以下の方法を使用して、DLL リンケージに確実に従うようにすることができます。

- COBOL の非 DLL プログラムから呼び出したい COBOL DLL プログラムを、メインプログラムを含むプログラム・オブジェクトに入れます。COBOL の非 DLL プログラムから静的呼び出しを使用して、COBOL DLL プログラムを呼び出します。
メインプログラムを含むプログラム・オブジェクトに含まれる COBOL DLL プログラムからは、他の DLL 内の COBOL DLL プログラムを呼び出せます。
- COBOL DLL プログラムを DLL に入れ、それを COBOL 非 DLL プログラムから CALL *function-pointer* を使用して呼び出します (*function-pointer* には、ターゲット・プログラムの関数記述子を設定します)。DLL 内におけるプログラムの関数記述子のアドレスを入手するには、`dllload` および `dllqueryfn` を使用する C ルーチン呼び出します。

[646 ページの『例: 非 DLL からの DLL の呼び出し』](#)

関連タスク

[617 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』](#)

例: 非 DLL からの DLL の呼び出し

次の例では、DLL に含まれていない COBOL プログラム (COBOL1) から、DLL 内の COBOL プログラム (DLL OOC05R 中のプログラム ooc05R) を呼び出す方法が示されています。

```

CBL NODYNAM
  IDENTIFICATION DIVISION.
  PROGRAM-ID. 'COBOL1'.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
  DATA DIVISION.
  FILE SECTION.
  WORKING-STORAGE SECTION.
  01 DLL-INFO.
      03 DLL-LOADMOD-NAME PIC X(12).
      03 DLL-PROGRAM-NAME PIC X(160).
      03 DLL-PROGRAM-HANDLE FUNCTION-POINTER.
  77 DLL-RC PIC S9(9) BINARY.
  77 DLL-STATUS PIC X(1) VALUE 'N'.
      88 DLL-LOADED VALUE 'Y'.
      88 DLL-NOT-LOADED VALUE 'N'.

  PROCEDURE DIVISION.

      IF DLL-NOT-LOADED
      THEN
      *      Move the names in. They must be null terminated.
          MOVE Z'00C05R' TO DLL-LOADMOD-NAME
          MOVE Z'ooc05r' TO DLL-PROGRAM-NAME

      *      Call the C routine to load the DLL and to get the
      *      function descriptor address.
          CALL 'A1CCDLGT' USING BY REFERENCE DLL-INFO
                                BY REFERENCE DLL-RC

          IF DLL-RC = 0
          THEN
              SET DLL-LOADED TO TRUE
          ELSE
              DISPLAY 'A1CCDLGT failed with rc = '
                      DLL-RC
              MOVE 16 TO RETURN-CODE
              STOP RUN
          END-IF
      END-IF

      *      Use the function pointer on the call statement to call the
      *      program in the DLL.
      *      Call the program in the DLL.
          CALL DLL-PROGRAM-HANDLE

      GOBACK.

#include <stdio.h>
#include <dll.h>
#pragma linkage (A1CCDLGT,COBOL)

typedef struct dll_lm {
    char    dll_loadmod_name[(12)];
    char    dll_func_name[(160)];
    void    (*fptr) (void); /* function pointer */
} dll_lm;

void A1CCDLGT (dll_lm *dll, int *rc)
{
    dllhandle *handle;
    void (*fptr1)(void);
    *rc = 0;
    /* Load the DLL */
    handle = dllload(dll->dll_loadmod_name);
    if (handle == NULL) {
        perror("A1CCDLGT failed on call to load DLL./n");
        *rc = 1;
        return;
    }

    /* Get the address of the function */
    fptr1 = (void (*)(void))
            dllqueryfn(handle,dll->dll_func_name);
    if (fptr1 == NULL) {
        perror("A1CCDLGT failed on retrieving function./n");
        *rc = 2;
        return;
    }
}

```



```

}
/* Return the function pointer          */
dll->fptr = fptr1;
return;
}

```

C/C++ プログラムでの COBOL DLL の使用

COBOL の DLL サポートは、z/OS C/C++ COBOL を除く製品 EXTERNAL データ。特に、COBOL アプリケーションは、C/C++ DLL からエクスポートされた関数を呼び出すことができ、C/C++ アプリケーションは、COBOL DLL からエクスポートされた COBOL プログラムを呼び出すことができます。

このタスクについて

EXTERNAL 属性で宣言された COBOL データ項目は、DLL サポートとは無関係です。これらのデータ項目は、プログラムが DLL 内にあるかどうかに関係なく、それらを宣言する実行単位内の任意の COBOL プログラムから名前によってアクセスすることができます。

COBOL オプション DLL、RENT、そして EXPORTALL 同じように機能します C/C++ DLL、RENT、そして EXPORTALL オプション。(DLL オプションは C にしか適用されません。) ただし、C/C++ コンパイラーは、デフォルトでは、DLL 対応コードを生成します。

C/C++ DLL 関数ポインターを COBOL に渡し、C/C++ 関数ポインターを関数ポインター・データ項目として受け取り、それを COBOL 内で使用することができます。次の例は、サービスへの関数ポインターを戻す C 関数への COBOL 呼び出しと、そのサービスへの COBOL 呼び出しを示しています。

```

Identification Division.
Program-id. Demo.
Data Division.
Working-Storage section.
01 fp usage function-pointer.
Procedure Division.
    Call "c-function" returning fp.
    Call fp.

```

関連タスク

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

[641 ページの『DLL のリンク』](#)

関連参照

[396 ページの『DLL』](#)

[401 ページの『EXPORTALL』](#)

[435 ページの『RENT』](#)

EXTERNAL 節 (*Enterprise COBOL for z/OS 言語解説書*)

CALLINTERFACE (*Enterprise COBOL for z/OS 言語解説書*)

OO COBOL アプリケーションでの DLL の使用

DLL、THREAD、RENT、および DBCS コンパイラー・オプションを使用して、各 COBOL クラス定義をコンパイルしてから、RENT バインダー・オプションを使用して、それを別個の DLL モジュールにリンク・エディットしなければなりません。

このタスクについて

関連タスク

[321 ページの『第 17 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』](#)

[640 ページの『DLL を作成するためのプログラムのコンパイル』](#)

[641 ページの『DLL のリンク』](#)

関連参照

[396 ページの『DLL』](#)

[453 ページの『THREAD』](#)

[435 ページの『RENT』](#)

[392 ページの『DBCS』](#)

第 32 章 マルチスレッド化のための COBOL プログラムの準備

バッチ、TSO、IMS、または z/OS ユニックス において、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。

このタスクについて

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・オプションを使用してコンパイルを行います。

COBOL はプログラム・スレッドの管理を直接サポートしません。ただし、THREAD コンパイラ・オプションでコンパイルした COBOL プログラムを、マルチスレッド・アプリケーション・サーバ、スレッドの作成に C/C++ ドライバ・プログラムを使用するアプリケーション、Java と相互運用し Java スレッドを使用するプログラム、PL/I タスク処理を使用するアプリケーションで実行することはできる。つまり、他のプログラムは、プロセス内の複数のスレッドで、または単一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 COBOL プログラムを実行するといった方法で、COBOL プログラムを呼び出すことができます。スレッド化されたアプリケーションは、単一の言語環境・エンクレーブ内で実行される必要があります。

LOCAL-STORAGE または WORKING-STORAGE の選択: マルチスレッド化プログラムは再帰的プログラムとしてコーディングしなければならないので、データの永続性は、再帰的プログラムの永続性となります。

- LOCAL-STORAGE SECTION 内のデータ項目は、プログラム起動のインスタンスごとに自動的に割り振られます。あるプログラムが複数のスレッドにおいて同時に実行されると、起動ごとに個別の LOCAL-STORAGE データのコピーを使用します。
- WORKING-STORAGE SECTION のデータ項目は、プログラムごとに一度割り振られるので、同じプログラムの呼び出しでは常に、最後に使用されたときの状態で提供されます。

個々のプログラム呼び出しインスタンスに分離したいデータについては、LOCAL-STORAGE SECTION。一般的に、この選択はスレッド化プログラム内の作業データに適しています。データを WORKING-STORAGE に定義し、プログラムがそのデータの内容を変更する場合には、以下のアクションのどちらかを実行する必要があります。

- WORKING-STORAGE 内のデータに 複数のスレッド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。
- データに別々のスレッドから同時にアクセスする場合には、適切な直列化コードを書き込みます。

関連概念

[651 ページの『マルチスレッド化』](#)

関連タスク

[653 ページの『マルチスレッド化サポートのための THREAD の選択』](#)

[653 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』](#)

[653 ページの『マルチスレッド化されたプログラムの終了』](#)

[654 ページの『マルチスレッド化によるファイルの処理』](#)

[656 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

[453 ページの『THREAD』](#)

PROGRAM-ID 段落 (Enterprise COBOL for z/OS 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。

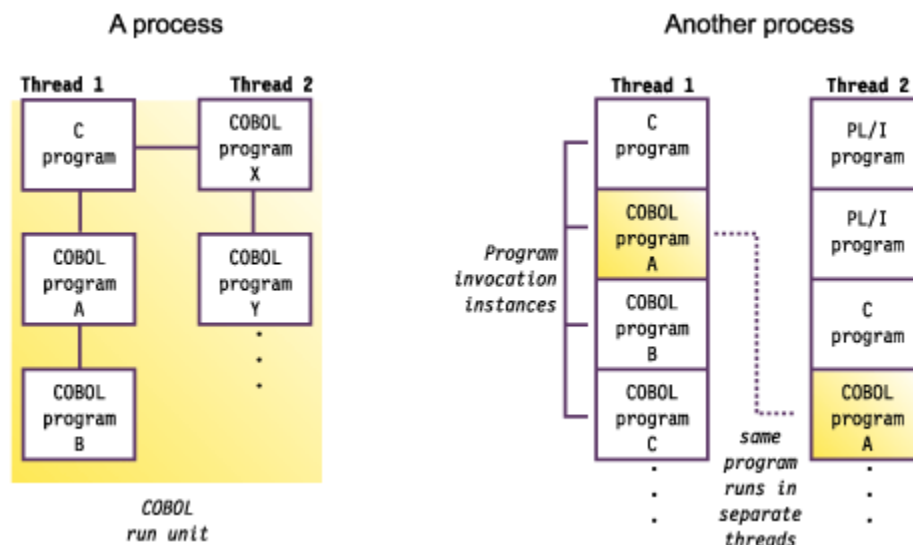
オペレーティング・システムおよびマルチスレッド化アプリケーションは、プロセス内の実行フローを処理することができます。実行フローとは、プログラムのすべて、または一部が実行時に発生する一連のイベントです。1つのプロセス内で実行されるプログラムはリソースを共用することができます。プロセスは操作することができます。例えば、システムがプロセスの実行に使用する時間において、プロセスに高い、または低い優先順位を持たせることができます。

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、実行プログラムの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、別個の COBOL プログラムと非 COBOL プログラムとの間で制御権が移動します。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。



関連概念

言語環境プログラミングガイド (プログラム管理モデル、
基本的な概念: スレッド)

関連タスク

[653 ページの『マルチスレッド化サポートのための THREAD の選択』](#)

[653 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』](#)

[653 ページの『マルチスレッド化されたプログラムの終了』](#)

[654 ページの『マルチスレッド化によるファイルの処理』](#)

[656 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

[453 ページの『THREAD』](#)

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。使用 THREAD プログラムが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。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。

このタスクについて

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。また、コンパイルには RENT コンパイラオプションを使用してリンクし、RENT オプションのバインダー（リンケージエディター）。

オブジェクト指向 (OO) のクライアントとクラスをコンパイルする場合は、THREAD オプションを使用します。

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: プログラムをコンパイルする前に THREAD コンパイラオプションを指定するには、RECURSIVE フレーズの PROGRAM-ID 段落。指定しなければ、エラーが発生します。

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用』](#)

[321 ページの『z/OS ユニックスのもとで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

[453 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。

このタスクについて

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とき、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ときは、呼び出されるプログラムは初期状態にあります。

CANCEL ステートメントで名前を付けるプログラムが、どのスレッドにおいてもアクティブでないことを確認してください。アクティブ・プログラムを取り消そうとした場合には、重大度 3 の言語環境条件が発生します。

スレッド化アプリケーションに事前初期設定が必要である場合には、言語環境・サービス (CEEPIPI インターフェース) を使用します。事前初期化に COBOL 固有のインターフェースを使用することはできません (実行時オプション RTEREUS) を使用してコンパイルされたプログラムから再利用可能な環境を確立します。THREAD オプション。

関連概念

言語環境プログラミングガイド (終了中に起こる事象:
エンクレーブの終了)

関連タスク

[653 ページの『マルチスレッド化されたプログラムの終了』](#)

[604 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

マルチスレッドプログラムを終了するには、GOBACK、EXIT PROGRAM、または STOP RUN。

このタスクについて

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。そのスレッドがあるエンクレーブ内の初期スレッドである場合には、そのエンクレーブは終了します。

使用 EXIT PROGRAM あなたと同じように GOBACK ただし、メイン プログラムからは効果がありません。

言語環境のエンクレーブ全体を終了し、メインプログラム (オペレーティング・システムであると考えられます) の呼び出し側に制御を戻すには、STOP RUN を使用します。エンクレーブ内で実行中のすべてのスレッドが終了します。

関連概念

言語環境プログラミングガイド (終了中に起こる事象:
エンクレーブの終了)

関連タスク

[604 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ファイルの処理

スレッド化アプリケーションでは、QSAM、VSAM、および行順次ファイルでの入出力のために COBOL ステートメントを コーディングすることができます。

このタスクについて

各ファイル定義 (FD) には暗黙のシリアライゼーション・ロックがあります。このロックは、以下のステートメントの実行に関連付けられた入力または出力操作の間に、自動シリアライゼーション・ロジックとともに使用されます。

- OPEN
- CLOSE
- READ
- WRITE
- REWRITE
- START
- DELETE

また、自動シリアライゼーションは、以下のステートメントに関連付けられた 暗黙の MOVE の場合にも発生します。

```
WRITE record-name FROM identifier  
READ file-name INTO identifier
```

自動シリアライゼーションは、以下の条件付き句内で指定されるステートメントには適用されません。

- AT END
- NOT AT END
- INVALID KEY
- NOT INVALID KEY
- AT END-OF-PAGE
- NOT AT END-OF-PAGE

関連概念

[655 ページの『ファイル定義 \(FD\) ストレージ』](#)

関連タスク

[190 ページの『QSAM ファイルのクローズ』](#)

[217 ページの『VSAM ファイルのクローズ』](#)

[270 ページの『ERROR 宣言のコーディング』](#)

[655 ページの『マルチスレッド化によるファイル・アクセスのシリアライズ』](#)

ファイル定義 (FD) ストレージ

すべてのプログラム起動の際に、ファイル定義に関連付けられたストレージ (FD レコードや SAME RECORD AREA 節に関連付けられたレコード域) は割り振られ、最後に使われた状態で使用可能です。

実行のすべてのスレッドはこのストレージを 共有します。OPEN、CLOSE、READ、WRITE、REWRITE、START、および DELETE ステートメントの実行中は、このストレージの自動シリアライゼーションに依存することができますが、これらのステートメントの使用の間は依存することはできません。

関連タスク

[655 ページの『マルチスレッド化によるファイル・アクセスのシリアライズ』](#)

マルチスレッド化によるファイル・アクセスのシリアライズ

自動シリアライゼーションの利点を最大に活用し、固有のシリアライゼーション・ロジックの明示的書き込みを避けるには、スレッド化プログラム内の ファイルにアクセスするときに、推奨されるいずれかのファイル編成および使用パターンを使用します。

このタスクについて

以下のいずれかのファイル編成を使用する。

- 順次編成
- 行順次編成
- 順次アクセスによる相対編成
- 順次アクセスによる索引編成

入力に次のパターンを使用します。

```
OPEN INPUT fn
  .
  .
  .
READ fn INTO local-storage-item
  .
  .
  .
* Process the record from the local-storage item
  .
  .
  .
CLOSE fn
```

出力に次のパターンを使用します。

```
OPEN OUTPUT fn
  .
  .
  .
* Construct output record in local-storage item
  .
  .
  .
WRITE rec FROM local-storage-item
  .
  .
  .
CLOSE fn
```

他の使用パターンの場合には、以下のアクションのいずれかを 実行します。

- アプリケーション・ロジックの安全を検証します。プログラムの 2 つの インスタンスが別々のスレッドで決して同時にアクティブにならないことを確認します。
- POSIX サービスに対する呼び出しを使用して、明示的シリアライゼーション・ロジックをコーディングします。

複数のスレッドからファイルにアクセスするときにシリアル化の問題を回避するには、ファイルに関連付けられたデータ項目（ファイルステータスデータ項目やキー引数など）を LOCAL-STORAGE SECTION。

656 ページの『例: マルチスレッド化によるファイル入出力の使用パターン』

関連タスク

582 ページの『UNIX/POSIX API の呼び出し』

例: マルチスレッド化によるファイル入出力の使用パターン

以下の例では、マルチスレッド化アプリケーションにおいてファイル入出力の推奨使用パターンから逸脱した際の明示的シリアライゼーション・ロジックの必要性について説明します。また、これらの例では、シリアライゼーションを適切に処理できない結果として発生する予期しない動作についても説明します。

それぞれの例において、サンプル操作を含むプログラムの 2 つのインスタンスが 1 つの実行単位内で、2 つの異なるスレッド上で実行されています。

```
READ F1
      .
      .
      .
REWRITE R1
```

上記の例で、2 番目のスレッドは、最初のスレッド上で READ ステートメントが実行された後に（ただし、最初のスレッド上で REWRITE ステートメントが実行される前に）READ ステートメントを実行すると考えられます。REWRITE ステートメントは、意図したレコードの更新は行わない可能性があります。目的の結果が確実に出るようにするには、明示的シリアライゼーション・ロジックを書き込みます。

```
READ F1
      .
      .
      .
* Process the data in the FD record description entry for F1
      .
      .
      .
```

上記の例で、2 番目のスレッドは、最初のスレッドが FD レコード記述項目内のレコードをまだ処理中に READ ステートメントを実行すると考えられます。2 番目の READ ステートメントは、最初のスレッドが処理中であるレコードをオーバーレイする可能性があります。この問題を回避するには、次の推奨技法を使用してください。

```
READ F1 INTO LOCAL-STORAGE-item
```

その他のケース: 後に READ NEXT が続く START や、後に DELETE が続く READ など、一連の関連した入出力操作を必要とするその他の使用パターンについても同様に考慮してください。適切な手順を実行して、ファイル入出力の正しい処理を確実に行います。

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。

このタスクについて

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、また推奨使用パターンを持つサポート対象の COBOL ファイル入出力ステートメントからも使用できます。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。

CICS: マルチスレッドアプリケーションを実行できないで CICS。CICS では、THREAD オプションを使用してコンパイルし、複数のスレッドまたは PL/I タスクを持たないアプリケーションの一部である COBOL プログラムを実行できます。

再帰的: マルチスレッド化アプリケーション内のプログラムは再帰的プログラムとしてコーディングする必要があります。したがって、ネストされたプログラムをコーディングしないなど、再帰的プログラムに適用される制限やプログラミング上の考慮事項を忠実に守る必要があります。

再入可能性: マルチスレッド化プログラムは、RENT コンパイラー・オプションを使用してコンパイルし、それらをバインダー (リンケージ・エディター) の RENT オプションとリンクしなければなりません。

POSIX および PL/I: マルチスレッド化アプリケーションで POSIX スレッドを使用する場合には、言語環境・ランタイム・オプション POSIX(ON) を指定しなければなりません。アプリケーションが PL/I タスキングを使用する場合には、POSIX(OFF) を指定しなければなりません。POSIX スレッドと PL/I タスクを同一のアプリケーションに混在させることはできません。

PL/I タスク: 複数の PL/I タスクを含むアプリケーションに COBOL プログラムを組み込む場合の指針を以下に示します。

- 複数の PL/I タスクで実行する COBOL プログラムはすべて、THREAD オプションを指定してコンパイルします。NOTHREAD オプションを指定してコンパイルされた COBOL プログラムが 1 つでもあると、どの COBOL プログラムも単一の PL/I タスクでしか実行できなくなります。
- THREAD オプションを指定してコンパイルされた COBOL プログラムは、1 つ以上の PL/I タスクから呼び出すことができます。しかし、PL/I プログラムからの COBOL プログラムの呼び出しでは、TASK オプションや EVENT オプションを指定することができません。PL/I タスク呼び出しでは、最初に PL/I のプログラムまたは関数を呼び出し、そこから COBOL プログラムを呼び出す必要があります。このような間接呼び出しが必要になるのは、TASK オプションまたは EVENT オプションを含む PL/I の CALL ステートメントのターゲットとして COBOL プログラムを指定することができないからです。
- COBOL プログラムから STOP RUN ステートメントを発行したり、PL/I プログラムから STOP ステートメントを発行すると、言語環境のエンクレープ全体 (すべてのタスク実行を含む) が終了してしまいます。
- PL/I タスクを含む実行単位では、明示的な POSIX スレッド化 (pthread_create() の呼び出し) をコーディングしないでください。

C および言語環境準拠アセンブラー: マルチスレッド COBOL プログラムは、C プログラムおよび言語環境準拠アセンブラー・プログラムがマルチスレッド実行用に適切にコーディングされている場合、同じ実行単位内でそれらのプログラムと結合することができます。

AMODE: マルチスレッド化アプリケーションを AMODE 31 で実行しなければなりません。AMODE 24 を複数のスレッドまたは PL/I タスクを持たないアプリケーションの一部として、THREAD オプションでコンパイルした COBOL プログラムを実行することができます。

非同期シグナル: スレッド化アプリケーションにおいて、COBOL プログラムは、非同期シグナルまたは割り込みによって割り込まれる場合があります。プログラムにそのような割り込みを容認できないロジックが含まれている場合には、そのロジックが継続する間はそうした割り込みを使用不可にする必要があります。C/C++ 関数を呼び出して、シグナル・マスクを適切に設定します。

古い COBOL プログラム: マルチスレッド・アプリケーションの複数のスレッドで COBOL プログラムを実行するには、その COBOL プログラムを Enterprise COBOL でコンパイルし、THREAD オプションを使用する必要があります。古いコンパイラーでコンパイルされたプログラムを含むアプリケーションは、1 つのスレッド上でのみ実行してください。

IGZETUN および IGZEOPT: THREAD オプションでメインプログラムをコンパイルした、アプリケーションに対して、モジュール IGZETUN (ストレージ・チューニング用) または IGZEOPT (ランタイム・オプション用) を使用してはなりません。これらの CSECT は無視されます。

UPSI スレッド: アプリケーション内のすべてのプログラムおよびすべてのスレッドは UPSI スイッチの単一コピーを共有します。スレッド化アプリケーションでスイッチを変更する場合には、適切なシリアライゼーション・ロジックをコーディングする必要があります。

関連タスク

[617 ページの『再帰呼び出しの実行』](#)

[655 ページの『マルチスレッド化によるファイル・アクセスのシリアライズ』](#)

[XL C/C++ プログラミング・ガイド \(z/OS UNIX システム・サービス](#)

[・アプリケーションでのスレッドの使用\)](#)

S

第 33 章 ユーザー定義関数の使用

COBOL ユーザー定義関数を使用すると、FUNCTION キーワードを使用して呼び出すことができる独自の関数定義を作成できます。ユーザー定義関数は常に再帰的であり、それ自体を呼び出すことができます。

このタスクについて

ユーザー定義関数は、静的、動的、またはダイナミック リンク ライブラリ (DLL) 経由で呼び出すことができます。FUNCTION-ID 段落の ENTRY-INTERFACE 句は、関数に関連付けられた呼び出しの種類を決定します。CALLINTERFACE ディレクティブは、ユーザー定義関数のインターフェース規則には影響しません。

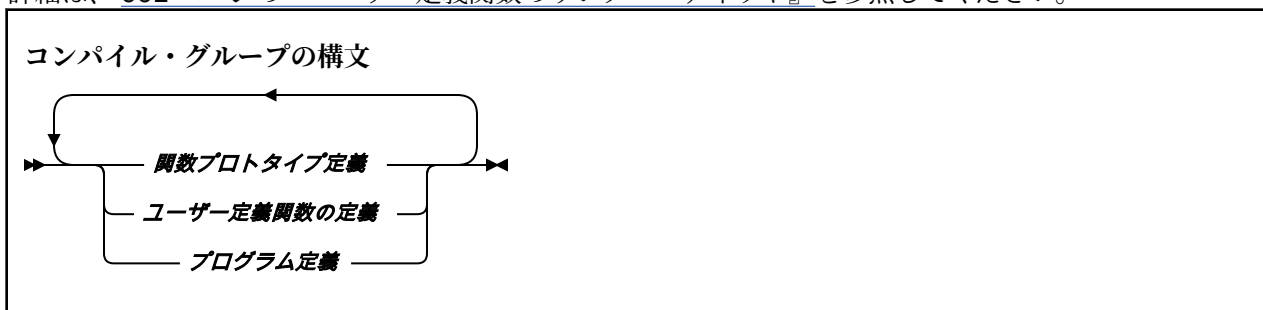
ユーザー定義関数への関数ポインターはサポートされていません。ユーザー定義関数が CALL 文によって呼び出された場合、その動作は予測不可能である。

ユーザー定義関数の作成

ユーザー定義関数定義は、独立したコンパイル・ユニットであり、プログラム、メソッド、他のユーザー定義関数などの他のコンパイル・ユニットの中に入れ子にすることはできません。ユーザー定義関数を別のファイル、つまり別のバッチ コンパイルでコーディングし、関数プロトタイプを使用して呼び出し元プログラムから関数を呼び出すことをお勧めします。同じバッチ コンパイルでプロトタイプなしでユーザー定義関数をコーディングすることを選択した場合、ユーザー定義関数の定義は、それを呼び出すプログラムまたは関数の前に、コンパイル グループ内で最初に出現する必要があります。プログラムのエントリポイントがデフォルトのエントリポイントである場合、これを行うには特別なリンク編集手順が必要です。同じバッチ コンパイルでプロトタイプを使用してユーザー定義関数をコーディングすることを選択した場合、プロトタイプが、それを呼び出す最初のプログラムまたは関数の前に表示されるか、関数が複数のプログラムから呼び出される場合はコピーブックから組み込まれている限り、ユーザー定義関数は、それを呼び出すさまざまなプログラムまたは関数の間で任意の順序で表示できます。バッチプログラムの最初のプログラムまたはユーザー定義関数が、デフォルトのエントリーポイントとして選択されるため、プログラムの順序を注意深く選択しないと、特別なリンク編集ステップが必要になる場合がある。

このタスクについて

詳細は、661 ページの『ユーザー定義関数のリンク・エディット』を参照してください。



注：現在、SQL、CICS、そして JAVAIO ユーザー定義関数の定義をプログラムが使用する前に組み込む必要があるため、これらの機能はユーザー定義関数では使用できません。

ユーザー定義関数定義のコーディング

ユーザー定義関数には、IDENTIFICATION DIVISION 内に FUNCTION-ID 段落が含まれている必要があります。END FUNCTION キーワードは、ユーザー定義関数定義の終わりを示します。ユーザー定義関数には、関数の戻り項目を示す PROCEDURE DIVISION ヘッダーの RETURNING 句が含まれている必要があります。

このタスクについて

ユーザー定義関数のゼロ個以上のパラメーターが、PROCEDURE DIVISION ヘッダーの USING 句に現れることがあります。パラメーターには、BY REFERENCE または BY VALUE を渡すことができます。

以下の例は、「docalc」というユーザー定義関数定義を示しています。

```
Identification division.  
  Function-id. docalc.  
Data division.  
  Linkage section.  
    1 kind pic x(3).  
    1 argA pic 999.  
    1 argB pic v999.  
    1 res pic 999v999.  
Procedure division  
  using by reference kind argA argB  
  returning res.  
  if kind equal "add" then  
    compute res = argA + argB  
  end-if  
  goback.  
End function docalc.
```

関連参照

FUNCTION-ID 段落 (*Enterprise COBOL for z/OS 言語解説書*)

ユーザー定義関数定義のコンパイル

コンパイラーは、ユーザー定義関数定義を検出すると、その定義から自動的に特性を抽出します。これらの特性が組み合わさって関数プロトタイプを形成します。収集される特性には、関数名、戻り項目データの説明エントリ、およびパラメータ データの説明エントリなどがあります。

このタスクについて

プログラムまたはユーザー定義関数定義内でコンパイラーが関数呼び出しに遭遇すると、コンパイラーは引数と戻り項目が適切かどうかを判断します。引数は対応する形式パラメータと照合され、適合性チェック。*Enterprise COBOL for z/OS 言語解説書*のユーザー定義関数用のパラメーターの規格適合を参照してください。

ユーザー定義関数の呼び出し

ユーザー定義関数は、組み込み関数を呼び出すことができるほとんどの場所で呼び出すことができます。

このタスクについて

FUNCTION キーワードを指定せずにユーザー定義関数を呼び出すには、REPOSITORY 段落に関数名を指定します。詳しくは、*Enterprise COBOL for z/OS 言語解説書*の REPOSITORY 段落を参照してください。

以下の例は、「docalc」というユーザー定義関数定義と、関数呼び出しを含むプログラムを含むコンパイル・グループを示しています。

```
Identification division.  
  Function-id. docalc.  
Data division.  
  Linkage section.  
    1 kind pic x(3).  
    1 argA pic 999.  
    1 argB pic v999.  
    1 res pic 999v999.  
Procedure division  
  using by reference kind argA argB  
  returning res.  
  if kind equal "add" then  
    compute res = argA + argB  
  end-if  
  goback.  
End function docalc.
```

```

Identification division.
  Program-id. 'mainprog'.
Environment division.
  Configuration section.
    Repository.
      function docalc.
Data division.
  Working-storage section.
    1 result pic 999v999 usage display.
Procedure division.
  compute result = docalc("add" 10 0.23)
  display "hello from mainprog, result=" result
  goback.
End program 'mainprog'.

```

このプログラムの実行後、出力は以下のようになります。

```
hello from mainprog, result=010230
```

ユーザー定義関数の外部名の指定

ユーザー定義関数は、オプションで、IDENTIFICATION DIVISION の FUNCTION-ID 段落に外部名を使用して指定することができます。関数の呼び出しに使用される名前とは異なる、稼働環境に公開される外部名を指定することができます。外部名が指定されている場合、PGMNAME オプションは外部名に適用されます (内部名には適用されません)。

このタスクについて

外部名の指定について詳しくは、*Enterprise COBOL for z/OS* 言語解説書の *FUNCTION-ID* 段落を参照してください。PGMNAME オプションについて詳しくは、[432 ページの『PGMNAME』](#)を参照してください。

外部名は、稼働環境に公開される名前になります。COBOL プログラム内で関数を呼び出すために使用することはできません。

以下の例は、「XYZCALC」という外部名を持つ「docalc」のユーザー定義関数を示しています。

```

Identification division.
  Function-id. docalc as 'XYZCALC'.
Data division.
  Linkage section.
    1 kind pic x(3).
    1 argA pic 999.
    1 argB pic v999.
    1 res pic 999v999.
Procedure division
  using by reference kind argA argB
  returning res.
  if kind equal "add" then
    compute res = argA + argB
  end-if
  goback.
End function docalc.

```

ユーザー定義関数のリンク・エディット

コンパイル・グループに複数のユーザー定義関数やプログラムが含まれている場合、バインダーが最初に見つけたエントリー・ポイントがデフォルトのエントリー・ポイントとなる。これは、ユーザー定義関数が呼び出しプログラムよりも先に現れる場合に問題となる。この場合、LP(32) オプションを使用する AMODE 31 (31 ビット) コンパイルでは、目的のプログラム・エントリ・ポイントにバインダー ENTRY 文を指定する必要があります。AMODE 64 (64 ビット) コンパイルで LP(64) オプションを使用する場合、バインダーに ENTRY CELQSTRT ステートメントを指定する必要があります。バインダーは最初に見つかったプログラムのエントリー・ポイントを選択し、ユーザー定義関数のエントリー・ポイントはスキップします。

このタスクについて

以下の JCL ステップの例を参照してください。

```
//LKED1      EXEC PGM=IEWL,
// PARM='LIST,MAP '
//SYSLIB     DD DSN=CEE.SCEELKED,DISP=SHR
//OBJECT     DD DSN=&&OBJECT,DISP=(SHR,PASS)
//SYSLIN     DD *
             INCLUDE OBJECT(OBJ1)
             ENTRY MAINPROG
/*
```

引数 BY CONTENT をユーザー定義関数に渡します

ユーザー定義関数定義の仮パラメーターは BY REFERENCE または BY VALUE としてのみ指定できますが、CONTENT-OF 組み込み関数を使用して、ユーザー定義関数 BY CONTENT に引数を効果的に渡すことができます。これを行うには、関数定義で仮パラメーターを BY REFERENCE として指定する必要があります。ユーザー定義関数呼び出しでは、CONTENT-OF 組み込み関数を使用して、効果的に渡されるようにしたい各引数をラップします BY CONTENT。

このタスクについて

以下の関数定義の例と、関数の呼び出しを含むプログラムを参照してください。

```
Identification division.
  Function-id. docalc.
Data division.
  Linkage section.
    1 kind pic x(3).
    1 argA pic 999.
    1 argB pic v999.
    1 res pic 999v999.
Procedure division
  using by reference kind argA argB
  returning res.
  if kind equal "add" then
    compute res = argA + argB
  end-if
  goback.
End function docalc.
Identification division.
  Program-id. 'mainprog'.
Environment division.
  Configuration section.
    Repository.
      function content-of intrinsic
      function docalc.
Data division.
  Working-storage section.
    1 result pic 999v999 usage display.
    1 arg-1 pic x(3).
    1 arg-2 pic 999.
    1 arg-3 pic v999.
Procedure division.
  move "add" to arg-1
  move 10 to arg-2
  move 0.23 to arg-3
* arg-1 is effectively BY CONTENT
* arg-2 is BY REFERENCE
* arg-3 is BY REFERENCE
  compute result = docalc(content-of(arg-1) arg-2 arg-3)
  display "hello from mainprog, result=" result
  goback.
End program 'mainprog'.
```

CONTENT-OF 組み込み関数は、入力引数の一時コピーを作成します。その後、一時コピーが BY REFERENCE ユーザー定義関数に提供され、ユーザー定義関数定義内の仮パラメーターに対する変更は、効果的に一時コピーに対して行われます。

関連参照

CONTENT-OF 組み込み関数 (*Enterprise COBOL for z/OS* 言語解説書)

第 34 章 関数プロトタイプの使用

関数プロトタイプを使用すると、次の方法で COBOL ユーザー定義関数を呼び出すことができます。

このタスクについて

- FUNCTION キーワードを使用して、COBOL ユーザー定義関数または C 関数などの非 COBOL 外部関数を呼び出すことができます。
- コンパイル グループなどの他のファイルに定義が記述されているユーザー定義関数を呼び出すことができます。
- FUNCTION-ID 段落の ENTRY-INTERFACE 句を使用してプロトタイプで指定されたリンケージ インターフェイス (STATIC、DYNAMIC、DLL など) を使用して、ユーザー定義関数を呼び出すことができます。

関数プロトタイプ定義は、コンパイル時にオブジェクト コードを生成せず、関数プロトタイプについてコンパイラに通知するスタンドアロンのコンパイル単位です。その後、コンパイラが関数の呼び出しに遭遇すると、コンパイラは適合性チェックを実行し、指定されたリンケージ インターフェイスの正しい呼び出しコードを生成することができます。関数のプロトタイプ定義は、関数の呼び出しの前にコンパイル グループ内に記述する必要があります。したがって、すべての関数プロトタイプは、プログラムおよびユーザー定義関数の定義の前に最初に出現することをお勧めします。

関数プロトタイプの例

次の例には、関数「myfunc」の関数プロトタイプ定義、関数「myfunc」を呼び出すプログラム定義、および関数「myfunc」のユーザー定義関数定義が含まれています。FUNCTION-ID 段落の IS PROTOTYPE 句は、このコンパイル単位が関数プロトタイプであることを示すことに注意してください。

このタスクについて

```
Identification division.
  Function-id. myfunc is prototype.
Data division.
  Linkage section.
    1 retval pic 9(9).
  Procedure division returning retval.
  End function myfunc.
*****
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 a pic 9(9).
  Procedure division.
    compute a = function myfunc
    goback.
  End program myprog.
*****
Identification division.
  Function-id. myfunc.
Data division.
  Linkage section.
    1 parm1 pic x(10).
    1 retval pic 9(9).
  Procedure division returning retval.
    compute retval = 0
    goback.
  End function myfunc.
```

外部名を持つ関数プロトタイプ

場合によっては、名前が COBOL ワードとして表現できない関数プロトタイプ、または名前に外部オペレーティング環境での制限や要件がある関数プロトタイプの呼び出し用のコードをコンパイラで生成するこ

とが望ましい場合があります。このような場合には、AS をコード化することができます外部名関数プロトタイプ定義の FUNCTION-ID 段落に句を追加して、外部化された名前を指定します。

このタスクについて

たとえば、関数のプロトタイプに名前を付けることができます"GetRecord",しかし、"GetRecord"実際には、オペレーティング環境での外部名が「QQGETREC」である関数を呼び出します。これを行うには、次のようにコーディングします。

```
Identification division.
  Function-id. GetRecord as 'QQGETREC' is prototype.
Data division.
  Linkage section.
    1 retval pic x(100).
  Procedure division returning retval.
  End function GetRecord.
*****
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 a pic x(100).
  Procedure division.
    move function GetRecord to a
    goback.
  End program myprog.
```

"GetRecord"その後、別のバッチコンパイルで次のユーザー定義関数として定義される可能性があります。

```
Identification division.
  Function-id. GetRecord as 'QQGETREC'.
Data division.
  Linkage section.
    1 retval pic x(100).
  Procedure division returning retval.
    move "data" to retval
    goback.
  End function GetRecord.
```

「QQGETREC」の代わりに、"abc123Get_Customer_Record"。この外部名を使用するには、次の例のように、PGMNAME コンパイラ オプションまたは FUNCTION-ID 段落の ENTRY-NAME 句のいずれかを使用する必要があります。

```
Identification division.
  Function-id. GetRecord as 'abc123Get_Customer_Record' is prototype
    entry-name is longmixed.
Data division.
  Linkage section.
    1 retval pic x(100).
  Procedure division returning retval.
  End function GetRecord.
*****
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 a pic x(100).
  Procedure division.
    move function GetRecord to a
    goback.
  End program myprog.
```

関数プロトタイプのベストプラクティス

関数プロトタイプをコピーブックで定義し、COPY ステートメントを使用してそれらを参照するコンパイルグループに含めることをお勧めします。この方法により、アプリケーション全体のプロトタイプ定義が同じになることが保証されます。さらに、コンパイラが一貫性をチェックできるように、関数プロトタイプ定義を COPY 経由でユーザー定義関数定義と同じコンパイルグループに含めることも推奨されます。

このタスクについて

たとえば、この関数プロトタイプは GETREC というコピーブックに配置されます。

```
Identification division.  
  Function-id. GetRecord is prototype.  
Data division.  
  Linkage section.  
    1 retval pic x(100).  
Procedure division returning retval.  
End function GetRecord.
```

このコピーブックは、ユーザー定義関数の定義とともに別のコンパイルグループに含まれています。

```
Copy GETREC.  
Identification division.  
  Function-id. GetRecord.  
Data division.  
  Linkage section.  
    1 retval pic x(100).  
Procedure division returning retval.  
  move "data" to retval  
  goback.  
End function GetRecord.
```

コピーブックは、以下のコンパイルグループにも含まれています。"GetRecord"が呼び出されます。

```
Copy GETREC.  
Identification division.  
  Program-id. myprog.  
Data division.  
  Working-storage section.  
    1 a pic x(100).  
Procedure division.  
  move function GetRecord to a  
  goback.  
End program myprog.
```

言語標準の考慮

関数プロトタイプは、Enterprise COBOL ISO/IEC 2014 COBOL 標準に基づいています。この機能の一部は Enterprise COBOL ただし、変更されており、Enterprise COBOL。

このタスクについて

変更された部分は次のとおりです。

- オプションの ENTRY-INTERFACE 段落: これは、標準の IDENTIFICATION DIVISION の OPTIONAL 段落の ENTRY-CONVENTION 句に大まかに基づいています。IDENTIFICATION DIVISION 構文の実装は COBOL 標準とは異なるため、規則に従うために、ENTRY-INTERFACE は節ではなく段落になりました。
- オプションの ENTRY-NAME 段落: 理由は上記と同じです。

第 5 部 Web サービスに COBOL を使用

第 35 章 Web サービス・インターフェース

COBOL は、SOAP (Simple Object Access Protocol) や REST (REpresentational State Transfer) に基づくアーキテクチャを使用して Web サービスを提供したり要求したりできます。通常、そのようなサービスへのインターフェースにおけるデータは、JSON (JavaScript Object Notation) または XML (eXtensible Markup Language) で表されます。以下の情報は、JSON や XML のデータを生成したり使用したりするために使用できる COBOL および他の機能について記述したものです。

JSON 入力の処理

JSON ステートメントを使用すると、COBOL プログラム内で JSON テキスト入力を処理できます。このステートメントは、JSON テキストが入っているソース・データ項目と、パーサーによって取り込まれる受信データ項目を識別します。

さらに、z/OS Client Web Enablement Toolkit は、任意のソースから JSON テキストを取り込むための組み込み z/OS JSON パーサー (z/OS 2.2 に付属、または APAR OA46575 用 PTF がインストールされた z/OS 2.1) を提供することにより、アプリケーションがクライアント / サーバー・スペースに参加できるようにします。ただし、このパーサーは EBCDIC コード・ページ 1047 のみをサポートするため、アプリケーションが別のエンコード形式で受け取る JSON テキストは最初に EBCDIC 1047 に変換してからでないとパーサーに入力できません。詳細については、*z/OS MVS プログラミング: 高水準言語向け呼び出し可能サービスの z/OS JSON パーサーの説明*を参照してください。

関連タスク

673 ページの『[第 36 章 JSON 入力の処理](#)』

683 ページの『[第 37 章 JSON 出力の生成](#)』

687 ページの『[第 38 章 XML 入力の処理](#)』

727 ページの『[第 39 章 XML 出力の生成](#)』

第 36 章 JSON 入力処理

JSON ステートメントを使用すると、COBOL プログラム内で JSON テキスト入力を処理できます。このステートメントは、JSON テキストが入っているソース・データ項目と、パーサーによって取り込まれる受信データ項目を識別します。

必要に応じて、以下の句も指定できます。

- 非例外状態と例外状態のためにメッセージが生成されることを示す WITH DETAIL
- ENCODING ソース JSON ドキュメントで想定されるエンコーディングを指定します
- 取り込まれたデータ項目の代替名を指定する NAME OF
- JSON パーサーによる割り当てからデータ項目を除外するための SUPPRESS
- JSON BOOLEAN 名前/値ペアからデータ項目を変換するための CONVERTING
- 例外が発生した場合に制御を受け取るための ON EXCEPTION
- 例外が発生しなかった場合に制御を受け取るための NOT ON EXCEPTION

JSON テキスト入力は次のようにエンコードされていると想定されます。UTF-8 (CCSID 1208) であり、英数字の、国内、または UTF-8 グループ項目、または英数字カテゴリの基本データ項目、国内、または UTF-8。

JSON PARSE ステートメントを指定すると、制御が JSON パーサーに渡され、入力 JSON テキストが読み取られ、同等の COBOL MOVE ステートメントと同じセマンティクスで受信データ項目への取り込みが行われます。

JSON PARSE ステートメントの実行後、以下の特殊レジスターを使用して、情報をパーサーから受け取ることができます。

- ゼロ以外の JSON-CODE は、発生した例外状態の種類を示す
- ゼロ以外の JSON-STATUS は、発生した非例外状態の種類を示す

JSON 文書の構文解析

以下のステートメントが入っている COBOL ソース・プログラムを考えてみます。

```
Identification division.
  Program-id. jparse1.
Data division.
  Working-storage section.
    1 msg.
      4 ver usage comp-1.
      4 uid pic 9999 usage display.
      4 txt pic x(32).
  Linkage section.
    1 json-text pic x(128).
Procedure division using json-text.
  Json parse json-text into msg
  end-json.
  If ver equal to 5 then
    display "Message ID is " uid
    display "Message text is " txt ".
  Goback.
End program jparse1.
```

上の JSON PARSE ステートメントは、データ項目 *json-text* を JSON テキストの UTF-8 エンコード・ソースとして、データ項目 *msg-data* を JSON 値の受信側として識別しています。

データ項目 *json-text* に次の内容が含まれていると仮定します。

```
{"msg":{"ver":5,"uid":1000,"txt":"Hello World!"}}
```

ここで、プログラム実行の出力は次のようになります。

```
Message ID is 1000
Message text is 'Hello World!'
```

有効な COBOL データ名ではない JSON 名をデータ項目に一致させる方法

JSON では、COBOL がデータ名において許可している数より多くの文字および文字のタイプを JSON 名に使用できます。JSON 名と COBOL データ名との一致を容易にするには、JSON PARSE ステートメントで NAME 句を使用してください。次の JSON テキストを考えてみます。

```
{"abc+":100}
```

JSON 名 *abc+* は有効な COBOL データ名ではありませんが、NAME 句を使用して、有効な COBOL データ名に一致させることができます。次の COBOL プログラムは、この JSON テキストを COBOL データ項目に解析する方法を示しています。

```
Identification division.
  Program-id. name1.
Data division.
  Working-storage section.
    1 mydata pic 999.
  Linkage section.
    1 json-text pic x(128).
Procedure division using json-text.
  Json parse json-text into mydata
    name of mydata is "abc+"
  end-json.
  Display "mydata is " mydata.
  Goback.
End program name1.
```

NAME 句の使用に注意してください。このプログラムを実行すると、次のような出力が生成されます。

```
mydata is 100
```

上の例には、検討すべきいくつかの重要な詳細があります。

- NAME 句の *literal-1* に表示される文字は、有効な CODEPAGE コンパイラー・オプションの CCSID を使用してエンコードされていると想定されます。
- *literal-1* に表示される文字は、大/小文字を区別しない方式で一致する COBOL データ名とは異なり、大/小文字を区別する方式で JSON 名と一致します。
- NAME 句が全体として、あいまいな名前の指定を生じさせることはありません。¹

1. あいまいな名前の指定について詳しくは、「Enterprise COBOL 言語解説書」で JSON PARSE ステートメントの NAME 句を参照してください。

データ項目が JSON PARSE ステートメントによって取り込まれないようにする

受信側に従属する特定のデータ項目を、JSON PARSE ステートメントによって取り込みたくない場合があります。特定のデータ項目が取り込まれないようにするためには、JSON PARSE ステートメントの SUPPRESS 句を使用し、それらのデータ項目を無視するよう JSON パーサーに指示してください。次の COBOL プログラムを考えてみます。

```
Identification division.
  Program-id. suppl.
Data division.
  Working-storage section.
    1 msg.
    4 ver usage comp-1.
    4 uid pic 9999 usage display.
    4 txt pic x(32).
  Linkage section.
    1 json-text          pic x(128).
```

```

Procedure division using json-text.
  Move 2 to uid.
  Json parse json-text INTO msg
  SUPPRESS uid
end-json.
If ver equal to 5 then
  display "Message ID is " uid
  display "Message text is '" txt "'".
Goback.
End program supp1.

```

データ項目 *uid* は、プログラム内で値 2 に設定されていることに注意してください。ここでは、JSON PARSE ステートメントによるその割り当てを、SUPPRESS 句を使用して抑止します。データ項目 *json-text* の着信 JSON テキストに次の内容が含まれていると仮定します。

```
{ "msg": { "ver": 5, "uid": 10, "txt": "Hello" } }
```

ここで、プログラムの実行結果は次の出力のようになります。

```

Message ID is 0002
Message text is 'Hello'

```

データ項目 *uid* は、値 10 を取り込む代わりに、値 2 を保持しました。

JSON 配列の処理

JSON 配列は、COBOL データ項目 (そのデータ記述項目に OCCURS 節または OCCURS DEPENDING ON 節が入っている) に解析できます。名前 *msg* の JSON 配列が同様の名前の COBOL データ項目に解析される、次の例を考えてみます。

データ項目 *json-text* に次の内容の JSON テキストが含まれていると仮定します。

```
{ "some-data": { "msg": [ { "ver": 5, "uid": 10, "txt": "Hello" },
  { "ver": 5, "uid": 11, "txt": "World" }, { "ver": 5, "uid": 12, "txt": "!" } ] } }
```

OCCURS 節を持つ固定オカレンス・テーブルを使用して、この JSON テキストを解析する COBOL プログラムは、次のとおりです。

```

Identification division.
  Program-id. occ1.
Data division.
  Working-storage section.
    1 some-data.
      2 msg occurs 3.
        4 ver usage comp-1.
        4 uid pic 9999 usage display.
        4 txt pic x(32).
  Linkage section.
    1 json-text pic x(128).
Procedure division using json-text.
  Json parse json-text into some-data
end-json.
If ver(1) equal to 5 then
  Display "Message ID is " uid(1)
  Display "Message text is '" txt(1) "'".
If ver(2) equal to 5 then
  Display "Message ID is " uid(2)
  Display "Message text is '" txt(2) "'".
If ver(3) equal to 5 then
  Display "Message ID is " uid(3)
  Display "Message text is '" txt(3) "'".
Goback.
End program occ1.

```

このプログラムを実行すると、結果は次の出力になります。

```

Message ID is 0010
Message text is 'Hello'
Message ID is 0011
Message text is 'World'
Message ID is 0012
Message text is '!'

```

同様に、OCCURS DEPENDING ON 節を持つ変数オカレンス・テーブルへの解析は、次のようにして実行できます。

```
Identification division.
  Program-id. odo1.
Data division.
  Working-storage section.
    1 i pic 9.
    1 n pic 9.
    1 t pic x(128).
    1 msg_count pic 9.
    1 some-data.
      2 msg occurs 0 to 5 depending on n.
      4 ver usage comp-1.
      4 uid pic 9999 usage display.
      4 txt pic x(32).
  Linkage section.
    1 json-text pic x(128).
  Procedure division using json-text.
  Main section.
    Move 4 to n.
    Move 0 to ver(1).
    Move 0 to ver(2).
    Move 0 to ver(3).
    Move 0 to ver(4).
    Json parse json-text into some-data
  end-json.
  Perform disp_msg varying i from 1 by 1 until i > n.
  Display "Message count: " msg_count.
  Goback.
  Disp_msg section.
    If ver(i) equal to 5 then
      display "Message ID is " uid(I)
      display "Message text is '" txt(I) "'"
      add 1 to msg_count
    else
      display "Invalid Message Version, ID is " uid(I).
  End program odo1.
```

このプログラムを実行すると、結果は次の出力になります。

```
Message ID is 0010
Message text is 'Hello
Message ID is 0011
Message text is 'World
Message ID is 0012
Message text is '!'
Invalid Message Version, ID is 0001
Message count: 3
```

テーブル・エレメント *msg(4)* の従属データ項目は、*msg* テーブルの 4 番目のテーブル項目が JSON テキストに含まれていないために、JSON PARSE ステートメントによって割り当てられないことに注意してください。また、この例で *n* として定義されている OCCURS DEPENDING ON *object* は、データ項目 *some-data* に従属させてはならず、また JSON PARSE ステートメントがプログラム制御を受け取る前に、値が与えられなければなりません。OCCURS DEPENDING ON *object* の値は、JSON PARSE ステートメントが取り込むことができるテーブル・エレメントの最大数です。この JSON テキストに、OCCURS DEPENDING ON *object* の値より多くのテーブル・エレメントがあれば、それらのテーブル・エレメントは無視され、その条件が JSON-STATUS 特殊レジスターに示されます。OCCURS DEPENDING ON *object* は、JSON PARSE ステートメントによって、その設定も更新も行われません。

JSON 匿名配列の解析

NAME IS OMITTED 句を使用しながら、JSON PARSE ステートメントでテーブルのデータ名を指定することで、JSON 匿名配列を解析できます。

次の例を考えてみましょう。

```
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 ACT.
    2 B1 occurs 2.
```

```

        3 C1.
        4 M1 pic 9.
        4 D1 occurs 2.
        5 N1 pic 9.
    1 json-text pic u dynamic.
Procedure division.
    move spaces to ACT
    move
        ' [{"C1":{"M1":1,"D1":[{"N1":3},{"N1":4]}]},
-      '{"C1":{"M1":2,"D1":[{"N1":5},{"N1":6]}]}]'
        to json-text
    json parse json-text into b1
    name b1 is omitted
end-json
display M1(1)
display M1(2)
display N1(1 1)
display N1(1 2)
display N1(2 1)
display N1(2 2)
goback
.
End program myprog.

```

プログラムを実行すると、次の出力が生成されます。

```

1
2
3
4
5
6

```

JSON PARSE レシーバー b1 すべての出現箇所を参照するために下付き文字が付けられていなかった b1。

次の例を考えてみましょう。b1 2 つのディメンションがあり、以下のように別のテーブル内にネストされています。

```

Identification division.
Program-id. myprog.
Data division.
Working-storage section.
    1 ACT.
        2 TOPTABLE occurs 3.
            3 B1 occurs 2.
                4 C1.
                    5 M1 pic 9.
                    5 D1 occurs 2.
                    6 N1 pic 9.
        1 json-text pic u dynamic.
Procedure division.
    move spaces to ACT
    move
        ' [{"C1":{"M1":1,"D1":[{"N1":3},{"N1":4]}]},
-      '{"C1":{"M1":2,"D1":[{"N1":5},{"N1":6]}]}]'
        to json-text
    json parse json-text into b1(2)
    name b1 is omitted
end-json
display M1(2 1)
display M1(2 2)
display N1(2 1 1)
display N1(2 1 2)
display N1(2 2 1)
display N1(2 2 2)
goback
.
End program myprog.

```

プログラムを実行すると、次の出力が生成されます。

```

1
2
3
4
5
6

```

ご了承ください b1(2)JSON PARSE ステートメントで単一のインデックス添え字が指定されました。下付き文字の値 2 テーブル TOPTABLE の 2 番目の出現にデータを入力することを示します。2 番目のインデックスを省略すると、JSON PARSE ステートメントは、入力 JSON テキストにテーブル全体が含まれていると想定することを示します。b1（単一の発生ではなく b1 これは、2 番目のインデックス サブスクリプトが指定された場合の想定結果です。このパターンは、一般に複数のネストされたテーブルに適用されます。つまり、匿名配列は、NAME IS OMITTED 句と、その次元より 1 つ少ない添え字で受信項目を指定することの組み合わせによって解析できます。

JSON null 値の処理

このセクションでは、JSON PARSE ステートメントを使用して JSON null 値を解析する方法について説明します。

次のいずれかの方法で JSON null 値を解析できます。

- 使用方法 IGNORINGJSON の null 値を無視するフレーズ。
- 使用方法 CONVERTINGJSON の null 値を特定の値に変換するフレーズ。
- 使用方法 INDICATING 衛星データ項目を NULL インジケータとして割り当てるフレーズ。

それぞれの詳細は以下の通りです。

JSON null 値を無視する

JSON PARSE 文で JSON null 値を単に無視したい場合は、IGNORING フレーズ。個々の項目を指定することができます IGNORING どの項目が null になる可能性があるかを事前に知っている場合は、このフレーズを使用します。また、ALL キーワードを使用してすべての null 値を無視することもできます。

JSON の null 値を特定の値に変換する

JSON の null を、null が発生した COBOL データ項目内の特定の値に変換する場合は、CONVERTING 句を使用します。たとえば、次の COBOL ソースを考えてみましょう。

```
`Identification division.`
Program-id. myprog.
Data division.
Working-storage section.
1 json-text pic u dynamic.
1 A.
2 B-data PIC X(10).
Procedure division.
move all 'z' to B-data
move u'{"A":{"B-data":null}}' to json-text
json parse json-text into A
converting B-data from json null using space
display "B-data: '" B-data "'"
goback.
End program myprog.
```

コンパイルして実行すると、JSON PARSE 文は"B-data":null つまり、スペースは実質的に B-data 実際、このプログラムの出力は次のようになります。

```
B-data: ' '
```

すべてのアイテムが使用できるわけではないので注意してください。CONVERTING MOVE 互換性ルールによるフレーズ。

衛星データ項目をヌルインジケータとして割り当てる

使用 INDICATING 衛星データ項目を NULL インジケータとして指定できる場合、または指定したい場合に使用するフレーズ。たとえば、次の COBOL ソースを考えてみましょう。

```
`Identification division.`
Program-id. myprog.
```

```

Data division.
Working-storage section.
1 json-text pic u dynamic.
1 A.
2 B-data pic x(10).
2 B-indicator pic x.
Procedure division.
move all 'z' to A.
move u'{"A":{"B-data":null}}' to json-text
json parse json-text into A
indicating B-data is json null
using 'Y' and 'N' in B-indicator
display "B-data: " B-data ""
display "B-indicator: " B-indicator
goback.
End program myprog.

```

コンパイルして実行すると、JSON PARSE 文は"B-data":null そして人口は B-indicator と 'Y' プログラムの出力を見るとわかるように、

```

B-data: 'zzzzzzzzzz'
B-indicator: Y

```

ご了承ください B-dataJSON PARSE によって入力されませんでした。

JSON PARSE の例

この例は、JSON PARSE ステートメントによる、さまざまなタイプの COBOL データ項目への JSON テキストの処理を示しています。この例の目的として、JSON テキストは直接プログラム・ソースに組み込まれています。プログラムの出力は、この後に示します。

```

Identification division.
Program-id. jp_ex.
Data division.
Working-storage section.
1 jtxt-1047-client-data.
3 pic x(16) value '{"client-data":{"'.
3 pic x(28) value ' "account-num":123456789012,'.
3 pic x(19) value ' "balance":-125.53,'.
3 pic x(17) value ' "billing-info":{"'.
3 pic x(22) value ' "name-first":"John",''.
3 pic x(22) value ' "name-last":"Smith",''.
3 pic x(37) value ' "addr-street":"12345 First Avenue",''.
3 pic x(25) value ' "addr-city":"New York",''.
3 pic x(27) value ' "addr-region":"New York",''.
3 pic x(21) value ' "addr-code":"10203"''.
3 pic x(3) value '}''.
3 pic x(2) value '}''.
3 pic x(1) value '}''.
1 jtxt-1047-transactions.
3 pic x(16) value '{"transactions":['.
3 pic x(14) value ' {"tx-record":['.
3 pic x(3) value ' [''.
3 pic x(4) value ' {''.
3 pic x(19) value ' "tx-uid":107,'.
3 pic x(34) value ' "tx-item-desc":"prod a ver 1",''.
3 pic x(30) value ' "tx-item-uid":"ab142424",''.
3 pic x(26) value ' "tx-priceinUS$":12.34,'.
3 pic x(35) value ' "tx-comment":"express shipping"''.
3 pic x(5) value ' },''.
3 pic x(4) value ' {''.
3 pic x(19) value ' "tx-uid":1904,'.
3 pic x(35) value ' "tx-item-desc":"prod g ver 2",''.
3 pic x(30) value ' "tx-item-uid":"gb051533",''.
3 pic x(27) value ' "tx-priceinUS$":833.22,'.
3 pic x(35) value ' "tx-comment":"digital download"''.
3 pic x(5) value ' },''.
3 pic x(3) value ' ]''.
3 pic x(2) value '}''.
3 pic x(1) value '}''.
1 jtxt-1208 pic x(1000) value is all x'20'.
77 txnum pic 999999 usage display value zero.
1 client-data.
3 account-num pic 999,999,999,999.
3 balance pic $$$9.99CR.
3 billing-info.

```



```

5 name-first pic n(20).
5 name-last pic n(20).
5 addr-street pic n(20).
5 addr-city pic n(20).
5 addr-region pic n(20).
5 addr-code pic n(10).
3 transactions.
5 tx-record occurs 0 to 100 depending txnum.
7 tx-uid pic 99999 usage display.
7 tx-item-desc pic x(50).
7 tx-item-uid pic AA/9999B99.
7 tx-price pic $$$9.99.
7 tx-comment pic n(20).
Procedure division.
Initialize jtxt-1208 all value.
*****
* In this example, input to JSON PARSE starts in EBCDIC codepage *
* 1047 and then be converted to UTF-8 (codepage 1208). *
* Convert to specific codepages using the display-of function. *
* The first argument to display-of should be type 'national', *
* which the COBOL compiler represents in UTF-16. *
*****
Move function display-of(
function national-of(
jtxt-1047-client-data) 1208)
to jtxt-1208(1:function length(jtxt-1047-client-data)).

Json parse jtxt-1208 into client-data
with detail
suppress transactions
not on exception
display "Successful JSON Parse"
end-json.

Display "Account Number:"
Display " " account-num
Display "Balance:"
Display " " balance
Display "Client Information: "
Display " Name:"
Display " " function display-of(name-last)
Display " " function display-of(name-first)
Display " Address:"
Display " " function display-of(addr-street)
Display " " function display-of(addr-city)
Display " " function display-of(addr-region)
Display " " function display-of(addr-code).

Move 2 to txnum.
Initialize jtxt-1208 all value.
Move function display-of(
function national-of(
jtxt-1047-transactions) 1208)
to jtxt-1208(1:function length(jtxt-1047-transactions)).

Json parse jtxt-1208 into transactions
with detail
name tx-price is 'tx-priceinUS$'
not on exception
display "Successful JSON Parse"
end-json.

Display "Transactions:"
Display " Record 1:"
Display " TXID: " tx-uid(1)
Display " Description: " tx-item-desc(1)
Display " Item ID: " tx-item-uid(1)
Display " Price: " tx-price(1)
Display " Comment: "
function display-of(tx-comment(1))
Display " Record 2:"
Display " TXID: " tx-uid(2)
Display " Description: " tx-item-desc(2)
Display " Item ID: " tx-item-uid(2)
Display " Price: " tx-price(2)
Display " Comment: "
function display-of(tx-comment(2))

Goback.
End program jp_ex.

```

プログラムの出力は次のとおりです。

```
Successful JSON Parse
Account Number:
  123,456,789,012
Balance:
  $125.53CR
Client Information:
  Name:
    Smith
    John
  Address:
    12345 First Avenue
    New York
    New York
    10203
Successful JSON Parse
Transactions:
Record 1:
  TXID:      00107
  Description: prod a ver 1
  Item ID:   ab/1424 24
  Price:     $12.34
  Comment:   express shipping
Record 2:
  TXID:      01904
  Description: prod g ver 2
  Item ID:   gb/0515 33
  Price:     $833.22
  Comment:   digital download
```

関連参照

JSON PARSE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

第 37 章 JSON 出力の生成

JSON GENERATE ステートメントを使用すれば COBOL データ項目を JSON テキストとして表すことができます。これにより、ソースと出力のデータ項目が識別されます。

オプションで以下の項目も指定できます。

- 生成された文字数を受け取るデータ項目
- 生成される JSON ドキュメントのエンコーディング
- 入力データ項目の代替名。
- 出力 JSON テキストから除外されるデータ項目。
- JSON BOOLEAN 名前/値ペアに変換されるデータ項目
- 例外発生時に制御を受け取るステートメント

JSON テキストは Web サービスへのインターフェースのリソースを表すために使用でき、次のようにエンコードされます。UTF-8 出力データ項目が英数字の場合または UTF-8、または UTF-16 出力データ項目が国別である場合。

JSON GENERATE ステートメントの使用

次の例を考えてみましょう。

```
01 Greeting.  
  02 Msg pic x(80) value 'Hello, World!'.  
01 Jtext national pic n(80).  
01 i binary pic 99.  
...  
  JSON generate Jtext from Greeting count in i  
  on exception  
  display 'JSON generation error: ' json-code  
  not on exception  
  display function display-of(Jtext(1:i))  
End-JSON
```

上記のコード・シーケンスからは次の出力が生成されます。

```
{"Greeting":{"msg":"Hello, World!"}}
```

次の例は、以下の処理を行うオプションの句を示していて、より複雑です。

- 組み込まれているデータ項目に対して代替 JSON 名を指定する (NAME)
- 機密情報や不要な情報を出力から除外できるようにする (SUPPRESS)

```
01 GRP.  
  05 Ac-No PIC AA9999 value 'SX1234'.  
  05 More.  
  10 Stuff PIC S99V9 OCCURS 2.  
  05 SSN PIC 999/99/9999 value '987-65-4321'.  
01 d pic x(80).  
01 i binary pic 99.  
...  
  move 7.8 to stuff(1), move -9 to stuff(2)  
  JSON generate d from grp count i  
  NAME of stuff is 'Value' SUPPRESS ssn  
  display function display-of(function national-of(  
  d(1:i) 1208))
```

この例では次の出力が生成されます。

```
{"GRP":{"Ac-No":"SX1234","More":{"Value":[7.8,-9.0]}}}
```

JSON 匿名配列の生成

NAME IS OMITTED 句を使用しながら、JSON GENERATE ステートメントでテーブルのデータ名を指定することで、JSON 匿名配列を生成できます。

次の例を考えてみましょう。

```
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 ACT.
      2 B1 occurs 2.
        3 C1.
          4 M1 pic 9.
          4 D1 occurs 2.
            5 N1 pic 9.
      1 json-text pic u dynamic.
Procedure division.
  move 1 to M1(1)
  move 2 to M1(2)
  move 3 to N1(1 1)
  move 4 to N1(1 2)
  move 5 to N1(2 1)
  move 6 to N1(2 2)
  move spaces to json-text
  json generate json-text from b1
    name b1 is omitted
  end-json
  goback
  .
End program myprog.
```

プログラムを実行すると、アイテムに次の JSON テキストが生成されます。json-text:

```
[{"C1":{"M1":1,"D1":[{"N1":3},{"N1":4}]}},{"C1":{"M1":2,"D1":[{"N1":5},{"N1":6}]}}]
```

JSON GENERATE 送信者は b1 すべての出現箇所を参照するために下付き文字が付けられていなかった b1。

次の例を考えてみましょう。b1 2 つのディメンションがあり、以下のように別のテーブル内にネストされています。

```
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 ACT.
      2 TOPTABLE occurs 3.
        3 B1 occurs 2.
          4 C1.
            5 M1 pic 9.
            5 D1 occurs 2.
              6 N1 pic 9.
      1 json-text pic u dynamic.
Procedure division.
  move 1 to M1(1 1)
  move 2 to M1(1 2)
  move 3 to N1(1 1 1)
  move 4 to N1(1 1 2)
  move 5 to N1(1 2 1)
  move 6 to N1(1 2 2)
  move spaces to json-text
  json generate json-text from b1(1)
    name b1 is omitted
  end-json
  goback
  .
End program myprog.
```

プログラムを実行すると、アイテムに次の JSON テキストが生成されます。json-text:

```
[{"C1":{"M1":1,"D1":[{"N1":3},{"N1":4}]}},{"C1":{"M1":2,"D1":[{"N1":5},{"N1":6}]}}]
```

ご了承ください b1(1)JSON GENERATE ステートメントで単一のインデックス添え字が指定されました。下付き文字の値 1 テーブル TOPTABLE の最初の出現が生成されることを示します。2 番目のインデックスを省略すると、JSON GENERATE ステートメントはテーブル全体に対して操作を行うことになります。b1 (単一の発生に基づいて操作するのではなく、b1 これは、2 番目のインデックス サブスクリプトが指定された場合にも発生します。このパターンは、一般に複数のネストされたテーブルに適用されます。つまり、NAME IS OMITTED 句と、送信項目をその次元より 1 つ少ない添え字で指定することを組み合わせ、匿名配列を生成できます。

JSON null 値の生成

このセクションでは、JSON GENERATE ステートメントを使用して JSON null 値を生成する方法について説明します。

JSON null 値を生成するには、次のいずれかの方法を使用できます。

- 使用方法 CONVERTINGCOBOL データ項目の特定の値を JSON null に変換するフレーズ。
- 使用方法 INDICATING 衛星データ項目を NULL インジケータとして割り当てるフレーズ。

それぞれの詳細は以下の通りです。

COBOL データ項目の特定の値を JSON null に変換する

使用 CONVERTINGCOBOL データ項目の特定の値で JSON null を表す場合に使用します。たとえば、次の COBOL ソースを考えてみましょう。

```
CBL CODEPAGE(1047)
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 json-text pic u dynamic.
    1 A.
    2 B-data PIC X(10).
Procedure division.
  move space to B-data
  set length of json-text to 0
  json generate json-text from A
    converting B-data to json null using space
  display function display-of(json-text 1047)
  goback.
End program myprog.
```

コンパイルして実行すると、JSON GENERATE 文は次のように変換します。B-data JSON null の場合 B-data スペースが含まれています。プログラムの出力は次のようになります。

```
{"A":{"B-data":null}}
```

すべての項目を指定できるわけではないので注意してください。CONVERTING MOVE 互換性ルールによるフレーズ。

衛星データ項目をヌルインジケータとして割り当てる

使用 INDICATING 衛星データ項目を NULL インジケータとして指定できる場合、または指定したい場合に使用するフレーズ。たとえば、次の COBOL ソースを考えてみましょう。

```
`CBL CODEPAGE(1047)`
Identification division.
  Program-id. myprog.
Data division.
  Working-storage section.
    1 json-text pic u dynamic.
    1 A.
    2 B-data pic x(10).
    2 B-indicator pic x.
Procedure division.
  move "Y" to B-indicator
  json generate json-text from A
```

```
indicating B-data is json null  
using 'Y' in B-indicator  
display function display-of(json-text 1047)  
goback.  
End program myprog.
```

コンパイルして実行すると、JSON GENERATE 文は、次の場合に JSON null を生成します。B-indicator 含む 'Y'、指定されている通り INDICATING フレーズ。プログラムの出力は次のようになります。

```
{"A":{"B-data":null}}
```

第 38 章 XML 入力処理

COBOL プログラムで XML 入力処理するには、XML PARSE 声明。

このタスクについて

XML PARSE ステートメントは、高速 XML パーサーとの間の COBOL 言語インターフェースです。次のように XMLPARSE コンパイラ・オプションを使用して、アプリケーションに適切なパーサーを選択します。

- XMLPARSE(XMLSS) は、z/OS XML System Services パーサーを選択します。

このオプションによって、名前空間の処理、XML スキーマに対する XML 文書の検証、およびテキスト・フラグメントの国別文字表現 (Unicode UTF-16) への変換などの拡張機能が提供されます。

- XMLPARSE(COMPAT) は、COBOL ライブラリーに組み込まれた XML パーサーを選択します。

このオプションにより、Enterprise COBOL 3 および 4 の XML 構文解析との互換性が実現します。

XML 入力の処理には、XML パーサーと、パーサー・イベントを処理する処理プロシージャの間で制御を渡すことが含まれます。

XML 入力を処理するには、以下の COBOL 機能を使用します。

- の XML PARSEXML 解析を開始し、ソース XML 文書と処理手順を識別するためのステートメント。

XML PARSE ステートメントの次のオプション句を使用することもできます。

- ENCODING は、XML 文書のエンコードを指定します。
- VALIDATING は、XML 文書の検証で照合される XML スキーマを特定します
- 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。
- 特殊レジスターは、パーサーと処理プロシージャの間で次のように情報を交換します。
 - XML-CODEXML 解析のステータスを受け取る場合によってはパーサーに情報を返す
 - 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-INFORMATION には、XML イベントが完了しているかどうかを簡単に判別するための手段があります。
 - 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。
 - XML-NAMESPACE または XML-NNAMESPACE は、NAMESPACE-DECLARATION XML イベント、または名前空間の要素名または属性名の名前空間 ID を受け取ります。
 - XML-NAMESPACE-PREFIX または XML-NNAMESPACE-PREFIX は、NAMESPACE-DECLARATION XML イベント、または接頭部の要素名または属性名の名前空間接頭部を受け取ります。
- XML PARSE ステートメントのオプションの RETURNING NATIONAL 句は、英数字データ項目の XML 文書のフラグメントが、UTF-16 に変換されて、国別特殊レジスター XML-NTEXT、XML-NNAMESPACE、および XML-NNAMESPACE-PREFIX の処理プロシージャに返されることを指示します。

あなたは ENCODING、VALIDATING、そして RETURNING NATIONAL のフレーズ XML PARSE 声明場合にのみ XMLPARSE(XMLSS)有効です。

リンク・エディットの考慮事項: XML PARSE ステートメントを含んだ COBOL プログラムは、AMODE 31 を使用してリンク・エディットする必要があります。

関連概念

[688 ページの『COBOL での XML パーサー』](#)

関連タスク

689 ページの『XML 文書へのアクセス』

690 ページの『XML 文書の構文解析』

709 ページの『XML PARSE の例外処理』

713 ページの『XML 構文解析の終了』

関連参照

462 ページの『XMLPARSE』(コンパイラー・オプション)

704 ページの『XML 文書のエンコード方式』

872 ページの『XML 参照資料』

Extensible Markup Language (XML)

COBOL での XML パーサー

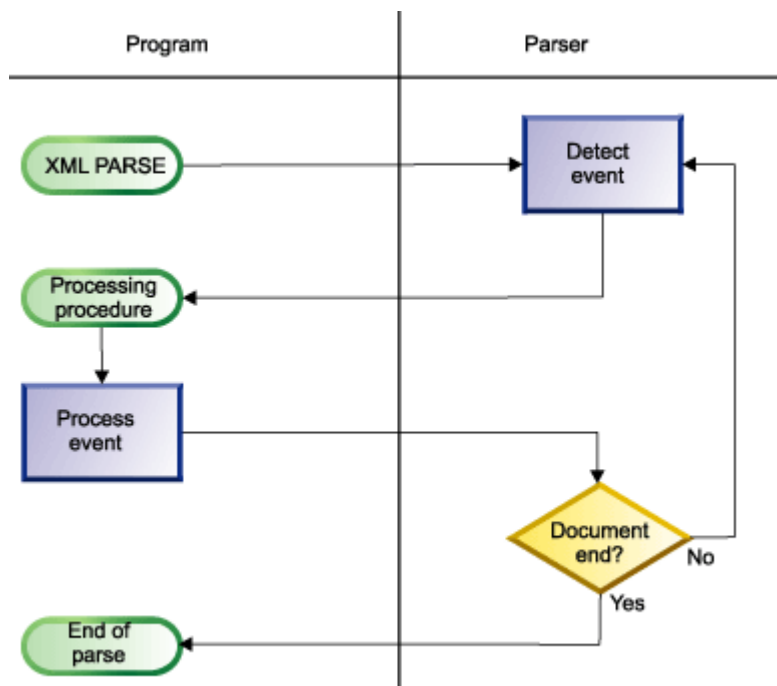
Enterprise COBOL ではイベント・ベースのインターフェースが提供されるため、これを使用して XML 文書を構文解析し、さらに COBOL データ構造に変換することができます。

XML パーサーがソース XML 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。

の実行 XML PARSE ステートメントは解析を開始し、パーサーによる処理手順を確立します。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返します。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。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。

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 COBOL 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続行されます。

XML パーサーは、XML 文書のさまざまな側面が整形形式であ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。

XML スキーマと照合した検証を行って XML 文書を解析する場合、z/OS XML System Services パーサーは、スキーマに規定された内容と構造に XML 文書が従っているかを追加で検証します。例えば、パーサーは、予期しないエレメントや属性がないか、必須のエレメントや属性で欠落しているものがないか、また、エレメントや属性の値が正しいかを確認します。

関連概念

[700 ページの『XML スキーマ』](#)

[705 ページの『XML 入力文書エンコード』](#)

関連タスク

[690 ページの『XML 文書の構文解析』](#)

[698 ページの『検証を伴う XML 文書の構文解析』](#)

[709 ページの『XML PARSE の例外処理』](#)

[713 ページの『XML 構文解析の終了』](#)

関連参照

[704 ページの『XML 文書のエンコード方式』](#)

[XML specification](#)

XML 文書へのアクセス

XML 文書を解析する前に、XML PARSE ステートメントを実行する場合は、ドキュメントをプログラムでできるようにする必要があります。XML 文書を取得する一般的な方法としては、WebSphere MQ メッセージ、CICS 一時キュー、通信域、または IMS メッセージ処理キュ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。

このタスクについて

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。データ項目は、WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION。
- (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。

XMLPARSE (XMLSS) オプションが有効である場合には、ファイルのテキストを 1 レコード (またはセグメント) ずつパーサーに渡すことによって、ファイル内の XML 文書を構文解析することができます。この機能は、非常に大きな XML 文書を構文解析する場合に便利です。

関連タスク

[545 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』](#)

[569 ページの『第 25 章 COBOL プログラムの開発 \(IMS の場合\)』](#)

[701 ページの『XML 文書を 1 セグメントずつ構文解析』](#)

関連参照

[462 ページの『XMLPARSE』](#) (コンパイラー・オプション)

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・フラグメントで示すように、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。

このタスクについて

```
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
```

最初に XML PARSE ステートメントで、XML 文書文字ストリームを含む構文解析データ項目 (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、カテゴリ・国別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、基本英数字データ項目または英数字グループ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 UTF-16、CCSID 1200 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、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。

次に、文書の構文解析で発生した 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。

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合には、XML PARSE ステートメントの次のオプション句を使用することもできます。

- ENCODING は、文書の CCSID を指定します。
- RETURNING NATIONAL は、パーサーに UTF-8 または 1 バイト文字を国別文字に自動変換させて、処理プロシージャに返します。
- VALIDATING は、パーサーに、XML スキーマと照合して文書を検証させます

さらに、以下のオプションの句の一方または両方を指定して (上記フラグメントを参照)、構文解析の終了後に実行さ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す。
- 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。使用 END-XML ネストする XML PARSE を使用するステートメント ON EXCEPTION または NOT ON EXCEPTION 条件文内のフレーズ。

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、制御はパーサーに返されます。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。
- XMLPARSE(XMLSS) が有効である場合、次のいずれかになります。
 - パーサーは文書内のエラーを検出して、(例外の種類に関係なく) EXCEPTION イベントをシグナル通知します。

- パーサーは END-OF-INPUT イベントをシグナル通知します。処理プロシージャは、特殊レジスター XML-CODE がまだゼロに設定された状態でパーサーに戻ります。これは、これ以上 XML データがパーサーに渡されないことを示します。
- XMLPARSE (COMPAT) が有効である場合、次のいずれかになります。
 - パーサーはエンコード競合の EXCEPTION イベントをシグナル通知します。処理プロシージャは、パーサーに戻る前に、特殊レジスター XML-CODE をゼロまたは正しい CCSID に再設定しません。
 - パーサーが文書内にエラーを検出し、EXCEPTION イベント (エンコード競合以外) をシグナル通知します。処理プロシージャは、パーサーに戻る前に、特殊レジスター XML-CODE をゼロに再設定しません。
- パーサーに戻る前に XML-CODE 特殊レジスターを -1 に設定する、処理プロシージャ内のユーザーのコードによって、構文解析プロセスは意図的に終了されます。

関連概念

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

[700 ページの『XML スキーマ』](#)

[696 ページの『XML-INFORMATION』](#)

関連タスク

[691 ページの『XML を処理するためのプロシージャの作成』](#)

[698 ページの『検証を伴う XML 文書の構文解析』](#)

[701 ページの『XML 文書を 1 セグメントずつ構文解析』](#)

[708 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

[462 ページの『XMLPARSE』 \(コンパイラ・オプション\)](#)

[704 ページの『XML 文書のエンコード方式』](#)

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

[874 ページの『XMLPARSE \(COMPAT\) が有効な場合の XML PARSE 例外』](#)

XML PARSE ステートメント (*Enterprise COBOL for z/OS 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。

このタスクについて

パーサーは、イベントを検出するたびに、情報を特殊レジスターに入れて処理プロシージャに渡します。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。

XML-EVENT 特殊レジスターを検査して、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。XML-TEXT または XML-NTEXT の特殊レジスターからイベントに関連付けられたテキストを取得します。

また、XMLPARSE (XMLSS) オプションが有効な場合、特殊レジスター XML-NAMESPACE または XML-NNAMESPACE を使用して、XML イベントに関連付けられた名前空間 ID があればそれを判別し、XML-NAMESPACE-PREFIX または XML-NNAMESPACE-PREFIX の特殊レジスターを検査して、関連接頭部があればそれを判別します。

XML 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。

XML の特殊レジスターに関する追加の詳細については、以下の表を参照してください。

表 76. 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	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 -INFORMATION ¹	PICTURE S9(9) USAGE BINARY VALUE 0	XML EVENT が完了しているかどうかを簡単に判別するための手段
XML -TEXT ¹	可変長基本カテゴリー英数字項目。	XML PARSE ID ³ 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ー国別項目。	XML PARSE ID ³ 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)
XML -NAMESPACE ^{1, 4}	可変長基本カテゴリー英数字項目。	XML 文書が英数字データ項目である場合に、NAMESPACE-DECLARATION XML イベントまたは名前空間に存在するエレメントまたは属性名の名前空間 ID ³
XML -NNAMESPACE ^{1, 4}	可変長基本カテゴリー国別項目。	名前空間識別子 NAMESPACE-DECLARATIONXML イベントまたは名前空間内の要素または属性名の場合、XML 文書が国別データ項目内にあるか、RETURNING NATIONAL フレーズは、XML PARSE 声明
XML -NAMESPACE-PREFIX ^{1, 4}	可変長基本カテゴリー国別項目。	XML 文書が英数字データ項目である場合の、NAMESPACE-DECLARATION XML イベント、またはデフォルトではない名前空間に存在するエレメントまたは属性名の接頭部 (存在する場合) ³
XML -NNAMESPACE-PREFIX ^{1, 4}	可変長基本カテゴリー国別項目。	接頭辞 (ある場合) NAMESPACE-DECLARATION XML イベントまたは非デフォルト名前空間にある要素または属性名の場合、XML 文書が国別データ項目内にあるか、RETURNING NATIONAL フレーズは、XML PARSE 声明
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、XML GENERATE 処理手順のステートメントで、XML -CODE の前に XML GENERATE ステートメントを実行し、保存した値を復元します。XML GENERATE 声明。 3. XML PARSE ステートメントで英数字データ項目について RETURNING NATIONAL 句を指定した場合は、対応する国別特殊レジスターでテキストが返されます。RETURNING NATIONAL 句は、XMLPARSE(XMLSS) オプションが有効である場合にのみ指定できます。 4. パーサーは、XMLPARSE(XMLSS) オプションが有効である場合にのみ、名前空間特殊レジスターを設定します。		

制約事項:

- 処理手順は、XML PARSE 声明。ただし、処理手順がメソッドまたは最外部プログラムに制御を渡す場合、INVOKE または CALL ステートメントでは、ターゲットメソッドまたはプログラムは、同じまたは異なる XML PARSE 声明。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。
- 処理プロシーチャーの範囲で、GOBACK または EXIT PROGRAM ステートメントを実行してはなりません。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、処理プロシーチャーの範囲で実行されます。

処理プロシーチャーに 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。

コンパイラーは、各処理プロシーチャーの最後のステートメントの後に、戻り機構を挿入します。

715 ページの『例: XML の処理用プログラム』

関連概念

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

[696 ページの『XML-TEXT および XML-NTEXT』](#)

[697 ページの『XML-NAMESPACE および XML-NNAMESPACE』](#)

[697 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』](#)

関連タスク

[701 ページの『XML 文書を 1 セグメントずつ構文解析』](#)

[698 ページの『検証を伴う XML 文書の構文解析』](#)

[713 ページの『XML 構文解析の終了』](#)

関連参照

[462 ページの『XMLPARSE』](#) (コンパイラー・オプション)

XML-EVENT (*Enterprise COBOL for z/OS* 言語解説書)

XML イベント

XML イベントは、XML 文書の処理中に XML パーサーがさまざまな条件 (END-OF-INPUT や EXCEPTION など) を検出したり、文書フラグメント (CONTENT-CHARACTERS や START-OF-CDATA-SECTION など) を検出したりした場合に発生します。

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、XML-EVENT 特殊レジスターを処理プロシージャーに渡します。イベントによっては、パーサーは他の特殊レジスターを設定して、イベントに関する追加情報を含めます。

ほとんどの場合、パーサーは次のように XML-TEXT または XML-NTEXT の特殊レジスターに、イベントを引き起こした XML フラグメントを設定します。

- XMLPARSE (COMPAT) コンパイラー・オプションが有効であれば、XML 文書が国別データ項目である場合、またはパーサーが文字参照を検出した場合に、パーサーは XML-NTEXT を設定します。それ以外の場合には、XML-TEXT を設定します。
- もし XMLPARSE (XMLSS) が有効になっている場合、パーサーは XML-NTEXT もし RETURNING NATIONAL フレーズは、XML PARSE ステートメントの場合、または XML 文書が国別データ項目にある場合、パーサーは XML-TEXT。

XMLPARSE (XMLSS) が有効であれば、NAMESPACE-DECLARATION イベントの場合、または名前空間に存在する名前が検出された場合に、パーサーは名前空間特殊レジスターを設定します。

パーサーがエンコーディングの競合や整形式性の欠如を検出した場合または検証文書にエラーがある場合は、XML-EVENT に 'EXCEPTION' 例外に関する追加情報を提供します XML-CODE 特殊レジスタ。

(XMLPARSE (XMLSS) が有効である場合にのみ、検証を伴う構文解析が可能です。詳細については、検証を伴う構文解析に関する関連タスクを参照してください。

XML イベント・セットについて詳しくは、XML-EVENT に関する関連参照を参照してください。

[714 ページの『例: 単純な文書の構文解析』](#)

関連概念

[688 ページの『COBOL での XML パーサー』](#)

[694 ページの『XML-CODE』](#)

[696 ページの『XML-INFORMATION』](#)

[696 ページの『XML-TEXT および XML-NTEXT』](#)

[697 ページの『XML-NAMESPACE および XML-NNAMESPACE』](#)

[697 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』](#)

関連タスク

[691 ページの『XML を処理するためのプロシージャーの作成』](#)

[698 ページの『検証を伴う XML 文書の構文解析』](#)

関連参照

462 ページの『XMLPARSE』 (コンパイラー・オプション)

872 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』

874 ページの『XMLPARSE(COMPAT) が有効な場合の XML PARSE 例外』

XML-EVENT (Enterprise COBOL for z/OS 言語解説書)

XML-CODE

EXCEPTION イベント以外の各 XML イベントについて、パーサーは、XML-CODE 特殊レジスターの値にゼロを設定します。EXCEPTION イベントについては、パーサーは、XML-CODE に特定の例外を識別する値を設定します。

可能な例外コードについては、関連参照を参照してください。

パーサーが処理プロシージャから XML PARSE ステートメントに制御を戻したとき、XML-CODE には通常、パーサーによって設定された最新の値が入っています。ただし、EXCEPTION 以外のイベントでは、処理プロシージャで XML-CODE を -1 に設定すると、制御がパーサーに戻ったときに、ユーザー設定の例外条件で構文解析が終了し、XML-CODE には値 -1 が保持されます。

のために EXCEPTIONXML イベントいつ XMLPARSE (COMPAT) 有効です処理手順によっては、セット XML-CODE 意味のある制御がパーサーに戻る前の値。(詳細については、XML PARSE 例外の処理とエンコード競合の処理に関する関連タスクを参照してください。) XML-CODE を他のゼロ以外の値に設定した場合、または他の例外のために設定した場合には、パーサーは XML-CODE を元の例外コードに再設定します。

START-OF-DOCUMENT XML イベントで、コンパイラー・オプション XMLPARSE (COMPAT) が有効である場合、制御がパーサーに戻る前に、処理プロシージャは XML-CODE を 1 に設定できます。このアクションは、構文解析時に獲得されたすべての言語環境・リソースを (構文解析の終了時に) 解放するようにパーサーに指示します。

以下の表では、XML-CODE をさまざまな値に設定した場合の結果を示しています。左端の列は、処理プロシージャに渡された XML イベントのタイプを表し、他の列の見出しは、処理プロシージャによって設定された XML-CODE の値を表します。それぞれの行と列が交差したところのセルに、XML イベントと XML-CODE 値の特定の組み合わせで処理プロシージャから戻ったときのパーサーのアクションが示されています。

表 77. XMLPARSE (XMLSS) が有効な場合の XML-CODE に対する処理プロシージャの変更の結果				
XML イベント・タイプ	XML-CODE を -1 に設定	XML-CODE を 0 に設定	XML-CODE を 1 に設定	XML-CODE を他のゼロ以外の値に設定
致命的な EXCEPTION	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持
警告 EXCEPTION (理由コード 800 または 801)	設定を無視。元の XML-CODE 値を保持	次のイベントは ATTRIBUTE-NAME または START-OF-ELEMENT	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持
END-OF-INPUT	即時に終了。XML-CODE = -1 ¹	次のイベントは END-OF-DOCUMENT ²	次のイベントは入力に依存 ²	致命的な実行時エラー (メッセージ 230S)
通常イベント	即時に終了。XML-CODE = -1 ¹	XML-CODE は既に 0 であり、変更なし	致命的な実行時エラー (メッセージ 230S)	致命的な実行時エラー (メッセージ 230S)
1. XML 構文解析の終了に関する関連タスクを参照してください。 2. 1 セグメントずつの文書構文解析に関する関連タスクを参照してください。				

表 78. 処理手順の変更の結果 XML-CODE と XMLPARSE (COMPAT) 事実上				
XML イベント・タイプ	-1	0	XML - CODE-100,000	その他のゼロ以外の値
エンコード競合例外 (例外コード 50 から 99)	設定を無視。元の XML-CODE 値を保持	個別の例外コードに応じてエンコードを選択 ¹	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持
エンコード選択例外 (例外コード > 100,000)	設定を無視。元の XML-CODE 値を保持	CODEPAGE 値を使用して構文解析 ²	差 (上記) をエンコード値として使用して構文解析 ²	設定を無視。元の XML-CODE 値を保持
その他の例外	設定を無視。元の XML-CODE 値を保持	例外コード 1 から 49 の場合にのみ限定的に継続 ³	設定を無視。元の XML-CODE 値を保持	設定を無視。元の XML-CODE 値を保持
通常イベント (START-OF-DOCUMENT を除く)	即時に終了。XML-CODE = -1 ⁴	[XML-CODE に明らかな変更はなし]	即時に終了。XML-CODE = -1	即時に終了。XML-CODE = -1
START-OF-DOCUMENT	即時に終了。XML-CODE = -1 ⁴	[XML-CODE に明らかな変更はなし]	即時に終了。XML-CODE = -1	- XML-CODE = 1 - そうでない場合は即時に終了。XML-CODE = -1
1. 例外コードについては関連リファレンスを参照してください。XML PARSE 例外と XMLPARSE (COMPAT) 事実上。 2. エンコード競合の処理に関する関連タスクを参照してください。 3. XML PARSE 例外の処理に関する関連タスクを参照してください。 4. XML 構文解析の終了に関する関連タスクを参照してください。				

XML 生成でも XML-CODE 特殊レジスターが使用されます。詳細については、取り扱いに関する関連タスクを参照してください。XML GENERATE 例外。

関連概念

[711 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

[691 ページの『XML を処理するためのプロシージャの作成』](#)

[701 ページの『XML 文書を 1 セグメントずつ構文解析』](#)

[709 ページの『XML PARSE の例外処理』](#)

[713 ページの『XML 構文解析の終了』](#)

[732 ページの『XML GENERATE 例外の処理』](#)

関連参照

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

[874 ページの『XMLPARSE \(COMPAT\) が有効な場合の XML PARSE 例外』](#)

[880 ページの『XML GENERATE 例外』](#)

XML-CODE (Enterprise COBOL for z/OS 言語解説書)

XML-EVENT (Enterprise COBOL for z/OS 言語解説書)

XML-INFORMATION

大半の XML イベントで、パーサーは XML-INFORMATION を設定し、XML EVENT が完全であるかどうか、または XML コンテンツが複数のイベントにわたっているかどうかを示します。

アプリケーション・プログラム・ロジックでは、構文解析された XML コンテンツの断片を連結するために、XML-INFORMATION 特殊レジスターを使用できます。

関連概念

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

関連タスク

[691 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 (*Enterprise COBOL for z/OS* 言語解説書)

XML-NTEXT (*Enterprise COBOL for z/OS* 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
- XMLPARSE(XMLSS) オプションが有効で、RETURNING NATIONAL 句が XML PARSE ステートメントで指定されている
- の ATTRIBUTE-NATIONAL-CHARACTER または CONTENT-NATIONAL-CHARACTER イベントが発生します。

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。

の数を決定するには文字エンコード単位で XML-NTEXT、使用 LENGTH 固有関数; 例えば FUNCTION LENGTH(XML-NTEXT)。バイト数を決定するには XML-NTEXT、特殊レジスタを使用する LENGTH OF XML-NTEXT。文字エンコード・ユニット数は、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)。

XML-TEXT および XML-NTEXT 特殊レジスターは、処理プロシーチャー外では未定義です。

関連概念

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

関連タスク

[691 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

[462 ページの『XMLPARSE』](#) (コンパイラー・オプション)

XML-TEXT (*Enterprise COBOL for z/OS* 言語解説書)

XML-NTEXT (*Enterprise COBOL for z/OS* 言語解説書)

XML-NAMESPACE および XML-NNAMESPACE

XMLPARSE (XMLSS) オプションが有効であれば、NAMESPACE-DECLARATION XML イベントの場合、または名前空間に存在するエレメント名または属性名が検出された場合に、XML パーサーは、XML-NAMESPACE または XML-NNAMESPACE の特殊レジスターに名前空間 ID を設定します。

XML 文書が国別データ項目である場合、または XML PARSE ステートメントに RETURNING NATIONAL 句が指定された場合、パーサーは XML-NNAMESPACE を設定します。それ以外の場合、パーサーは XML-NAMESPACE を設定します。

特殊レジスター XML-NAMESPACE と XML-NNAMESPACE は、相互に排他的です。パーサーが XML-NAMESPACE を設定した場合、XML-NNAMESPACE は空で長さゼロになります。パーサーが XML-NNAMESPACE を設定した場合、XML-NAMESPACE は空で長さゼロになります。

の数を決定するには文字エンコード単位で XML-NNAMESPACE、使用 LENGTH 固有関数。例:FUNCTION LENGTH(XML-NNAMESPACE)。XML-NNAMESPACE 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NNAMESPACE を使用します。文字エンコード・ユニット数は、バイト数とは異なります。

XML-NAMESPACE 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NAMESPACE 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)。

XML 名前空間特殊レジスターは、処理プロシージャー外では未定義です。

関連概念

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

[697 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』](#)

[696 ページの『XML-TEXT および XML-NTEXT』](#)

関連タスク

[691 ページの『XML を処理するためのプロシージャーの作成』](#)

関連参照

[462 ページの『XMLPARSE』](#) (コンパイラー・オプション)

XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX

XMLPARSE (XMLSS) オプションが有効であれば、名前空間の接頭部も定義する NAMESPACE-DECLARATION XML イベントの場合、または名前空間内のエレメント名または属性名に接頭部を付ける場合に、XML パーサーは、XML-NAMESPACE-PREFIX 特殊レジスターまたは XML-NNAMESPACE-PREFIX 特殊レジスターを設定します。

XML 文書が国別データ項目である場合、または XML PARSE ステートメントに RETURNING NATIONAL 句が指定された場合、パーサーは XML-NNAMESPACE-PREFIX を設定します。それ以外の場合、パーサーは XML-NAMESPACE-PREFIX を設定します。

特殊レジスター XML-NAMESPACE-PREFIX と XML-NNAMESPACE-PREFIX は、相互に排他的です。パーサーが XML-NAMESPACE-PREFIX を設定した場合、XML-NNAMESPACE-PREFIX は空で長さゼロになります。パーサーが XML-NNAMESPACE-PREFIX を設定した場合、XML-NAMESPACE-PREFIX は空で長さゼロになります。

の数を決定するには文字エンコード単位で XML-NNAMESPACE-PREFIX、使用 LENGTH 固有関数。

例:FUNCTION LENGTH(XML-NNAMESPACE-PREFIX)。XML-NNAMESPACE-PREFIX 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NNAMESPACE-PREFIX を使用します。文字エンコード・ユニット数は、バイト数とは異なります。

XML-NAMESPACE-PREFIX 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NAMESPACE-PREFIX 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)。

XML 名前空間接頭部特殊レジスターは、処理プロシージャー外では未定義です。

関連概念

[693 ページの『XML イベント』](#)

[697 ページの『XML-NAMESPACE および XML-NNAMESPACE』](#)

関連タスク

[691 ページの『XML を処理するためのプロシージャの作成』](#)

関連参照

[462 ページの『XMLPARSE』 \(コンパイラー・オプション\)](#)

XML テキストの COBOL データ項目への変換

このタスクについて

XML データは固定長ではなく、固定形式でもないのので、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、項目の定義に JUSTIFIED RIGHT 節を指定します。

数値の XML 値、特に「\$1,234.00」または「\$1234」などの"装飾された"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 で同じものを意味することがありますが、COBOL 送信フィールドとして使用された場合には、まったく別の定義を必要とします。

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

[123 ページの『数値への変換 \(NUMVAL、NUMVAL-C、NUMVAL-F\)』](#)

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

[691 ページの『XML を処理するためのプロシージャの作成』](#)

検証を伴う XML 文書の構文解析

XML 文書の検証では、文書の構造と内容が一連の規則に従っているかどうかを判断します。Enterprise COBOL では、この規則は、実質的に文書のクラスの青写真である XML スキーマに表されています。

このタスクについて

構文解析中に XML 文書を検証するには、XML PARSE ステートメントの VALIDATING 句を使用します。このためには、XMLPARSE (XMLSS) 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する必要があります。

XML 文書は、XML スキーマに対してのみ、検証できます。

Enterprise COBOL で、XML 検証に使用されるスキーマは、*Optimized Schema Representation*、または *OSR* と呼ばれる前処理済みのフォーマットのものである必要があります。テキスト形式のスキーマから OSR 形式のスキーマを生成するには、z/OS ユニックス指示 xsdosrg は、OSR ジェネレーターを呼び出します。

z/OS システム サービス。(あるいは、OSR 生成プログラムをプログラムで呼び出します。詳細については、z/OS XML System Services に関する関連参照を参照してください。)

例えば、item.xsd ファイル内のテキスト形式のスキーマを item.osr ファイル内の前処理済みフォーマットのスキーマに変換するには、次の z/OS ユニックス コマンドを使用できます。

```
xsdosrg -v -o /u/HLQ/xml/item.osr /u/HLQ/xml/item.xsd
```

前処理済みスキーマの場所に応じて、次の 2 つの形式の VALIDATING 句から、いずれかを使用します。

- 1 つの形式として、FILE キーワードを使用し、XML スキーマ名を指定します。この場合、スキーマが、MVS データ・セットまたは z/OS ユニックス ファイル内にある必要があります。
- もう 1 つの形式として、スキーマを含むデータ項目の ID を指定します。

FILE キーワードを使用して XML スキーマ名を指定した場合、COBOL ランタイム・ライブラリーが、XML PARSE ステートメントの実行中にスキーマを自動的に取得します。以下は、この方法で検証を指定したコード断片です。

```
XML PARSE document-item
  VALIDATING WITH FILE schema-name
  PROCESSING PROCEDURE xml-event-handler
ON EXCEPTION
  DISPLAY 'Document has an error.'
  GOBACK
NOT ON EXCEPTION
  DISPLAY 'Document is valid.'
END-XML
```

XML スキーマが入っている外部ファイルに XML スキーマ名を関連付けるには、SPECIAL-NAMES 段落に XML-SCHEMA 節をコーディングし、ファイルを特定するリテラルまたはユーザー定義語を指定します。

例えば、上記のコード断片に示された XML スキーマ名 schema-name を DD 名 DDSHEMA に関連付けるには、次のように XML-SCHEMA 節にリテラルとして DD 名をコーディングします。

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
  XML-SCHEMA schema-name IS 'DDSCHEMA'.
```

プログラムを実行する際に、DD ステートメントを次のようにコーディングして、DD 名 DDSHEMA を z/OS ユニックス ファイル item.osr に関連付けます。

```
//GO.DDSCHEMA DD PATH='/u/HLQ/xml/item.osr'
```

または、類似の TSO ALLOCATE コマンドを使用します。

あるいは、上記の例の DDSHEMA は、MVS データ・セットを指定した DSN オプション、または z/OS ユニックス ファイルを指定した PATH オプションによって外部ファイルを特定する環境変数の名前にすることもできます。

スキーマが MVS データ・セット内にある場合、このデータ・セットは、任意の順次データ・セット (QSAM 固定ブロックまたは可変ブロック、あるいは VSAM ESDS など) にすることができます。

XML スキーマが入っている外部ファイルに XML スキーマ名を関連付ける方法の詳細については、XML-SCHEMA 節に関する関連参照を参照してください。

制約事項: CICS では、FILE キーワードを使用した XML 検証はサポートされていません。

FILE キーワードを使用した場合に行われる自動検索は便利です。ただし、同じタイプの複数の XML 文書を検証する場合には、スキーマを一度メモリーに読み込んで、各文書にそのスキーマを再使用するほうが、

自動検索を行うよりもパフォーマンスが向上します。この場合には、もう一方の形式の VALIDATING 句を使用して、XML スキーマが入っている英数字データ項目を参照する ID を指定します。例:

```
XML PARSE document-item
  VALIDATING WITH xmlschema
  PROCESSING PROCEDURE xml-event-handler
ON EXCEPTION
  DISPLAY 'Document has an error.'
  GOBACK
NOT ON EXCEPTION
  DISPLAY 'Document is valid.'
END-XML
```

通常の COBOL ステートメントを使用するなどして、前処理済みのスキーマをデータ項目に読み込みます。この形式の VALIDATING 句について詳しくは、XML PARSE ステートメントに関する関連参照を参照してください。

検証を伴う構文解析では、検証エラーまたは整形式性エラーで例外が発生するまで、通常の XML イベントが返されます。XML 文書が妥当でない場合、パーサーは XML 例外をシグナル通知し、特殊レジスター XML-EVENT に 'EXCEPTION'、特殊レジスター XML-CODE の上位ハーフワードに戻りコード 24、下位ハーフワードに詳細な検証理由コードが設定された状態で、処理プロシーチャーに制御を渡します。

検証を伴う XML 文書の構文解析で発生する例外の戻りコードと理由コードについては、XMLPARSE (XMLSS) が有効な場合の例外に関する関連参照を参照してください。

[724 ページの『例: 検証を伴う XML 文書の構文解析』](#)

関連概念

[694 ページの『XML-CODE』](#)

[700 ページの『XML スキーマ』](#)

関連タスク

[709 ページの『XML PARSE の例外処理』](#)

関連参照

[462 ページの『XMLPARSE』 \(コンパイラー・オプション\)](#)

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

XML PARSE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

XML-SCHEMA 節 (*Enterprise COBOL for z/OS 言語解説書*)

z/OS XML System Services User's Guide and Reference

XML スキーマ

XML スキーマとは、W3C によって定義されたメカニズムで、XML 文書の構造と内容を記述し、制約します。XML スキーマは、それ自体が XML で表され、特定タイプ (購入注文など) の XML 文書のクラスを効率的に定義する。

Enterprise COBOL の場合、XML 文書の検証に使用される XML スキーマは、*Optimized Schema Representation (OSR)* と呼ばれる前処理済みのフォーマットのものである必要があります。このフォーマットについては、z/OS XML System Services に関する関連参照を参照してください。

以下のように在庫管理用の項目を記述する XML 文書があるとします。

```
<stockItem itemNumber="453-SR">
  <itemName>Stainless steel rope thimbles</itemName>
  <quantityOnHand>23</quantityOnHand>
</stockItem>
```

上記のサンプルは、整形式で、以下のスキーマに従った妥当な文書です。(各行の先頭の番号は、スキーマの一部ではなく、スキーマに続く説明で使用されます。)

1. <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- 2.

```

3. <xsd:element name="stockItem" type="stockItemType"/>
4.
5. <xsd:complexType name="stockItemType">
6.   <xsd:sequence>
7.     <xsd:element name="itemName" type="xsd:string" minOccurs="0"/>
8.     <xsd:element name="quantityOnHand">
9.       <xsd:simpleType>
10.        <xsd:restriction base="xsd:nonNegativeInteger">
11.          <xsd:maxExclusive value="100"/>
12.        </xsd:restriction>
13.      </xsd:simpleType>
14.    </xsd:element>
15.  </xsd:sequence>
16.  <xsd:attribute name="itemNumber" type="SKU" use="required"/>
17. </xsd:complexType>
18.
19. <xsd:simpleType name="SKU">
20.   <xsd:restriction base="xsd:string">
21.     <xsd:pattern value="\d{3}-[A-Z]{2}" />
22.   </xsd:restriction>
23. </xsd:simpleType>
24.
25. </xsd:schema>

```

このスキーマでは、ルート・エレメントが **stockItem** であることを宣言しています (行 3)。これには、SKU タイプの必須 **itemNumber** 属性があり (行 16)、一連の他のエレメントが含まれています (行 6 から 15)。

- オプションの **itemName** エレメント。string タイプ (行 7)。
- 必須の **quantityOnHand** エレメント。nonNegativeInteger タイプで、1 から 99 の範囲に制約されます (行 8 から 14)。

タイプ宣言はインラインかつ無名にすることができます (行 9 から 13)。これには、quantityOnHand エレメントに正しい値を指定するための maxExclusive ファセットが含まれています。

一方、itemNumber 属性については、名前付きタイプ SKU が別途宣言されています (行 19 から 23)。これには、正規表現構文を使用して、そのタイプの正しい値が 3 桁の数字、ハイフン (負符号)、2 つの大文字がこの順序で構成されるように指定する、パターン・ファセットが含まれています。

以下に参照される例に、このスキーマと照合して文書を構文解析するプログラムが示されています。

[724 ページの『例: 検証を伴う XML 文書の構文解析』](#)

関連タスク

[698 ページの『検証を伴う XML 文書の構文解析』](#)

関連参照

z/OS XML System Services User's Guide and Reference

XML 文書を 1 セグメントずつ構文解析

パーサーに XML テキストを 1 セグメント (またはレコード) ずつ渡すことによって、XML 文書を構文解析することができます。この手法の主な適用として、非常に大きな文書の処理や、データ・セット内の XML 文書の処理の 2 つがあります。

このタスクについて

この機能を使用するには、XMLPARSE (XMLSS) コンパイラー・オプションを有効にしてプログラムをコンパイルします。

1 セグメントずつ XML 文書を構文解析するには、構文解析データ項目を XML 文書の最初のセグメントに初期化してから、XML PARSE ステートメントを実行します。パーサーは通常の場合と同様に、XML テキストを処理してから、XML イベントを処理プロシーチャーに返します。

テキスト・セグメントの最後で、パーサーは、XML-CODE をゼロに設定して、END-OF-INPUT XML イベントをシグナル通知します。処理対象の文書のセグメントがまだ存在する場合には、処理プロシーチャーで XML データの次のセグメントを構文解析データ項目に移動し、XML-CODE を 1 に設定してから、パーサー

に戻ります。XML セグメントの最後をパーサーにシグナル通知するには、XML-CODE をゼロに設定したままにしてパーサーに戻ります。

構文解析データ項目の長さはセグメントごとに評価され、これによってセグメント長が決定されます。

可変長セグメント: XML 文書セグメントが可変長である場合、構文解析データ項目には可変長項目を指定します。例えば、可変長の XML セグメントの場合、構文解析データ項目を次のいずれかの項目として定義できます。

- 可変長のグループ項目で、OCCURS DEPENDING ON 句
- 参照変更された項目
- アン FD 指定する記録 RECORD IS VARYING DEPENDING ON 句では、依存データ項目は参照修飾子または ODO オブジェクトの長さとして使用されます。FD 記録

XML 文書をパーサーに複数のセグメントで送信した場合、文書コンテンツは、単一イベントで 1 つの大きなフラグメントとしてではなく、複数のイベントで複数のフラグメントとして、処理プロシージャに返されることがあります。

例えば、コンテンツ文字のストリングの真ん中を分割点として 2 つのセグメントに文書を分割した場合、パーサーは 2 つの別個の CONTENT-CHARACTERS イベントでコンテンツを返します。処理プロシージャは、必要に応じてアプリケーションでコンテンツのストリングを再組み立てする必要があります。

開始エレメント・タグ、属性名、名前空間宣言、および終了エレメント・タグは、それらの項目が文書の 2 セグメントに分割された場合でも常に、単一イベントで処理プロシージャに送信されます。

セグメントの分割がマルチバイト文字のバイトを横断して行われた場合、パーサーは、単一イベントで送信するために、分割を検出して文字を再組み立てします。

処理する反復エレメント数が不明な XML 文書を構文解析する場合は、無制限テーブルを使用します。無制限テーブルの詳細については、[95 ページの『無制限テーブルおよび無制限グループの処理』](#)を参照してください。

指定の文書内にあるこれらの各エレメントに対しては、以下のいずれかの方法でテーブル・サイズを管理します。

- エレメント数の計算:
 1. 初期構文解析時に文書内のエレメント数をカウントします。
 2. テーブル用 OCCURS DEPENDING ON オブジェクトをそのサイズに設定します。
 3. テーブル用のストレージを割り振ります。
 4. 2 回目の構文解析を行い、XML を処理します。
- 増分拡張:
 1. 無制限テーブル用 OCCURS DEPENDING ON オブジェクトの初期サイズを設定します。
 2. 文書を通常の方法で構文解析します。各エレメントに対して以下を行います。
 - a. 限度を確認し、必要に応じて無制限テーブルを拡張します。
 3. より大きいストレージ域を新たに割り振ります。
 4. より小さい領域からデータをコピーします。
 5. より小さい領域を解放します。
 6. テーブル・ポインターをより大きいストレージ域のアドレスに設定します。

QSAM ファイルと VSAM ファイル: QSAM ファイルまたは VSAM ファイルに保管された XML 文書は、次のように処理することができます。

1. ファイルを開いて、XML 文書の最初のレコードを読み取る。
2. FD レコードを構文解析データ項目として、XML PARSE ステートメントを実行する。
3. END-OF-INPUT イベントを処理するための処理プロシージャ・ロジックで、XML 文書の次のレコードを読み取って、構文解析データ項目に格納する。ファイルの終わり (ファイル状況コード 10) ではな

い場合、XML-CODE を 1 に設定して、パーサーに戻る。ファイルの終わりである場合、XML-CODE をゼロに設定したままでパーサーに戻る。

4. END-OF-DOCUMENT イベントの処理プロシーチャー・ロジックで、ファイルを閉じる。

ルート・エレメントの後の各種情報:

XML 文書のルート・エレメントの後には任意の順序の 0 個以上のコメントまたは処理命令が続く可能性があります。1 セグメントずつ文書を構文解析した場合、パーサーは、セグメントの最後の項目が不完全である場合にのみ、ルート・エレメントの終了タグの処理後に、END-OF-INPUT XML イベントをシグナル通知します。セグメントが完全な XML 項目 (ルート・エレメント終了タグ、またはその後ろに完全なコメントまたは処理命令がある場合など) で終了している場合、項目自体のイベントの後に発生する、次の XML イベントは、END-OF-DOCUMENT XML イベントになります。

ヒント: ルート・エレメントの最後に続けて XML データのセグメントを提供するには、各セグメントの最後に、少なくとも最初がスペースではない文字の XML 項目を組み込んでください。パーサーが処理する最後のセグメントにのみ、完全な項目を組み込んでください。

例えば、次の例では、各行が XML 文書のセグメントを表しており、テキストを含むセグメントは This comment ends this segment 解析される最後のセグメントです。

```
<Tagline>
COBOL is the language of the future!
</Tagline> <
!--First comment--
> <?pi data=?> <!--
-This comment ends this segment-->
<!-- This segment is not included in the parse-->
```

[722 ページの『例: XML 文書を 1 セグメントずつ構文解析』](#)

関連概念

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

関連タスク

[701 ページの『XML 文書を 1 セグメントずつ構文解析』](#)

[694 ページの『XML-CODE』](#)

関連参照

[462 ページの『XMLPARSE』](#) (コンパイラー・オプション)

XML-INFORMATION 特殊レジスターを使用した分割の処理

XML-INFORMATION 特殊レジスターを使用して、大きい XML 文書を構文解析することができます。

このタスクについて

この機能を使用するには、XMLPARSE (XMLSS) コンパイラー・オプションを有効にしてプログラムをコンパイルします。

文字内容の分割は、入力がセグメント化されていない場合でも、XML データ・ストリーム内の任意の点で起きることがあります。XML-INFORMATION 特殊レジスターにより、内容の再アセンブリーが単純化されます。このレジスターは、あらゆる属性値およびエレメント文字内容で必要とされる可能性があります。

構文解析データ項目の長さはセグメントごとに評価され、これによってセグメント長が決定されます。

例 [715 ページの『例: XML の処理用プログラム』](#) には、後で処理するために、XML 文書から取得した値をプログラム・データ項目に割り当てるためのさまざまな方法が示されています。

XML データは、40 バイト・レコードでパーサーに渡されます。これは、データ・ファイルなどの外部ソースから XML 文書を取得する方法に倣っています。レコード境界は、1 つを除くすべてのデータ分割がパーサーに収容されるように設計されています。例えば、サンプルでは "filling" エレメントの内容を除き、どの内容でも分割をエラーとして扱っています。

例では、"filling" エレメントの内容の再アセンブリーを単純化するためのみに、XML-INFORMATION 特殊レジスターを使用しています。このレジスターは、あらゆる属性値およびエレメント文字内容に使用できます。XML-INFORMATION 値 2 は、ATTRIBUTE-CHARACTERS または CONTENT-CHARACTERS の XML イベントの文字データが後続の XML イベント内に続いているため、完全な文字ストリングを累積するには、文字データをバッファに入れる必要があることを示しています。XML-INFORMATION 値 1 を持つ同タイプの後続の XML イベントは、XML-TEXT または XML-NTEXT に文字内容の最後の部分が含まれていることと、完全なストリングを適切なデータ項目に移動できることを示しています。

この例では、STRING ... WITH POINTER ステートメントは、割り当てる完全な文字値を適切に蓄積して記述します。"filling" 識別子。

```
String xml-text delimited by size into
  content-buffer with pointer tally
On overflow
  Display 'content buffer ('
    length of content-buffer
    ' bytes) is too small'
  Move -1 to xml-code
End-string
```

関連概念

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

関連参照

[462 ページの『XMLPARSE』](#) (コンパイラ・オプション)

[715 ページの『例: XML の処理用プログラム』](#)

XML 文書のエンコード方式

XML 文書は、サポートされたコード・ページでエンコードする必要があります。

国別データ項目で生成または構文解析される XML 文書は、ビッグ・エンディアン形式の Unicode UTF-16、CCSID 1200 でエンコードする必要があります。

XML GENERATE ステートメントの場合、英数字データ項目で生成される文書は、下表にリストされた、Unicode UTF-8 (CCSID 1208)、または 1 バイト EBCDIC エンコードのいずれかでエンコードする必要があります。この表の CCSID は、XML GENERATE ステートメントの ENCODING 句で使用できます。

XML PARSE ステートメントの場合、英数字データ項目の文書は次のようにエンコードする必要があります。

- XMLPARSE(XMLSS) が有効である場合:
 - RETURNING NATIONAL 句が XML PARSE ステートメントで指定されている場合、z/OS Unicode サービスで UTF-16 への変換がサポートされている任意の EBCDIC または ASCII エンコード
 - RETURNING NATIONAL 句が XML PARSE ステートメントで指定されていない場合、下表にリストされている、UTF-8 (CCSID 1208)、またはいずれかの 1 バイト EBCDIC エンコード
- XMLPARSE(COMPAT) が有効である場合: 下表にリストされているいずれかの 1 バイト EBCDIC エンコード

もし XMLPARSE(XMLSS) が有効になっている場合は、サポートされている任意の CCSID (上記のように) を使用できます。XML PARSE) の中に ENCODING のフレーズ XML PARSE 声明。

表 79. XML 文書のコード化文字セット	
CCSID	説明
1208	UTF-8 ¹
1047	Latin 1 / オープン・システム

表 79. XML 文書のコード化文字セット (続き)	
CCSID	説明
1140, 37	米国、カナダ、ユーロ国別拡張コード・ページ (ECECP)、国別拡張コード・ページ (CECP)
1141, 273	オーストリア、ドイツ ECECP、CECP
1142, 277	デンマーク、ノルウェー ECECP、CECP
1143, 278	フィンランド、スウェーデン ECECP、CECP
1144, 280	イタリア ECECP、CECP
1145, 284	スペイン、ラテンアメリカ (スペイン語圏) ECECP、CECP
1146, 285	英国 ECECP、CECP
1147, 297	フランス ECECP、CECP
1148, 500	国際 ECECP、CECP
1149, 871	アイスランド ECECP、CECP
1. サポートされている XML PARSE 声明の中に ENCODING フレーズもし XMLPARSE(XMLSS)有効です	

関連概念

705 ページの『XML 入力文書エンコード』

関連タスク

707 ページの『エンコードの指定』

708 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

727 ページの『第 39 章 XML 出力の生成』

関連参照

385 ページの『CODEPAGE』

462 ページの『XMLPARSE』(コンパイラオプション)

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るエンコードは、次の点に依存します。

- XML 文書が入っているデータ項目のカテゴリ
- XMLPARSE コンパイラ・オプションの設定
- XML PARSE ステートメントに指定されたオプションの句。

国別データ項目に入っている XML 文書の場合、サポートされているエンコードは、ビッグ・エンディアン形式の Unicode UTF-16、CCSID 1200 です。

英数字データ項目に入っている XML 文書の場合、XMLPARSE(XMLSS) コンパイラ・オプションが有効であるときにサポートされているエンコードは次のとおりです。

- もし、RETURNING NATIONAL フレーズは、XML PARSE 声明：UTF-8 または、サポートされている EBCDIC または ASCII エンコード。z/OS Unicode サービスへの変換 UTF-16
- RETURNING NATIONAL 句が指定されていない場合: UTF-8、または XML 文書のエンコードに関する関連参照でリストされている任意の 1 バイト EBCDIC CCSID。

英数字データ項目に入っている XML 文書の場合、XMLPARSE(COMPAT) が有効であるときにサポートされている CCSID は、XML 文書のエンコードに関する関連参照で指定されているものです。

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。特殊レジスター XML-NTEXT で処理プロシージャー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。

XML 宣言および空白文字 (white space)

XML 文書は、XML 宣言がない場合のみ、冒頭に空白文字 (white space) を置くことができます。

- XML 文書の冒頭が XML 宣言である場合、文書中の最初の不等号括弧 (<) がその文書の先頭文字である必要があります。
- XML 文書の冒頭が XML 宣言でない場合、文書中の最初の不等号括弧の前に置くことができるのは空白文字 (white space) のみです。

空白文字 (white-space characters) には、次の表に記載された 16 進値が含まれます。

表 80. 空白文字 (white-space characters) の 16 進値		
空白文字 (white-space character)	EBCDIC	Unicode
スペース	X'40'	X'0020'
水平タブ	X'05'	X'0009'
復帰	X'0D'	X'000D'
改行	X'25'	X'000A'
改行/次の行	X'15'	X'0085'

入力 XML 文書のエンコードの判別

パーサーは、XML 文書を正しく処理するために、そのエンコードを認識している必要があります。

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。

XML 文書のエンコードの判別では、次のような複数のソースが使用されます。

- XMLPARSE (XMLSS) オプションが有効である場合:
 - XML 文書が入っているデータ項目のデータ・タイプ
 - の ENCODING フレーズ (使用されている場合) XML PARSE 声明
 - CODEPAGE コンパイラー・オプションで指定された CCSID
- XMLPARSE (COMPAT) オプションが有効である場合:
 - XML 文書が入っているデータ項目のデータ・タイプ
 - パーサーが文書の最初の数バイトを検査して判別した実際のエンコード
 - XML 文書内で指定されているエンコード宣言
 - CODEPAGE コンパイラー・オプションで指定された CCSID

XMLPARSE (XMLSS) が有効である場合:

- XML 文書内で指定されているエンコード宣言は無視されます。
- 国別データ項目に入っている XML 文書の場合、XML PARSE ステートメントの ENCODING 句は、省略するか CCSID 1200 を指定する必要があります。CODEPAGE コンパイラー・オプションで指定された CCSID は無視されます。実際の文書エンコードがビッグ・エンディアン形式の UTF-16 ではない場合、パーサーは、XML 例外イベントをシグナル通知します。

- 英数字データ項目に入っている XML 文書の場合、ENCODING 句で指定された CCSID は、CODEPAGE コンパイラー・オプションをオーバーライドします。実際の文書エンコードが、指定された CCSID と矛盾する場合、パーサーは構文解析操作の開始時に XML 例外イベントを起動します。

関連タスク

- 142 ページの『国別 (Unicode) 表現との間の変換』
- 707 ページの『エンコードの指定』
- 708 ページの『UTF-8 でエンコードされた XML 文書の構文解析』
- 709 ページの『XML PARSE の例外処理』

関連参照

- 462 ページの『XMLPARSE』 (コンパイラー・オプション)
- 704 ページの『XML 文書のエンコード方式』
- 708 ページの『XML マークアップ内の EBCDIC コード・ページ依存文字』

エンコードの指定

英数字データ項目の XML 文書を構文解析するためのエンコードを指定する方法を選択できます。

このタスクについて

推奨する方法としては、文書からエンコード宣言を省略し、以下のいずれかの方法でエンコードを指定します。

- XMLPARSE (XMLSS) が有効である場合: XML PARSE ステートメントの ENCODING 句、または CODEPAGE コンパイラー・オプション
- XMLPARSE (COMPAT) が有効である場合: CODEPAGE コンパイラー・オプション

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。)

XMLPARSE (COMPAT) の場合:

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。例:

```
<?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。

エンコード宣言は、以下のいずれかの方法で指定してください。

- 先行ゼロ付きまたは先行ゼロなしの CCSID 番号に、接頭部として次のいずれかのストリング (大文字および小文字を任意に組み合わせ可能) を指定します。
 - IBM-
 - IBM_
 - CCSID-
 - CCSID_
- 次の表にリストした別名のいずれかを使用します。別名は、大文字と小文字を混合させてコーディングできます。

表 81. XML エンコード宣言の別名	
CCSID	サポートされる別名
037	EBCDIC-CP-US、EBCDIC-CP-CA、EBCDIC-CP-WT、EBCDIC-CP-NL
500	EBCDIC-CP-BE、EBCDIC-CP-CH
1200	UTF-16

表 81. XML エンコード宣言の別名 (続き)	
CCSID	サポートされる別名
1208	UTF-8

XML 構文解析にサポートされている CCSID について詳しくは、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。

関連概念

705 ページの『XML 入力文書エンコード』

関連タスク

708 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

712 ページの『エンコード競合の処理』

関連参照

704 ページの『XML 文書のエンコード方式』

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 CCSID の特殊文字とそれぞれの 16 進値を示します。

表 82. 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

もし、XMLPARSE (XMLSS) コンパイラオプションが有効になっている場合は、Unicode でエンコードされた XML 文書を解析できる UTF-8 他の XML ドキュメントを解析するのと同様の方法で。ただし、追加でいくつかの要件が適用されます。

このタスクについて

UTF-8 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 ステートメントの ENCODING 句に CCSID 1208 を指定する必要があります。

```
XML PARSE xml-document
  WITH ENCODING 1208
  PROCESSING PROCEDURE xml-event-handler
END-XML
```

WORKING-STORAGE または LOCAL-STORAGE に、英数字データ項目または英数字グループ項目として、xml-document を定義します。

コーディングしない場合は RETURNING NATIONAL フレーズの XML PARSE 声明では、パーサーは、XML 文書のフラグメントを英数字の特殊文字で返します。レジスター XML-TEXT、XML-NAMESPACE、そして XML-NAMESPACE-PREFIX。

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。

より確実に処理できます UTF-8 文書の断片を指定することで RETURNING NATIONAL フレーズの XML PARSE 声明。RETURNING NATIONAL 句を使用すると、XML 文書フラグメントは、効率的に UTF-16 エンコードに変換され、国別特殊レジスター XML-NTEXT、XML-NNAMESPACE、および XMLNNAMESPACE-PREFIX でアプリケーションに返されます。その後、国別データ項目の XML テキスト・フラグメントを処理することができます。(国別データ項目の UTF-16 エンコードによって、COBOL での Unicode 処理が非常に簡素化します。)

次のコードフラグメントは、ENCODING フレーズと RETURNING NATIONAL 解析するためのフレーズ UTF-8 XML ドキュメント:

```
XML PARSE xml-document
  WITH ENCODING 1208 RETURNING NATIONAL
  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
```

関連概念

[696 ページの『XML-TEXT および XML-NTEXT』](#)

[697 ページの『XML-NAMESPACE および XML-NNAMESPACE』](#)

[697 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』](#)

関連タスク

[162 ページの『UTF-16 \(国別\) データ・タイプを使用して UTF-8 データを処理』](#)

[690 ページの『XML 文書の構文解析』](#)

[707 ページの『エンコードの指定』](#)

関連参照

[462 ページの『XMLPARSE』 \(コンパイラー・オプション\)](#)

[704 ページの『XML 文書のエンコード方式』](#)

XML PARSE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

XML PARSE の例外処理

XML 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、XML 例外イベントをシグナル通知します。発生する可能性のある特定の例外コードとその後の対処方法は、XMLPARSE コンパイラオプション。

このタスクについて

XMLPARSE (XMLSS) の場合:

戻りコードと理由コード: 例外コードは、パーサーが生成する戻りコードと理由コードで構成されます。戻りコードと理由コードはそれぞれハーフワード・バイナリー値です。XML-CODE の値はそれらの 2 つの値を連結したものです。

例えば、次の XML 文書は、エレメント終了タグ `mmsg` がエレメント開始タグ `msg` と一致していないので、正しい形式ではありません。

```
<msg>Hello</mmsg>
```

戻りコードは 16 進数です 000C (XRC_NOT_WELL_FORMED)、理由コードは 16 進数 3035 (XRSN_ENDTAG_NAME_MISMATCH)、検証せずにドキュメントを解析する場合。この 2 つの値を連結した 16 進数の 000C3035 が、XML-CODE 特殊レジスターで、処理プロシージャに返されます。

検証を伴って文書を構文解析した場合、整形エラーに関して XML-CODE に返される値は、検証を行わずに構文解析したときに同じエラーに対して返される値と異なります。検証エラーに対して z/OS XML System Services パーサーが生成する戻りコードは 24 (16 進数の 0018) です。

生成される戻りコードと理由コードについて詳しくは、XMLPARSE(XMLSS) が有効な場合の例外に関する関連参照を参照してください。

XMLPARSE(XMLSS) が有効である場合、処理プロシージャは例外イベントを処理できず、構文解析を再開させることができません。処理プロシージャが例外イベントからパーサーに戻ると、パーサーはイベントをそれ以上シグナル通知しません。パーサーは、指定された文に制御を移します。ON EXCEPTION のフレーズ XML PARSE 声明。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。XML-CODE には、パーサーが設定した元の例外コードが含まれます。

構文解析で例外が発生しない場合、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。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。XML-CODE には、ゼロが含まれます。

XMLPARSE (COMPAT) の場合:

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を処理するには、以下の手順に従ってください。

手順

1. XML-CODE の内容を検査します。
2. 例外を適切に処理します。
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。

タスクの結果

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 (エンコードの競合が検出されたことを示します) のみです。

- 50 から 99
- 100,001 から 165,535

例外コード 1 から 49: 処理プロシージャでは、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。)

制約事項: 互換モード COBOL XML パーサーは、すべての追加例外イベントをシグナル通知できない場合があります。例外数は、XML PARSE イベント・トークン配列の残りスペース (おそらく 8192 イベント) に限定されます。

例外コードが 1 から 49 の範囲内の例外が発生した後の構文解析終了時には、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。コーディングしなかった場合 ON EXCEPTION フレーズの最後に制御が渡されます XML PARSE 声明。XML-CODE には、パーサーが最新の例外に設定したコードが含まれます。

上記の範囲内の例外コードを持つ例外を除くすべての例外については、パーサーはそれ以上のイベントを通知せず、指定されたステートメントに制御を渡します。ON EXCEPTION フレーズ。XML-CODE 元の例外コードが含まれている場合でもあなたが設定した XML-CODE でパーサーに制御を返す前の処理手順。

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を移します。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。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移動します。XML-CODE には、ゼロが含まれます。

関連概念

[694 ページの『XML-CODE』](#)

[705 ページの『XML 入力文書エンコード』](#)

[711 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

[691 ページの『XML を処理するためのプロシージャの作成』](#)

[698 ページの『検証を伴う XML 文書の構文解析』](#)

[712 ページの『エンコード競合の処理』](#)

関連参照

[462 ページの『XMLPARSE』 \(コンパイラ・オプション\)](#)

[704 ページの『XML 文書のエンコード方式』](#)

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

[874 ページの『XMLPARSE\(COMPAT\) が有効な場合の XML PARSE 例外』](#)

[z/OS XML System Services User's Guide and Reference](#)

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ジャに渡します。

パーサーは、以下の情報を特殊レジスターに入れて処理プロシージャに渡します。

- 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。

例外コードについては、関連資料で説明されています。参照について XML PARSE 例外。

- 致命的な例外に関しては、例外が検出されたポイントまでの (そのポイントを含む) 文書テキストが XML-TEXT または XML-NTEXT に含まれます。
- 宣言されていない接頭部を使用していたために発行された警告例外に関しては、完全修飾の属性名またはエレメント名が XML-TEXT または XML-NTEXT に含まれます。すなわち、これらの名前には、宣言されていない接頭部と、区切り記号のコロン (:) が含まれます。
- XMLPARSE(COMPAT) が有効である場合には、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。
- XMLPARSE(XMLSS) が有効である場合には、XML-TEXT または XML-NTEXT には、エラーまたは異常の検出ポイントまでの文書テキストが含まれています。XML 文書を 1 セグメントずつ処理した場合には、該当する特殊レジスターには現在のセグメントのみが含まれます。

他のすべての XML 特殊レジスターは空で、長さがゼロになります。

XMLPARSE(XMLSS) の場合:

処理プロシージャで XML-CODE をゼロに設定したとしても、致命的な例外の後では、構文解析は続行できません。処理手順からパーサーに戻ると、パーサーは制御を ON EXCEPTION 指定されていない場合は、パーサーは制御を文末に移します。XML PARSE 声明。XML-CODE には、パーサーが設定した元の例外コードが含まれます。

XMLPARSE (COMPAT) の場合:

例外コードが次のいずれかの範囲内にある場合には、構文解析を続行できるように処理プロシージャーが例外を処理できる場合があります。

- 1 から 99
- 100,001 から 165,535

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。

エンコード競合: 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。

例外コード 1 から 49: 例外コードが 1 から 49 までの範囲にある例外は、XML 仕様によって致命的エラーになります。したがって、処理プロシージャ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ません。ただし、パーサーは、文書の終わりに到達するまで、または既存の XML EVENT トークン配列が使い尽くされるまで、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

[693 ページの『XML イベント』](#)

[694 ページの『XML-CODE』](#)

[705 ページの『XML 入力文書エンコード』](#)

関連タスク

[701 ページの『XML 文書を 1 セグメントずつ構文解析』](#)

[709 ページの『XML PARSE の例外処理』](#)

[712 ページの『エンコード競合の処理』](#)

[713 ページの『XML 構文解析の終了』](#)

関連参照

[462 ページの『XMLPARSE』 \(コンパイラー・オプション\)](#)

[704 ページの『XML 文書のエンコード方式』](#)

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

[874 ページの『XMLPARSE \(COMPAT\) が有効な場合の XML PARSE 例外』](#)

[z/OS XML System Services User's Guide and Reference](#)

[XML specification](#)

エンコード競合の処理

エンコード競合例外を処理する方法は、XMLPARSE コンパイラー・オプションの設定に依存します。

このタスクについて

XMLPARSE (XMLSS) の場合:

パーサーは、エンコード競合例外やその他のタイプの例外の後、処理を続行しません。処理プロシージャーで XML-CODE の値に加えた変更はすべて無視されます。パーサーが XML PARSE ステートメントに戻る際の XML-CODE の値は、パーサーが設定した、元の例外コードです。

XMLPARSE (COMPAT) の場合:

ご使用の処理プロシージャーが、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。解析データ項目が英数字で、例外コードが XML-CODE100,001 - 165,535 の範囲内であるドキュメントのコード ページ (エンコーディング宣言で指定) が外部のコード ページ情報と競合していることを示します。

この特別なケースでは、100,000 を減算することで文書のコードページを使用して解析することを選択できます。の値から XML-CODE。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、外部コード・ページ (CODEPAGE コンパイラー・オプションの値) 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 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

[694 ページの『XML-CODE』](#)

[705 ページの『XML 入力文書エンコード』](#)

[711 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

[709 ページの『XML PARSE の例外処理』](#)

関連参照

[462 ページの『XMLPARSE』 \(コンパイラー・オプション\)](#)

[704 ページの『XML 文書のエンコード方式』](#)

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

[874 ページの『XMLPARSE\(COMPAT\) が有効な場合の XML PARSE 例外』](#)

z/OS XML System Services User's Guide and Reference

XML 構文解析の終了

プロシージャが通常の 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 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。

このタスクについて

この技法は、処理プロシージャで文書を十分に検査済みである場合、あるいは処理プロシージャが文書に何らかの異常を検出し、それ以上処理を続けても意味がない場合に使用できます。

このように構文解析を終了した場合、パーサーはそれ以上 XML イベント (例外イベントを含む) をシグナル通知しません。管理権は ON EXCEPTION のフレーズ XML PARSE そのフレーズが指定されている場合、ステートメント。

ON EXCEPTION 句の命令ステートメントでは、XML-CODE に -1 が入っているかどうかをテストすることによって、構文解析が意図的に終了されたかどうかを判断します。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。

もし、XMLPARSE (COMPAT) コンパイラオプションが有効になっている場合は、XML の後に解析を終了することもできる EXCEPTION イベントは、処理手順からパーサーに戻り、値を変更せずに XML-CODE。結果は意図的な終了の結果と似ていますが、パーサーは XML PARSE 声明文 XML-CODE 元の例外コードが含まれます。

XMLPARSE (XMLSS) オプションが有効である場合には、例外イベント発生後に構文解析は常に終了します。

関連概念

[694 ページの『XML-CODE』](#)

[711 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

[691 ページの『XML を処理するためのプロシージャーの作成』](#)

[709 ページの『XML PARSE の例外処理』](#)

XML PARSE の例

以下で参照される例では、XML PARSE ステートメントのさまざまな使用方法を説明します。

これらの例を使用して、基本的な使用方法を理解します XML PARSE そして XMLPARSE (XMLSS) 名前空間を含む文書の解析、文書を 1 セグメントずつ解析するなどの特殊な用途、スキーマに対する検証を伴う文書の解析。

[714 ページの『例: 単純な文書の構文解析』](#)

[715 ページの『例: XML の処理用プログラム』](#)

[719 ページの『例: 名前空間を使用する XML 文書の構文解析』](#)

[722 ページの『例: XML 文書を 1 セグメントずつ構文解析』](#)

[724 ページの『例: 検証を伴う XML 文書の構文解析』](#)

例: 単純な文書の構文解析

この例では、単純な XML 文書の構文解析によるイベントのフローおよび特殊レジスター XML-TEXT の内容を示します。

COBOL プログラムには次のようなデータ項目 Doc の XML 文書が含まれていると仮定します。

```
<?xml version="1.0"?><msg type="short">Hello, World!</msg>
```

次のコードフラグメントは、XML PARSE 解析のためのステートメント Doc、および処理手順、P XML イベントを処理するためのもの:

```
XML Parse Doc
Processing procedure P
P. Display XML-Event XML-Text.
```

処理プロシージャーでは、パーサーが構文解析中にシグナル通知する各イベントの XML-EVENT および XML-TEXT のコンテンツが表示されます。次の表に、イベントおよびテキストを示します。

表 83. XML イベントおよび特殊レジスター	
XML-EVENT	XML-TEXT
START-OF-DOCUMENT	
VERSION-INFORMATION	1.0
START-OF-ELEMENT	msg
ATTRIBUTE-NAME	type
ATTRIBUTE-CHARACTERS	short
CONTENT-CHARACTERS	Hello, World!
END-OF-ELEMENT	msg
END-OF-DOCUMENT	

関連概念

[693 ページの『XML イベント』](#)

[696 ページの『XML-TEXT および XML-NTEXT』](#)

例: XML の処理用プログラム

以下の例では、XML 文書の構文解析と、各種 XML イベントおよびその関連テキスト・フラグメントを報告する処理プロシーチャーを示しています。

構文解析のフローを簡単に追えるように、プログラム・ソースに XML 文書が示されています。プログラムの出力と XMLPARSE (XMLSS) そして XMLPARSE (COMPAT) 事実上例の後に表示されます。

パーサーと処理プロシーチャーとの間の相互作用を理解し、イベントと文書フラグメントを突き合わせるために、XML 文書とプログラムの出力を比較してください。

```
Process codepage(1047)
  Identification division.
    Program-id. XMLSAMPL.
  Data division.
    Working-storage section.
*****
* XML document data, encoded as initial values of data items.      *
*****
  1 xml-document-data.
    2 pic x(39) value '<?xml version="1.0" encoding="IBM-1047"'.
    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(33) value '<bread type="baker&apos;s best"/>'.
    2 pic x(36) value '<?spread We'll use real mayonnaise?>'.
    2 pic x(29) value '<meat>Ham & turkey</meat>'.
    2 pic x(34) value '<filling>Cheese, lettuce, tomato, '.
    2 pic x(32) value 'and that's all, Folks!</filling>'.
    2 pic x(25) value '<![CDATA[We should add a '.
    2 pic x(20) value '<relish> element!]]>'.
    2 pic x(28) value '<listprice>$4.99</listprice>'.
    2 pic x(25) value '<discount>0.10</discount>'.
    2 pic x(31) value '</sandwich>'.
*****
* XML document, represented as fixed-length records.                *
*****
  1 xml-document redefines xml-document-data.
  2 xml-segment pic x(40) occurs 10 times.
  1 xml-segment-no comp pic s9(4).
  1 content-buffer pic x(100).
  1 current-element-stack.
  2 current-element pic x(30) occurs 10 times.
*****
* Sample data definitions for processing numeric XML content.        *
*****
  1 element-depth comp pic s9(4).
  1 discount computational pic 9v99 value 0.
  1 display-price pic $$$9.99.
  1 filling pic x(4095).
  1 list-price computational pic 9v99 value 0.
  1 ofr-ed pic x(9) justified.
  1 ofr-ed-1 redefines ofr-ed pic 999999.99.
  Procedure division.
    Mainline section.
      Move 1 to xml-segment-no
      Display 'Initial segment {' xml-segment(xml-segment-no) '}'
      Display ' '
      XML parse xml-segment(xml-segment-no)
        processing procedure XML-handler
          On exception
            Display 'XML processing error, XML-Code=' XML-Code ' .'
            Move 16 to return-code
            Goback
          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 '}'
        Add 1 to element-depth
        Move XML-Text to current-element(element-depth)
    When 'CONTENT-CHARACTERS'
        Display 'Content characters: {' XML-Text '}'
* ==> In general, a split can occur for any element or attribute
* ==> data, but in this sample, it only occurs for "filling"...
    If xml-information = 2 and
        current-element(element-depth) not = 'filling'
        Display 'Unexpected split in content for element '
            current-element(element-depth)
        Move -1 to xml-code
    End-if
* ==> Transform XML content to operational COBOL data item...
    Evaluate current-element(element-depth)
    When 'filling'
* ==> After reassembling separate pieces of character content...
    String xml-text delimited by size into
        content-buffer with pointer tally
    On overflow
        Display 'content buffer ('
            length of content-buffer
            ' bytes) is too small'
        Move -1 to xml-code
    End-string
    Evaluate xml-information
    When 2
        Display ' Character data for element "filling" '
            'is incomplete.'
        Display ' The partial data was buffered for '
            'content assembly.'
    When 1
        subtract 1 from tally
        move content-buffer(1:tally) to filling
        Display ' Element "filling" data (' tally
            ' bytes) is now complete:'
        Display ' {' filling(1:tally) '}'
    End-evaluate
    When 'listprice'
* ==> Using function NUMVAL-C...
        Move XML-Text to content-buffer
        Compute list-price =
            function numval-c(content-buffer)
    When 'discount'
* ==> Using de-editing of a numeric edited item...
        Move XML-Text to ofr-ed
        Move ofr-ed-1 to discount
    End-evaluate
    When 'END-OF-ELEMENT'
        Display 'End element tag: {' XML-Text '}'
        Subtract 1 from element-depth
    When 'END-OF-INPUT'
        Display 'End of input'
        Add 1 to xml-segment-no
        Display ' Next segment: {' xml-segment(xml-segment-no)
            '}'
        Display ' '
        Move 1 to xml-code
    When 'START-OF-DOCUMENT'
        Display 'Start of document'
        Move 0 to element-depth
        Move 1 to tally
    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 section'
    When 'END-OF-CDATA-SECTION'
        Display 'End of CData section'
    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 tally = function length (XML-Text)
        Display 'Exception ' XML-Code ' at offset ' tally '.'
    When other
        Display 'Unexpected XML event: ' XML-Event '.'
End-evaluate
.
End program XMLSAMPL.

```

解析からの出力と XMLPARSE(XMLSS)

以下の出力では、構文解析中に発生したイベントに、どの文書フラグメントが関連付けられていたかを確認することができます。

```

Initial segment {<?xml version="1.0" encoding="ibm-1047" }

Start of document
End of input
  Next segment: {standalone="yes"?><!--This document is j}

Version: {1.0}
Encoding: {ibm-1047}
Standalone: {yes}
Comment: {This document is j}
End of input
  Next segment: {ust an example--><sandwich><bread type="}

Comment: {ust an example}
Start element tag: {sandwich}
End of input
  Next segment: {baker&apos;s best"/><?spread We'll use r}

Start element tag: {bread}
Attribute name: {type}
Attribute value characters: {baker's best}
End element tag: {bread}
PI target: {spread}
PI data: {We'll use r}
End of input
  Next segment: {eal mayonnaise?><meat>Ham &amp; turkey</}

PI target: {spread}
PI data: {eal mayonnaise}
Start element tag: {meat}
Content characters: {Ham & turkey}
End of input
  Next segment: {meat}<filling>Cheese, lettuce, tomato, a}

End element tag: {meat}
Start element tag: {filling}
Content characters: {Cheese, lettuce, tomato, a}
  Character data for element "filling" is incomplete.
  The partial data was buffered for content assembly.
End of input
  Next segment: {nd that's all, Folks!</filling><![CDATA[}

Content characters: {nd that's all, Folks!}
  Element "filling" data (00047 bytes) is now complete:
  {Cheese, lettuce, tomato, and that's all, Folks!}
End element tag: {filling}
End of input
  Next segment: {We should add a <relish> element!]]><lis}

Start of CData section
Content characters: {We should add a <relish> element!}
End of CData section
End of input
  Next segment: {tprice>$4.99</listprice><discount>0.10</}

```

```

Start element tag: {listprice}
Content characters: {$4.99}
End element tag: {listprice}
Start element tag: {discount}
Content characters: {0.10}
End of input
  Next segment: {discount}</sandwich>                                }

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!

```

解析からの出力と XMLPARSE (COMPAT)

以下の出力では、構文解析中に発生したイベントに、どの文書フラグメントが関連付けられていたかを確認することができます。

```

Start of document
Version: {1.0}
Encoding: {IBM-1047}
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!

```

関連概念

[693 ページの『XML イベント』](#)

関連タスク

703 ページの『XML-INFORMATION 特殊レジスターを使用した分割の処理』

関連参照

462 ページの『XMLPARSE』(コンパイラー・オプション)

XML-EVENT (*Enterprise COBOL for z/OS* 言語解説書)

例: 名前空間を使用する XML 文書の構文解析

この例では、名前空間と名前空間接頭部を使用する文書の構文解析を示します。プログラムは、XMLPARSE(XMLSS) コンパイラー・オプションを使用してコンパイルされる必要があります。

プログラムでは、名前空間 ID および名前空間接頭部を使用して、エレメント名および属性名を修飾しています。この修飾によって、同じ名前を複数のコンテキストで使うことが可能になります。author の title (作者の称号: Mr) および book の title (本のタイトル: Writing COBOL for Fun and Profit) として title が使用されています。

サンプル XML 文書

次の XML 文書には、複数の名前空間宣言 (デフォルト名前空間、および接頭部が付いた 3 つの名前空間 ID (bk、pi、および isbn)) が含まれています。エレメント comment で、デフォルト名前空間が空ストリングに設定されていること (xmlns='') に注目してください。これによってデフォルト名前空間が"非宣言化"され、その結果、デフォルト名前空間が存在しないようになります。

```
<section
  xmlns="http://www.ibm.com/events"
  xmlns:bk="urn:loc.gov:books"
  xmlns:pi="urn:personalInformation"
  xmlns:isbn='urn:ISBN:0-395-36341-6'>
  <title>Book-Signing Event</title>
  <signing>
    <bk:author pi:title="Mr" pi:name="Jim Ross"/>
    <book bk:title="Writing COBOL for Fun and Profit" isbn:number="0426070806"/>
    <comment xmlns=''>What a great issue!</comment>
  </signing>
</section>
```

構文解析の結果

次の表に、処理プロシージャがパーサーから受け取る一連のイベントと、関連付けられた XML 特殊レジスターの内容を示します。

表 84. XML イベントおよび特殊レジスター			
XML - EVENT	XML - TEXT	XML - NAMESPACE - PREFIX	XML - NAMESPACE
START-OF-DOCUMENT			
START-OF-ELEMENT	section		http://www.ibm.com/events
NAMESPACE-DECLARATION			http://www.ibm.com/events
NAMESPACE-DECLARATION		bk	urn:loc.gov:books
NAMESPACE-DECLARATION		pi	urn:personalInformation
NAMESPACE-DECLARATION		isbn	urn:ISBN:0-395-36341-6

表 84. XML イベントおよび特殊レジスター (続き)

XML - EVENT	XML - TEXT	XML - NAMESPACE - PREFIX	XML - NAMESPACE
START-OF-ELEMENT	title		http://www.ibm.com/events
CONTENT-CHARACTERS	Book-Signing Event		
END-OF-ELEMENT	title		http://www.ibm.com/events
START-OF-ELEMENT	signing		http://www.ibm.com/events
START-OF-ELEMENT	author	bk	urn:loc.gov:books
ATTRIBUTE-NAME	title	pi	urn:personalInformation
ATTRIBUTE-CHARACTERS	Mr		
ATTRIBUTE-NAME	name	pi	urn:personalInformation
ATTRIBUTE-CHARACTERS	Jim Ross		
END-OF-ELEMENT	author	bk	urn:loc.gov:books
START-OF-ELEMENT	book		http://www.ibm.com/events
ATTRIBUTE-NAME	title	bk	urn:loc.gov:books
ATTRIBUTE-CHARACTERS	Writing COBOL for Fun and Profit		
ATTRIBUTE-NAME	number	isbn	urn:ISBN:0-395-36341-6
ATTRIBUTE-CHARACTERS	0426070806		
END-OF-ELEMENT	book		http://www.ibm.com/events
START-OF-ELEMENT	comment		
NAMESPACE-DECLARATION			
CONTENT-CHARACTERS	What a great issue!		
END-OF-ELEMENT	comment		
END-OF-ELEMENT	signing		http://www.ibm.com/events
END-OF-ELEMENT	section		http://www.ibm.com/events
END-OF-DOCUMENT			

宣言されていない名前空間接頭部に関する XML PARSE 例

次の XML 文書には、宣言されていない名前空間接頭部が含まれています。

```
Identification division.
  Program-id. XMLup.
Data division.
  Working-storage section.
    1 d.
      2 pic x(40) value '<pfx0:root xmlns:pfx1="http://whatever">'.
      2 pic x(19) value '<pfx1:localElName1>'.
      2 pic x(20) value '<pfx2:localElName2/>'.
      2 pic x(40) value '<pfx3:localElName3 pfx4:localAtName4="">'.
      2 pic x(02) value 'c1'.
      2 pic x(41) value '<pfx5:localElName5 pfx6:localAtName6=""/>'.
      2 pic x(24) value 'c2</pfx3:localElName3>c3'.
      2 pic x(32) value '</pfx1:localElName1></pfx0:root>'.
  Procedure division.
    main.
      display 'XML document: ' d
      display ' '
      xml parse d processing procedure h
      goback.
    h.
      if xml-event = 'EXCEPTION'
        display ' '
      end-if
      display xml-event xml-code '|' xml-text '|'
      display xml-namespace-prefix '|'
      display xml-namespace '|'
      if xml-event = 'EXCEPTION' and xml-code = 264192 or 264193
        move 0 to xml-code
      end-if
    .
  End program XMLup.
```

宣言されていない名前空間接頭部を持つ XML 文書の構文解析結果

次の表には、処理プロシージャがパーサーから受け取る一連のイベントがリストされ、関連 XML 特殊レジスターの内容が示されています。

表 85. 宣言されていない名前空間接頭部を持つ XML 文書を解析することで得られる XML イベントと特殊レジスター				
XML - EVENT	XML - CODE	XML - TEXT	XML - NAMESPACE - PREFIX	XML - NAMESPACE
START-OF-DOCUMENT	000000000			
EXCEPTION	000264193	pfx0:root		
START-OF-ELEMENT	000000000	root	pfx0	
NAMESPACE-DECLARATION	000000000		pfx1	http://whatever
START-OF-ELEMENT	000000000	localElName1	pfx1	http://whatever
EXCEPTION	000264193	pfx2:localElName2		
START-OF-ELEMENT	000000000	localElName2	pfx2	
END-OF-ELEMENT	000000000	localElName2	pfx2	
EXCEPTION	000264193	pfx3:localElName3		

表 85. 宣言されていない名前空間接頭部を持つ XML 文書を解析することで得られる XML イベントと特殊レジスター (続き)

XML - EVENT	XML - CODE	XML - TEXT	XML - NAMESPACE - PREFIX	XML - NAMESPACE
START-OF-ELEMENT	000000000	localElName3	pfx3	
EXCEPTION	000264192	pfx4:localAtName4		
ATTRIBUTE-NAME	000000000	localAtName4	pfx4	
ATTRIBUTE-CHARACTERS	000000000			
CONTENT-CHARACTERS	000000000	c1		
EXCEPTION	000264193	pfx5:localElName5		
START-OF-ELEMENT	000000000	localElName5	pfx5	
EXCEPTION	000264192	pfx6:localAtName6		
ATTRIBUTE-NAME	000000000	localAtName6	pfx6	
ATTRIBUTE-CHARACTERS	000000000			
END-OF-ELEMENT	000000000	localElName5	pfx5	
CONTENT-CHARACTERS	000000000	c2		
END-OF-ELEMENT	000000000	localElName3	pfx3	
CONTENT-CHARACTERS	000000000	c3		
END-OF-ELEMENT	000000000	localElName1	pfx1	http://whatever
END-OF-ELEMENT	000000000	root	pfx0	
END-OF-DOCUMENT	000000000			

XML イベント・セットについて詳しくは、XML - EVENT に関する関連参照を参照してください。

関連概念

[693 ページの『XML イベント』](#)

[696 ページの『XML-TEXT および XML-NTEXT』](#)

[697 ページの『XML-NAMESPACE および XML-NNAMESPACE』](#)

[697 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』](#)

関連参照

[462 ページの『XMLPARSE』](#) (コンパイラー・オプション)

XML-EVENT (*Enterprise COBOL for z/OS* 言語解説書)

例: XML 文書を 1 セグメントずつ構文解析

この例では、文書の 1 セグメントずつの構文解析を示します。プログラムは、XMLPARSE (XMLSS) コンパイラー・オプションを使用してコンパイルされる必要があります。

例では、ファイルの XML コンテンツ、XML テキストを読み取りパーサーに送信するプログラム、および入力レコードを構文解析した結果の一連のイベントが示されています。

infile の内容

1 セグメントずつ構文解析される XML 文書は、以下のような infile ファイルに含まれています。

```
<?xml version='1.0'?>
<Tagline>
COBOL is the language of the future!
</Tagline>
```

プログラム PARSESEG

プログラム PARSESEG では、XML 文書のセグメント (レコード) を infile ファイルから読み取り、そのレコードをパーサーに XML PARSE ステートメントを使用して渡しています。パーサーは、XML イベントごとに XML テキストを処理して処理プロシージャーに制御を渡します。処理プロシージャーが各イベントを処理し、パーサーに返します。

セグメントの最後で、パーサーは XML-EVENT を END-OF-INPUT に設定し、XML-CODE をゼロに設定し、制御を処理プロシージャーに渡します。処理プロシージャーは次の XML レコード読み取って構文解析データ項目に入れ、XML-CODE を 1 に設定してパーサーに返します。

このように、処理プロシージャーとパーサーは、READ ステートメントがファイル終了状況コードを返すまで、交互に処理を行います。処理プロシージャーは、セグメント処理の最後を示す、まだゼロに設定された XML-CODE をパーサーに返します。

```
Identification division.
Program-id. PARSESEG.
Environment division.
Input-output section.
File-control.
    Select Input-XML
    Assign to infile
    File status is Input-XML-status.
Data division.
File section.
FD Input-XML
    Record is varying from 1 to 255 depending on Rec-length
    Recording mode V.
1 fdrec.
2 pic X occurs 1 to 255 depending on Rec-length .
Working-storage section.
1 Event-number comp pic 99.
1 Rec-length comp-5 pic 9(4).
1 Input-XML-status pic 99.
Procedure division.
    Open input Input-XML
    If Input-XML-status not = 0
        Display 'Open failed, file status: ' Input-XML-status
        Goback
    End-if
    Read Input-XML
    If Input-XML-status not = 0
        Display 'Read failed, file status: ' Input-XML-status
        Goback
    End-if
    Move 0 to Event-number
    Display 'Starting with: ' fdrec
    Display 'Event number and name      Content of XML-text'
    XML parse fdrec processing procedure Handle-parse-events
    Close Input-XML
    Goback
.
Handle-parse-events.
    Add 1 to Event-number
    Display '      ' Event-number ': ' XML-event '{' XML-text '{'
    Evaluate XML-event
        When 'END-OF-INPUT'
            Read Input-XML
            Evaluate Input-XML-status
```

```

      When 0
        Move 1 to XML-code
        Display 'Continuing with: ' fdrec
      When 10
        Display 'At EOF; no more input.'
      When other
        Display 'Read failed, file status:' Input-XML-status
        Goback
      End-evaluate
    When other
      Continue
    End-evaluate
  .
End program PARSESEG.

```

構文解析の結果

構文解析結果の表示では、処理プロシーチャーは入力の各レコードを表示し、その後の一連の XML イベントおよび XML-TEXT 内の関連テキスト・フラグメントを表示します。XML-TEXT のコンテンツは中括弧 ({}) で表示されます。空の中括弧は、XML-TEXT が空であることを示します。

イベント番号 08 の余分なゼロ長 CONTENT-CHARACTERS XML イベントに注意してください。(XML テキストを細切れに入力するとこのような例外が発生することがよくあります。)

```

Starting with:  <?xml version='1.0'?>
Event number and name      Content of XML-TEXT
01: START-OF-DOCUMENT      {}
02: VERSION-INFORMATION    {1.0}
03: END-OF-INPUT           {}
Continuing with:  <Tagline>
04: START-OF-ELEMENT       {Tagline}
05: END-OF-INPUT           {}
Continuing with:  COBOL is the language of the future!
06: CONTENT-CHARACTERS     {COBOL is the language of the future!}
07: END-OF-INPUT           {}
Continuing with:  </Tagline>
08: CONTENT-CHARACTERS     {}
09: END-OF-ELEMENT         {Tagline}
10: END-OF-DOCUMENT        {}

```

検出された XML イベントの詳細説明については、XML-EVENT に関する関連参照を参照してください。

関連参照

462 ページの『XMLPARSE』(コンパイラー・オプション)
XML-EVENT (Enterprise COBOL for z/OS 言語解説書)

例: 検証を伴う XML 文書の構文解析

この例では、スキーマと照合して検証する複数の XML 文書の構文解析と、各文書の構文解析後にパーサーが生成する戻りコードと理由コードを収集する処理プロシーチャーを示します。すべての XML 文書は整形式の文書ですが、妥当な文書であるとは限りません。

プログラムは、XMLPARSE(XMLSS) コンパイラー・オプションを使用してコンパイルされる必要があります。

この例では、XML スキーマに関する関連概念で説明したスキーマを使用します。

ファイル item.xsd には、スキーマがテキスト形式で含まれ、次の z/OS ユニックス コマンドによって、前処理済みのスキーマがファイル item.osr に生成されているします。

```
xsdosrg -v -o /u/HLQ/xml/item.osr /u/HLQ/xml/item.xsd
```

この例では、XML-SCHEMA 節を使用して XML スキーマ名 schema を DD 名 ddschema に関連付けます。
次の DD ステートメントは DD 名を外部に関連付けます z/OS ユニックススキーマを含むファイル:

```
//GO.DDSCHEMA DD PATH='/u/HLQ/xml/item.osr'
```

プログラム ValidCk

```
Identification division.
  Program-id. ValidCk.
Environment division.
  Configuration section.
    Special-names.
      xml-schema schema is 'ddschema'.
Data division.
  Working-storage section.
    1 xml-decode.
      2 rtn comp Pic 9(2).
      2 rsu comp-5 Pic 9(4).
      1 hv pic x(16) value '0123456789ABCDEF'.
      1 T Pic 999 COMP.
    1 xml-document-1.
      2 pic x(52) value
        '<!--Valid: the "itemName" element can be omitted-->'.
      2 pic x(31) value '<stockItem itemNumber="123-AB">'.
      2 pic x(36) value ' <quantityOnHand>1</quantityOnHand>'.
      2 pic x(12) value '</stockItem>'.
    1 xml-document-2.
      2 pic x(44)
        value '<!--Invalid: missing attribute itemNumber-->'.
      2 pic x(11) value '<stockItem>'.
      2 pic x(30) value ' <itemName>No name</itemName>'.
      2 pic x(36) value ' <quantityOnHand>1</quantityOnHand>'.
      2 pic x(12) value '</stockItem>'.
    1 xml-document-3.
      2 pic x(47)
        value '<!--Invalid: unexpected attribute warehouse-->'.
      2 pic x(46) value
        '<stockItem itemNumber="074-UN" warehouse="NJ">'.
      2 pic x(37) value ' <quantityOnHand>10</quantityOnHand>'.
      2 pic x(32) value ' <itemName>Not here!</itemName>'.
      2 pic x(12) value '</stockItem>'.
    1 xml-document-4.
      2 pic x(46)
        value '<!--Invalid: illegal attribute value 123-Ab-->'.
      2 pic x(31) value '<stockItem itemNumber="123-Ab">'.
      2 pic x(33) value ' <itemName>Paintbrush</itemName>'.
      2 pic x(37) value ' <quantityOnHand>10</quantityOnHand>'.
      2 pic x(12) value '</stockItem>'.
    1 xml-document-5.
      2 pic x(46)
        value '<!--Invalid: missing element quantityOnHand-->'.
      2 pic x(31) value '<stockItem itemNumber="074-UN">'.
      2 pic x(32) value ' <itemName>Not here!</itemName>'.
      2 pic x(12) value '</stockItem>'.
    1 xml-document-6.
      2 pic x(42)
        value '<!--Invalid: unexpected element comment-->'.
      2 pic x(31) value '<stockItem itemNumber="123-AB">'.
      2 pic x(33) value ' <itemName>Paintbrush</itemName>'.
      2 pic x(36) value ' <quantityOnHand>1</quantityOnHand>'.
      2 pic x(35) value ' <comment>Nylon bristles</comment>'.
      2 pic x(12) value '</stockItem>'.
    1 xml-document-7.
      2 pic x(46) value
        '<!--Invalid: out-of-range element value 100-->'.
      2 pic x(31) value '<stockItem itemNumber="123-AB">'.
      2 pic x(33) value ' <itemName>Paintbrush</itemName>'.
      2 pic x(38) value ' <quantityOnHand>100</quantityOnHand>'.
      2 pic x(12) value '</stockItem>'.
  Procedure division.
    m.
      xml parse xml-document-1 validating with file schema
        processing procedure p
      xml parse xml-document-2 validating with file schema
        processing procedure p
      xml parse xml-document-3 validating with file schema
```

```

        processing procedure p
xml  parse xml-document-4 validating with file schema
      processing procedure p
xml  parse xml-document-5 validating with file schema
      processing procedure p
xml  parse xml-document-6 validating with file schema
      processing procedure p
xml  parse xml-document-7 validating with file schema
      processing procedure p
goback
.
p.
  evaluate xml-event
    when 'COMMENT'
      display ' '
      display xml-text
    when 'END-OF-DOCUMENT'
      display ' Document successfully parsed.'
    when 'EXCEPTION'
      move xml-code to xml-decode
      Divide rsn by 16 giving tally remainder T
      display ' RC=' rtn ', reason=x'''
        hv(function mod(rsn / 4096 16) + 1:1)
        hv(function mod(rsn / 256 16) + 1:1)
        hv(function mod(rsn / 16 16) + 1:1)
        hv(T + 1:1) ''''
      end-evaluate
.
End program ValidCk.

```

プログラム *ValidCk* からの出力

次の出力では、ソース・プログラム内のどの XML 文書が、スキーマと照合する検証に失敗したかがわかります。

妥当でなかった文書について、パーサーは XML 例外をシグナル通知し、特殊レジスター XML-EVENT に 'EXCEPTION'、特殊レジスター XML-CODE に戻りコードと特定の理由コードが設定された状態で、処理プロセスに制御を渡しています。

```

Valid: the "itemName" element can be omitted
Document successfully parsed.

Invalid: missing attribute itemNumber
RC=24, reason=x'8613'

Invalid: unexpected attribute warehouse
RC=24, reason=x'8612'

Invalid: illegal attribute value 123-Ab
RC=24, reason=x'8809'

Invalid: missing element quantityOnHand
RC=24, reason=x'8611'

Invalid: unexpected element comment
RC=24, reason=x'8607'

Invalid: out-of-range element value 100
RC=24, reason=x'8803'

```

関連概念

[694 ページの『XML-CODE』](#)

[700 ページの『XML スキーマ』](#)

関連タスク

[698 ページの『検証を伴う XML 文書の構文解析』](#)

[709 ページの『XML PARSE の例外処理』](#)

関連参照

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

第 39 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。

このタスクについて

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 文書をエンコードするコード・ページ
- 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)
- 生成された XML 文書のユーザー定義の要素または属性名
- いくつかの指定済み条件に応じて抑止される属性または要素
- 生成された XML 出力で属性、要素、またはコンテンツとして指定される特定の項目。
- 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、WebSphere MQ にメッセージとして渡したり、または後に CICS 通信域へ変換するために伝送したり、

リンク・エディットの考慮事項: XML GENERATE ステートメントを含んだ COBOL プログラムは、AMODE 31 を使用してリンク・エディットする必要があります。

関連タスク

[727 ページの『XML 出力の生成』](#)

[732 ページの『生成される XML 出力のエンコードの制御』](#)

[732 ページの『XML GENERATE 例外の処理』](#)

[737 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 (*Enterprise COBOL for z/OS 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。

このタスクについて

```
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。

の中に DATA DIVISION、あなたはできる定義する受信識別子として英数字 (英数字グループ項目または英数字カテゴリの基本項目のいずれか) または国別 (国別グループ項目または国別カテゴリの基本項目のいずれか) として指定します。

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。

オプションで、COUNT IN XML 文字数を取得するフレーズエンコード単位 XML 出力の生成中に入力されます。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。の ATTRIBUTES このフレーズにより、生成された XML ドキュメントに含まれるすべての適格なデータ項目が、包含 XML 要素の子要素としてではなく、包含 XML 要素の属性として表現されます。データ項目が適格になるためには、データ項目は基本データ項目でなければならず、データ項目の名前は FILLER 以外でなければならず、そのデータ記述項目に OCCURS 節があってはなりません。データ項目を含む XML エレメントは、基本データ項目のすぐ上位にあるグループ・データ項目に対応しています。オプションとして、どのデータ項目を属性または要素として表現するかをより正確に制御するには、TYPE OF フレーズ。

例えば、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>
```

オプションで、ENCODING のフレーズ XML GENERATE 指定するためのステートメント国際連合生成された XML ドキュメントの。使用しない場合は ENCODING フレーズ、文書のエンコーディングは受信データ項目のカテゴリによって決定されますそして、CODEPAGE コンパイラオプション。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) またはプログラムで有効になっている CODEPAGE コンパイラ・オプション (ENCODING 句が指定されていない場合) から派生します。

例えば、以下のプログラムの抜粋では、XML-DECLARATION フレーズの XML GENERATE、エンコードを指定する CCSID 1208 (UTF-8) :

```
01 Greeting.  
   05 msg pic x(80) value 'Hello, world!'.  
   .  
   . XML Generate Doc from Greeting  
     with Encoding 1208  
     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・エレメントで定義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。

例えば、次のデータ定義を考えてみましょう。XML GENERATE 声明:

```
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。

とき XML GENERATE ステートメントが実行されるとき、プレフィックス値は有効な XML 名である必要がありますが、コロン (:); 詳細については、名前空間に関する以下の関連リファレンス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。

例えば、次のデータ定義を考えてみましょう。XML GENERATE 声明：

```
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>
```

オプションで、NAME 句をコーディングして、生成される XML 文書内に属性と要素の名前を指定することができます。属性名と要素名は、英数字または国別リテラルでなければならず、XML 1.0 標準に従う正式名でなければなりません。

例えば、次のデータ構造を考えてみましょう。XML GENERATE 声明：

```
01 Msg.  
    02 Msg-Severity pic 9 value 1.  
    02 Msg-Date pic 9999/99/99 value "2012/04/12".  
    02 Msg-Text pic X(50) value "Sell everything!".  
01 Doc pic X(500).
```

```
XML Generate Doc from Msg  
With attributes  
    Name of Msg is "Message"  
    Msg-Severity is "Severity"  
    Msg-Date is "Date"  
    Msg-Text is "Text"  
End-XML
```

結果として生成される XML 文書は以下のとおりです。

```
<Message Severity="1" Date="2012/04/12" Text="Sell everything!"></Message>
```

オプションで、SUPPRESS 句をコーディングし、特定の基準を満たすかどうかに基づいて個々のデータ項目を生成するかどうかを指定できます。

例えば、スペースとゼロを抑止する次のデータ構造および XML GENERATE ステートメントを検討してみましょう。

```
01 G.  
    02 SensitiveInfo.  
        03 SSN pic x(11) value '123-45-6789'.  
        03 HomeAddress pic x(50) value '123 Main St, Anytown, USA'.  
    02 Aarray value spaces.  
        03 A pic AAA occurs 5.  
    02 Barray value spaces.  
        03 B pic XXX occurs 5.  
    02 Carray value zeros.  
        03 C pic 999 occurs 5.  
    Move 'abc' to A(1)  
    Move 123 to C(3)  
    XML Generate Doc from G  
        Suppress SensitiveInfo  
            every nonnumeric element when space
```

```
every numeric element when zero
End-XML
```

結果として生成される XML 文書は以下のとおりです。

```
<G>
  <Aarray><A>abc</A></Aarray>
  <Carray><C>123</C></Carray>
</G>
```

オプションで、TYPE OF 句を使用して、個々のデータ項目を属性、エレメント、または内容のいずれとして表現するかを指定できます。

例えば、次のデータ構造を考えてみましょう。XML GENERATE 声明：

```
01 Msg.
  02 Msg-Severity pic 9 value 1.
  02 Msg-Date pic 9999/99/99 value "2012/04/12".
  02 Msg-Text pic X(50) value "Sell everything!".
01 Doc pic X(500).
  XML Generate Doc from Msg
  With attributes
    Type of Msg-Severity is attribute
    Msg-Date is attribute
    Msg-Text is element
End-XML
```

結果として生成される XML 文書は以下のとおりです。

```
<Msg Msg-Severity="1" Msg-Date="2012/04/12">
  <Msg-Text>Sell everything!</Msg-Text></Msg>
```

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。コード END-XML ネストする XML GENERATE 声明には ON EXCEPTION または NOT ON EXCEPTION 条件文内のフレーズ。

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は、その句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 生成中にエラーが発生しなかった場合、XML-CODE 特殊レジスタにはゼロが含まれており、制御は NOT ON EXCEPTION 指定された場合はフレーズを、XML GENERATE それ以外の場合は声明。

733 ページの『例: XML の生成』

関連概念

[Uniform Resource Identifier \(URI\): Generic Syntax](#)

関連タスク

732 ページの『生成される XML 出力のエンコードの制御』

732 ページの『XML GENERATE 例外の処理』

162 ページの『UTF-16 (国別) データ・タイプを使用して UTF-8 データを処理』

関連参照

XML GENERATE ステートメント (*Enterprise COBOL for z/OS 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 出力を生成する場合、XML GENERATE ステートメントでは、出力のエンコーディングをカテゴリ出力を受け取るデータ項目のそして、コードページ使用して WITH ENCODING フレーズの XML GENERATE 声明。

このタスクについて

指定する場合は WITH ENCODING コードページ出力文書のコード化文字セット識別子 (CCSID) を指定するフレーズ。コードページ XML ドキュメントのエンコードに関する以下の関連リファレンスで説明されているように、COBOL XML 処理でサポートされているコード ページの 1 つを識別する符号なし整数データ項目または符号なし整数リテラルを指定する必要があります。

- 生成された XML を受け取るデータ項目のカテゴリが国別である場合、WITH ENCODING 句には 1200 (Unicode UTF-16 の CCSID) を指定する必要があります。
- 受信識別子が英数字の場合、WITH ENCODING フレーズには、CCSID 1208 またはサポートされている EBCDIC コード ページの CCSID を指定する必要があります。

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 86. 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
英数字	ソースのコンパイル時に CODEPAGE コンパイラー・オプションで指定された有効なコード・ページ
国別	UTF-16 ビッグ・エンディアン (UTF-16BE、CCSID 1200)

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。

関連参照

385 ページの『CODEPAGE』

704 ページの『XML 文書のエンコード方式』

XML GENERATE ステートメント (Enterprise COBOL for z/OS 言語解説書)

XML GENERATE の操作 (Enterprise COBOL for z/OS 言語解説書)

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。

このタスクについて

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

コーディングすると ON EXCEPTION フレーズの XML GENERATE ステートメントの場合、制御は指定した命令ステートメントに移されます。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。

エラーが発生する場合、XML 出力を受け取るデータ項目が十分大きくないという問題であることがあります。この場合、XML 出力は不完全となり、XML-CODE 特殊レジスターにエラー・コード 400 が格納されます。

次のステップを実行することにより、生成された XML 出力を検査することができます。

手順

1. コード化する COUNT IN フレーズの XML GENERATE 声明。

指定するカウント・フィールドは、XML 生成時に充てんされる XML 文字エンコード・ユニットのカウントを保持します。XML 出力を国別として定義した場合、カウントは UTF-16 文字エンコード・ユニット数であり、すべての他のエンコードの場合 (UTF-8 の場合を含む)、カウントはバイト数です。

2. カウント・フィールドを参照変更長として使用して、エラー発生時点までに生成された XML 文字を含んだデータ項目を受け取るサブストリングを参照してください。

例えば、XML-OUTPUT が XML 出力を受け取るデータ項目であり、XML-CHAR-COUNT がカウント・フィールドである場合、XML-OUTPUT(1:XML-CHAR-COUNT) は XML 出力を指します。

タスクの結果

XML-CODE の内容を使用して、行うべき修正アクションを判別してください。XML 生成中に発生する可能性がある例外の一覧については、以下の関連参照を参照してください。

関連タスク

[117 ページの『データ項目のサブストリングの参照』](#)

関連参照

[880 ページの『XML GENERATE 例外』](#)

XML-CODE (*Enterprise COBOL for z/OS* 言語解説書)

例: XML の生成

以下の例は、グループ・データ項目での購入注文の作成をシミュレートし、その購入注文の XML 版を生成します。

プログラム XGFX は XML GENERATE を使用して、ソース・レコードであるグループ・データ項目 purchaseOrder から基本データ項目 xmlPO に XML 出力を生成します。ソース・レコード内の基本データ項目は、必要に応じて文字形式に変換され、変換された文字は、ソース・レコードのデータ名から派生した名前を持つ XML 属性の値として挿入されます。

XGFX はプログラム Pretty を呼び出します。このプログラムは、処理プロシージャ p を指定した XML PARSE ステートメントを使用しており、XML の内容を一層容易に検査できるよう改行とインデントを含んだ XML 出力をフォーマット設定するようになっています。

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

```

Process xmlparse(xmlss), codepage(37)
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
            If xml-namespace-prefix = space
                String '<' xml-text delimited by size
                    into buffer with pointer pose
            Else
                String '<' xml-namespace-prefix ':' xml-text
                    delimited by size into buffer with pointer pose
            End-if
            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
            If xml-namespace-prefix = space
                String xml-text '=' delimited by size into buffer
                    with pointer posd

```



```

Else
    String xml-namespace-prefix ':' xml-text '='
    delimited by size into buffer with pointer posd
End-if
Set attribute to true
When 'NAMESPACE-DECLARATION'
    If element
        String ' ' delimited by size into buffer
        with pointer posd
    Else
        String '" ' delimited by size into buffer
        with pointer posd
    End-if
    If xml-namespace-prefix = space
        String 'xmlns="' xml-namespace delimited by size
        into buffer with pointer posd
    Else
        String 'xmlns:' xml-namespace-prefix '=' xml-namespace
        delimited by size into buffer with pointer posd
    End-if
    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
        If xml-namespace-prefix = space
            String '</' xml-text '>' delimited by size
            into buffer with pointer posd
        Else
            String '</' xml-namespace-prefix ':' xml-text '>'
            delimited by size into buffer with pointer posd
        End-if
    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="IBM-037"?>
<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

687 ページの『第 38 章 XML 入力の処理』

関連参照

462 ページの『XMLPARSE』(コンパイラー・オプション)
XML GENERATE の操作 (Enterprise COBOL for z/OS 言語解説書)

XML 出力の拡張

XML 形式で表現したい情報が、すでにグループアイテムに存在する場合があります。DATA DIVISION だし、1 つ以上の要因により、その項目を直接使用して XML ドキュメントを生成することはできません。

このタスクについて

例:

- 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、外部表示には不適当なものであり、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。
- 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。

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。

こうした問題の多くに対処するための手法として優れているのは、XML GENERATE ステートメントの新しいオプション句を使用して以下を行うことです。

- 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。

- データ項目をその値に基づいて抑制することにより、生成された XML から不適切なデータ項目を除外します。

以下に示す例は、その方法を示しています。

738 ページの『例: XML 出力の拡張』

関連参照

XML GENERATE の操作 (*Enterprise COBOL for z/OS* 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 9(08).
           25  CDR-LIFE-BP-DASH   PIC X.
           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.
           25  CDR-LIFE-QTY-SPACE1 PIC X(03).
           25  FILLER              PIC X(02).
       20  CDR-LIFE-BASE-VALUE  PIC $$$9.99
           BLANK WHEN ZERO.
   15  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 (733 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>23</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>.</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>123</CDR-LIFE-QTY-SPACE>
      <CDR-LIFE-QTY-DASH>.</CDR-LIFE-QTY-DASH>
      <CDR-LIFE-QTY-SPACE1>000</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>34</CDR-LIFE-BP-SPACE>
      <CDR-LIFE-BP-DASH>.</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>234</CDR-LIFE-QTY-SPACE>
  <CDR-LIFE-QTY-DASH>.</CDR-LIFE-QTY-DASH>
  <CDR-LIFE-QTY-SPACE1>000</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>Very high!</CDR-LIFE-BASE-TOT-VALUE>
</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。また、必要なタグ名を指定する XML スキーマも存在する可能性があります。
- XML スキーマが、DATE/TIME などの COBOL 予約語であるいくつかのタグ名を必要としている可能性があります。
- エレメントであるいくつかのフィールド (CDR-LIFE-BASE-VAL-DATE や CDR-LIFE-BASE-DESC1 など) は属性にする必要があります。
- 不要なデータ、例えば、CDR-LIFE-BASE-LIT や CDR-LIFE-BASE-DTE 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 出力のこれらの特性やその他の特性は、以下のフレーズを追加することで改善できます。XML GENERATE 声明は次の通りです。

- NAME OF 句を使用して、適切なタグ名または属性名を指定する。
- TYPE OF ... IS ATTRIBUTE 句を使用して、XML エレメントではなく XML 属性にすべきフィールドを選択する。
- TYPE OF ... IS CONTENT 句を使用して、余分なサブコンポーネントのタグを抑制する。
- SUPPRESS ... WHEN 句を使用して、関心のない値が含まれているフィールドを除外する。

以下は、これらの問題に対処するための XML GENERATE ステートメントの例です。

```

XML generate Doc from CDR-LIFE-BASE-VALUES-BOX
Count in tally
Name of
  CDR-LIFE-BASE-VALUES-BOX
  is 'Base_Values'
  CDR-LIFE-BASE-VAL-DATE
  is 'Date'
  CDR-LIFE-BASE-DTE
  is 'Date'
  CDR-LIFE-BASE-VALUE-LINE
  is 'BaseValueLine'
  CDR-LIFE-BASE-DESC1
  is 'Description'
  CDR-LIFE-BASE-PRICE
  is 'BasePrice'
  CDR-LIFE-BASE-QTY
  is 'BaseQuantity'
  CDR-LIFE-BASE-VALUE
  is 'BaseValue'
  CDR-LIFE-BASE-TOT-VALUE
  is 'TotalValue'
Type of
  CDR-LIFE-BASE-VAL-DATE is attribute
  CDR-LIFE-BASE-DESC1 is attribute
  CDR-LIFE-BP-SPACE is content
  CDR-LIFE-BP-DASH is content
  CDR-LIFE-BP-SPACE1 is content
  CDR-LIFE-QTY-SPACE is content
  CDR-LIFE-QTY-DASH is content
  CDR-LIFE-QTY-SPACE1 is content
Suppress every nonnumeric when space
       every numeric when zero

```

上記のステートメントから XML を生成およびフォーマット設定した結果は、より使用に適したものになっています。

```
<Base_Values Date="01/02/03">
  <BaseValueLine Description="First">
    <Date>01/01/01</Date>
    <BasePrice>23.00</BasePrice>
    <BaseQuantity>123.000</BaseQuantity>
    <BaseValue>$765.00</BaseValue>
  </BaseValueLine>
  <BaseValueLine Description="Second">
    <Date>02/02/02</Date>
    <BasePrice>34.00</BasePrice>
    <BaseQuantity>234.000</BaseQuantity>
    <BaseValue>$654.00</BaseValue>
  </BaseValueLine>
  <TotalValue>Very high!</TotalValue>
</Base_Values>
```

COBOL 予約語 DATE は現在、出力で XML タグ名として使用できます。1 バイト・データ名では許可されていない、アクセント記号やピリオド．などの文字も使用できます。

関連参照

XML GENERATE の操作 (*Enterprise COBOL for z/OS 言語解説書*)

REPLACE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

第 6 部 オブジェクト指向プログラムの開発

第 40 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。

このタスクについて

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、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。

Enterprise COBOL を使用すると、以下を実行できます。

- 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。

Enterprise COBOL 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。

で Enterprise COBOL クラスでは、コードを書くことができます CALL 手続き型 COBOL プログラムとインターフェースするためのステートメント。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。

オブジェクト指向 COBOL プログラムや Java プログラムの開発や実行は、z/OS ユニックス 環境で行うことが推奨されます。

制約事項:

- COBOL クラス定義とメソッドには、EXEC SQL ステートメントを使用してコンパイルすることはできません。SQL コンパイラオプション。
- COBOL クラス定義とメソッドには、EXEC SQLIMS ステートメントを使用してコンパイルすることはできません。SQLIMS コンパイラオプション。
- Java 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 COBOL プログラムは、EXEC CICS ステートメントを含むことができず、CICS で実行することもできません。CICS コンパイラ・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。
- LP(64)コンパイラ・オプションが有効な場合、OO COBOL フレームワークはサポートされない。
- AMODE 64 Java では、OO COBOL フレームワークはサポートされていません。

744 ページの『例: 口座』

関連タスク

746 ページの『クラスの定義』

751 ページの『クラス・インスタンス・メソッドの定義』

759 ページの『クライアントの定義』

770 ページの『サブクラスの定義』

[774 ページの『ファクトリー・セクションの定義』](#)

[321 ページの『第 17 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』](#)

IBM COBOL ソース・プログラムのアップグレード
(Enterprise COBOL for z/OS 移行ガイド)

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。顧客は多数存在するため、Account クラスの複数インスタンスが同時に存在すると考えられます。

必要なクラスを判別後、次のステップでは、それらのクラスがそれぞれの作業を行う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

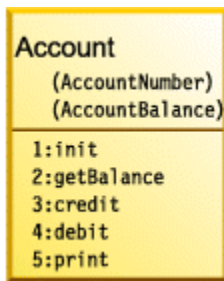
口座番号と勘定残高を表示します。

Account クラスとそのメソッドを策定すると、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

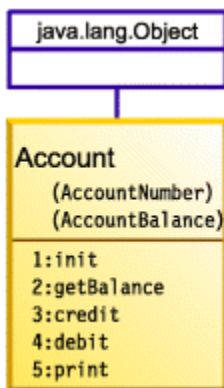
ただし、上記の例を単純化するため、口座番号と勘定残高は、Account クラスが必要とする唯一のインスタンス・データであると想定します。

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。



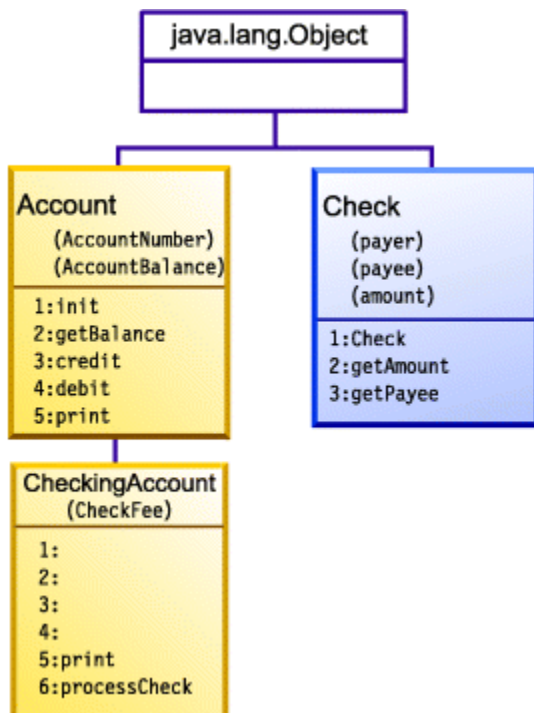
サブクラス

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。

757 ページの『例: メソッドの定義』

関連タスク

746 ページの『クラスの定義』

751 ページの『クラス・インスタンス・メソッドの定義』

770 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 そして ENVIRONMENT DIVISION、オプションのファクトリ定義とオプションのオブジェクト定義が続き、その後に END CLASS マーカー。

このタスクについて

表 87. クラス定義の構造		
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。	748 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)

表 87. クラス定義の構造 (続き)		
セクション	目的	構文
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) 748 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.

指定する場合は SOURCE-COMPUTER、OBJECT-COMPUTER、または SPECIAL-NAMES クラスの段落 CONFIGURATION SECTION これらは、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 プログラムと同じエントリで構成できる CONFIGURATION SECTION ただし、クラス CONFIGURATION SECTION 含むことはできません INPUT-OUTPUT SECTION。定義する INPUT-OUTPUT SECTION クラス レベルで定義するのではなく、それを必要とする個々のメソッドでのみ定義します。

上記のように、インスタンスデータとメソッドは DATA DIVISION そして PROCEDURE DIVISION、それぞれ、OBJECT クラス定義の段落。個々のオブジェクトインスタンスではなくクラス自体に関連付けられるデータやメソッドを必要とするクラスでは、別の DATA DIVISION そして PROCEDURE DIVISION 以内 FACTORY クラス定義の段落。

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。

751 ページの『例: クラスの定義』

関連タスク

750 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

751 ページの『クラス・インスタンス・メソッドの定義』

770 ページの『サブクラスの定義』

774 ページの『ファクトリー・セクションの定義』

5 ページの『コンピューター環境の記述』

321 ページの『第 17 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 (Enterprise COBOL for z/OS 言語解説書)

クラス定義用の CLASS-ID 段落

使用 CLASS-ID 段落の IDENTIFICATION DIVISION クラスに名前を付け、その継承情報を提供します。

このタスクについて

Identification Division. Class-id. Account inherits Base.	Required Required

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。

使用 REPOSITORY 段落の CONFIGURATION SECTION の ENVIRONMENT DIVISION スーパークラス名 (例では Base) を外部的に知られているスーパークラスの名前 (java.lang.Object ベース用)。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

[748 ページの『クラス定義用の REPOSITORY 段落』](#)

関連参照

CLASS-ID 段落 (Enterprise COBOL for z/OS 言語解説書)

ユーザー定義語 (Enterprise COBOL for z/OS 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。

このタスクについて

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. Configuration Section. Repository. Class Base is "java.lang.Object" Class Account is "Account".	Required Required Required Required Optional

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。例:

- 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。

外部クラス名を指定するには、REPOSITORY 段落は、完全修飾された段落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。ジャワクラス名。

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。
- 下線は変更されません。

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。

749 ページの『例: 外部クラス名および Java パッケージ』

関連タスク

791 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 (Enterprise COBOL for z/OS 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

ローカル・クラス名	Java パッケージ	外部クラス名
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

[748 ページの『クラス定義用の REPOSITORY 段落』](#)

関連参照

REPOSITORY 段落 (*Enterprise COBOL for z/OS* 言語解説書)

クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION

使用 WORKING-STORAGE SECTION の中に DATA DIVISION の OBJECT を説明する段落インスタンスデータ COBOL クラスに必要なデータ、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です。

このタスクについて

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。

の構文は WORKING-STORAGE SECTION 例えばデータ意味一般的にはプログラムと同じですが、次の例外があります。

- 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

[767 ページの『クラスのインスタンスの作成および初期化』](#)

[769 ページの『クラスのインスタンスの解放』](#)

[775 ページの『ファクトリー・メソッドの定義』](#)

[756 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 dll,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

321 ページの『[第 17 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](#)』
759 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

COBOL を定義するインスタンスメソッドの中に PROCEDURE DIVISION の OBJECT クラス定義の段落。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。

このタスクについて

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 88. インスタンス・メソッド定義の構造		
除算	目的	構文
IDENTIFICATION (必須)	メソッドの名前を指定する。	752 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 paragraph (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。	752 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。	753 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)

表 88. インスタンス・メソッド定義の構造 (続き)

除算	目的	構文
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。	754 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 メソッドの形式パラメータは、USING 方法のフレーズ PROCEDURE DIVISION ヘッダ。)

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。

[757 ページの『例: メソッドの定義』](#)

関連タスク

[754 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』](#)

[755 ページの『インスタンス・メソッドの多重定義』](#)

[755 ページの『インスタンス・メソッドのオーバーライド』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[772 ページの『サブクラス・インスタンス・メソッドの定義』](#)

[775 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。

このタスクについて

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

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[133 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 パラグラフ (Enterprise COBOL for z/OS 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、メソッド定義で使っ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。

このタスクについて

たとえば、Account クラスがファイルから情報を読み取るメソッドを定義した場合、Account クラスには INPUT-OUTPUT SECTION これは次のようにコード化されます:

```
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

[5 ページの『コンピューター環境の記述』](#)

関連参照

INPUT-OUTPUT セクション (*Enterprise COBOL for z/OS 言語解説書*)

クラス・インスタンス・メソッド定義用の DATA DIVISION

の DATA DIVISION インスタンス メソッドは、次の 4 つのセクションのいずれかで構成されます。FILE SECTION、LOCAL-STORAGE SECTION、WORKING-STORAGE SECTION、そして LINKAGE SECTION。

このタスクについて

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』](#)

[633 ページの『EXTERNAL 節によるデータの共用』](#)

関連参照

DATA DIVISION の概要 (*Enterprise COBOL for z/OS 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。

このタスクについて

ほとんどの COBOL 文は、PROCEDURE DIVISION コーディングできるメソッドの PROCEDURE DIVISION プログラムの。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
- 85 COBOL 標準の以下の廃止されたエレメント:
 - ALTER
 - プロシージャー名が指定されていない GOTO
 - SEGMENT-LIMIT
 - USE FOR DEBUGGING

さらに、すべての COBOL クラス定義を 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るため、SORT または MERGE ステートメントは COBOL メソッドで使用できません。

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".
```

渡された引数を取得するための USING 句:メソッドの仮パラメータを指定する場合は、USING 方法のフレーズ PROCEDURE DIVISION ヘッダ。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:メソッドの結果として返されるデータ項目がある場合は、RETURNING 方法のフレーズ PROCEDURE DIVISION ヘッダ。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

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[755 ページの『インスタンス・メソッドのオーバーライド』](#)

[755 ページの『インスタンス・メソッドの多重定義』](#)

[762 ページの『オブジェクト参照の比較および設定』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[321 ページの『第 17 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

[453 ページの『THREAD』](#)

手続き部ヘッダー (Enterprise COBOL for z/OS 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。

このタスクについて

スーパークラスのインスタンスメソッドをオーバーライドするには m1COBOL サブクラスでインスタンスメソッドを定義する m1 同じ名前を持つサブクラスで PROCEDURE DIVISION USING フレーズ (存在する場合) には、スーパークラスのメソッドと同じ数と型の仮パラメータがあります。(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 クラスはメソッドを定義している。debit だれの LINKAGE SECTION そして PROCEDURE DIVISION ヘッダーは次のようになります:

```
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す場合には、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。

メソッドの戻り値の有無と、メソッドで使用される戻り値のデータ型 PROCEDURE DIVISION RETURNING フレーズ (存在する場合) は、サブクラスのインスタンスメソッドとオーバーライドされたスーパークラスのインスタンスメソッドで同一である必要があります。

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。

[757 ページの『例: メソッドの定義』](#)

関連タスク

[754 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[766 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』](#)

[770 ページの『サブクラスの定義』](#)

[776 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。

このタスクについて

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。の `LINKAGE SECTION` そして `PROCEDURE DIVISION` ヘッダーの `init` メソッドは次のようになります。

```
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。の `LINKAGE SECTION` そして `PROCEDURE DIVISION` このヘッダー `init` メソッドは次のようになります。

```
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、必須メソッドのシグニチャーに一致する引数を渡すことで、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

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[775 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。

このタスクについて

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。)

[757 ページの『例: ゲット・メソッドのコーディング』](#)

関連タスク

[750 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』](#)

[18 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、[751 ページの『例: クラスの定義』](#)のことです。)

Account クラス

```
cb1 dll,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 ZZZZZZ999999.
*   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

[321 ページの『第 17 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。

このタスクについて

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 89. COBOL クライアントの構造		
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。 • 再帰的 (PROGRAM-ID 段落内の RECURSIVE 宣言) • 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)
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) 760 ページの『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。	761 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

オブジェクト指向構文を含む COBOL プログラムや、ジャワとともに THREAD コンパイラ オプションを使用すると、COBOL クライアントで次の言語要素を使用できなくなります。

- SORT または MERGE ステートメント
- ネストされたプログラム

THREAD コンパイラー・オプションを使用してコンパイルする プログラムは、再帰的である必要があります。各 OO COBOL クライアント・プログラムの PROGRAM-ID 段落で、RECURSIVE 節を指定しなければなりません。

[769 ページの『例: クライアントの定義』](#)

関連タスク

[321 ページの『第 17 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』](#)

[651 ページの『第 32 章 マルチスレッド化のための COBOL プログラムの準備』](#)

[785 ページの『第 41 章 Java メソッドとの通信』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[767 ページの『クラスのインスタンスの作成および初期化』](#)

[762 ページの『オブジェクト参照の比較および設定』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[777 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

[453 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は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。

このタスクについて

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.      Required
Configuration section.    Required
Source-Computer.  IBM-390.
Object-Computer.  IBM-390.
Repository.          Required
  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) として認識されます。

の SOURCE-COMPUTER、OBJECT-COMPUTER、そして SPECIAL-NAMES の段落 CONFIGURATION SECTION オプションです。

関連タスク

[748 ページの『クラス定義用の REPOSITORY 段落』](#)

関連参照

[REPOSITORY 段落 \(Enterprise COBOL for z/OS 言語解説書\)](#)

[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。

このタスクについて

```
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。ユニバーサルオブジェクト参照のコーディングは避けてください。ジャワ非常に限られた状況でのみ (**RETURNING** のフレーズ `INVOKE class-name NEW . . .` 声明)。

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

[761 ページの『LOCAL-STORAGE または WORKING-STORAGE の選択』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[760 ページの『クライアント定義用の REPOSITORY 段落』](#)

関連参照

[RETURNING 句 \(Enterprise COBOL for z/OS 言語解説書\)](#)

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。

このタスクについて

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

651 ページの『第 32 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。

このタスクについて

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。例:

```
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。例:

```
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。例:

```
Set anAccount To Self.
```

関連タスク

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[785 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。

このタスクについて

```
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。

二番目 INVOKE 文は返されたオブジェクト参照を使用する anAccount インスタンスメソッドを呼び出す credit これは Account クラスで定義されます。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、あるいはカテゴリー英字、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が参照するクラスでサポートされていない場合、1 つの severity-3 言語環境状態実行時にコードを実行しない限り、ON EXCEPTION フレーズの INVOKE 声明。

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

[764 ページの『引数の引き渡し用の USING 句』](#)

[766 ページの『戻り値の取得用の RETURNING 句』](#)

[754 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[766 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』](#)

[777 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 (*Enterprise COBOL for z/OS 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。

このタスクについて

表 90. 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 の	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	
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。

[765 ページの『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

[754 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』](#)

[755 ページの『インスタンス・メソッドの多重定義』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[623 ページの『データの受け渡し』](#)

関連参照

INVOKE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

SET ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

REDEFINES 節 (*Enterprise COBOL for z/OS 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) に割り当てた後、anObj は引数として M に渡すことができます。REDEFINES 節を使用して引数を合致させる例については、以下に参照されている例を参照してください。

797 ページの『例: COBOL で書かれた J2EE クライアント』

関連タスク

790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』

754 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 (*Enterprise COBOL for z/OS 言語解説書*)

SET ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

REDEFINES 節 (*Enterprise COBOL for z/OS 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。

このタスクについて

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 91. COBOL クライアントで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 の	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

790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』

関連参照

INVOKE ステートメント (*Enterprise COBOL for z/OS 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。

このタスクについて

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 現在のクラスのメソッドではなく、スーパークラスのメソッドを呼び出すことを示します。(SUPER 現在実行中のメソッドの呼び出しで使用されるオブジェクトへの暗黙的な参照です。

[744 ページの『例: 口座』](#)

関連タスク

[755 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 (*Enterprise COBOL for z/OS 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。

このタスクについて

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
```

コーディングするときに INVOKE . . . NEW メソッド内のステートメントで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、返されたオブジェクト参照を JNI サービスを呼び出してグローバル参照に変換する必要があります。NewGlobalRef:

```
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

[768 ページの『Java クラスのインスタンス化』](#)

[768 ページの『COBOL クラスのインスタンス化』](#)

[785 ページの『JNI サービスへのアクセス』](#)

[787 ページの『ローカル参照とグローバル参照の管理』](#)

[761 ページの『クライアント定義用の DATA DIVISION』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

関連参照

INVOKE ステートメント (*Enterprise COBOL for z/OS 言語解説書*)

Java クラスのインスタンス化

インスタンス化するにはジャワクラスでは、クラスがサポートするパラメータ化されたコンストラクタを呼び出すために、USING フレーズの INVOKE . . . NEW 直前の声明 RETURNING フレーズ、通過 BY VALUE コンストラクターのシグネチャに一致する引数の数と型。

このタスクについて

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

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

関連参照

VALUE 節 (*Enterprise COBOL for z/OS 言語解説書*)

INVOKE ステートメント (*Enterprise COBOL for z/OS 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。

このタスクについて

したがって、INVOKE . . . NEW ステートメントは、単純なインスタンス データのみを持つ COBOL クラスをインスタンス化する場合に役立ちます。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。

手順

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) コード INVOKE クラス名 NEW RETURNING *objectRef* インスタンスを作成するクラス名インスタンスデータ項目に初期値を与える VALUE 条項。

- 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。

タスクの結果

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、必要な引数 BY VALUE 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。

[777 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

[785 ページの『JNI サービスへのアクセス』](#)

[787 ページの『ローカル参照とグローバル参照の管理』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[774 ページの『ファクトリー・セクションの定義』](#)

関連参照

VALUE 節 (*Enterprise COBOL for z/OS* 言語解説書)

INVOKE ステートメント (*Enterprise COBOL for z/OS* 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。

このタスクについて

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

[787 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、[757 ページの『例: メソッドの定義』](#)に示されています。)

```
cb1 dll,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".

```

[777 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

[775 ページの『ファクトリー・メソッドの定義』](#)

[777 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』](#)

[321 ページの『第 17 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。

このタスクについて

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。P のインスタンスは、Q と S のメソッドと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しい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、スーパークラス内の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。

サブクラス定義の構造と構文はクラス定義と同一です。インスタンスデータとメソッドを DATA DIVISION そして PROCEDURE DIVISION、それぞれ、OBJECT サブクラス定義の段落。個々のオブジェクトインスタンスではなくサブクラス自体に関連付けられるデータやメソッドを必要とするサブクラスでは、別の DATA DIVISION そして PROCEDURE DIVISION 以内 FACTORY サブクラス定義の段落。

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。

[744 ページの『例: 口座』](#)

[773 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

[746 ページの『クラスの定義』](#)

[755 ページの『インスタンス・メソッドのオーバーライド』](#)

[756 ページの『属性 \(`get` および `set`\) メソッドのコーディング』](#)

[772 ページの『サブクラス・インスタンス・メソッドの定義』](#)

[774 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 \(Enterprise COBOL for z/OS 言語解説書\)](#)

サブクラス定義用の CLASS-ID 段落

このタスクについて

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.                Required
Class-id. CheckingAccount inherits Account. Required
```

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。

直接のスーパークラスの名前を REPOSITORY 段落の CONFIGURATION SECTION の ENVIRONMENT DIVISION。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

[748 ページの『クラス定義用の CLASS-ID 段落』](#)

[756 ページの『属性 \(`get` および `set`\) メソッドのコーディング』](#)

[771 ページの『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。

このタスクについて

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.                Required
Configuration Section.              Required
Repository.                          Required
```

Class CheckingAccount	is "CheckingAccount"	Optional
Class Check	is "Check"	Required
Class Account	is "Account".	Required

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。例:

- 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

[748 ページの『クラス定義用の REPOSITORY 段落』](#)

関連参照

REPOSITORY 段落 (*Enterprise COBOL for z/OS* 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

このタスクについて

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に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

[750 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規メソッドを定義することもできます。

このタスクについて

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。

[773 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

[751 ページの『クラス・インスタンス・メソッドの定義』](#)

[755 ページの『インスタンス・メソッドのオーバーライド』](#)

[755 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、[757 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)

```
cbl dll,thread,pgmname(longmixed)
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

[321 ページの『第 17 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』](#)

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[755 ページの『インスタンス・メソッドのオーバーライド』](#)

[766 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。

このタスクについて

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 92. ファクトリー定義の構造		
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)。	774 ページの『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・メソッド定義: 775 ページの『ファクトリー・メソッドの定義』

[777 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

[746 ページの『クラスの定義』](#)

[768 ページの『COBOL クラスのインスタンス化』](#)

[782 ページの『プロシージャ指向 COBOL プログラムのラッピング』](#)

[782 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

使用 WORKING-STORAGE SECTION の中に DATA DIVISION の FACTORY を説明する段落工場データ COBOL クラスに必要な、つまり、クラスのすべてのオブジェクト インスタンスによって共有される静的に割り当てられたデータです。

このタスクについて

IDENTIFICATION DIVISION 宣言を直前に指定する必要がある 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・メソッドの定義の始まりを示します。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

[756 ページの『属性 \(get および set\) メソッドのコーディング』](#)

[768 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

COBOL を定義するファクトリーメソッドの中に PROCEDURE DIVISION の FACTORY クラス定義の段落。ファクトリー・メソッドは、クラスのどのオブジェクト・インスタンスとは無関係に、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。

このタスクについて

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チャー指向の COBOL プログラムをラップすることもできます。ファクトリーメソッドを次のようにコーディングできます。main OO アプリケーションを実行できるようにするには、java コマンド、および標準に従ってアプリケーションを構造化しますジャワ練習する。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に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す。

表 93. 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)

表 93. ファクトリー・メソッド定義の構成 (続き)		
除算	目的	構文
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。

[777 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

[782 ページの『OO アプリケーションの構造化』](#)

[782 ページの『プロシージャ指向 COBOL プログラムのラッピング』](#)

[768 ページの『COBOL クラスのインスタンス化』](#)

[751 ページの『クラス・インスタンス・メソッドの定義』](#)

[756 ページの『属性 \(get および set\) メソッドのコーディング』](#)

[755 ページの『インスタンス・メソッドの多重定義』](#)

[776 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』](#)

[777 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』](#)

[573 ページの『IMS のもとでのオブジェクト指向 COBOL と Java の使用』](#)

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。

このタスクについて

スーパークラスのファクトリーメソッドを非表示にする f1COBOL サブクラスでファクトリーメソッドを定義する f1 同じ名前を持つサブクラスで、PROCEDURE DIVISION USING フレーズ (存在する場合) には、スーパークラスのメソッドと同じ数と型の仮パラメータがあります。(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。

[777 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

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[755 ページの『インスタンス・メソッドのオーバーライド』](#)

[763 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 \(Enterprise COBOL for z/OS 言語解説書\)](#)

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を呼び出すか、ジャワ COBOL メソッドまたはクライアントプログラムで静的メソッドを使用する場合は、クラス名を INVOKE 声明。

このタスクについて

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、あるいはカテゴリー英字、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。

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、重大度 3 の言語環境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。

[777 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

[763 ページの『メソッドの呼び出し \(INVOKE\)』](#)

[133 ページの『COBOL での国別データ \(Unicode\) の使用』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

関連参照

[INVOKE ステートメント \(Enterprise COBOL for z/OS 言語解説書\)](#)

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、必要なメソッドのシグニチャーに一致する引数を渡すことで、どちらの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、757 ページの『例: メソッドの定義』、769 ページの『例: クライアントの定義』、および 773 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 z/OS UNIX ファイルシステム の cobol/demo/oosample サブディレクトリーに入っています。一般的には、このソースへの絶対パスは /usr/lpp/cobol/demo/oosample 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 dll,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.
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 ZZZZZZ999999.
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 dll,thread,pgmname(longmixed)
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 dll,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

[767 ページの『クラスのインスタンスの作成および初期化』](#)

[775 ページの『ファクトリー・メソッドの定義』](#)

[777 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』](#)

[321 ページの『第 17 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』](#)

プロシーチャー指向 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チャー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、既存のプロシーチャー型 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。

このタスクについて

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。

手順

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チャー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。

タスクの結果

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 プロシーチャー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

[746 ページの『クラスの定義』](#)

[774 ページの『ファクトリー・セクションの定義』](#)

[775 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。

このタスクについて

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。

z/OS ユニックスのもとでは、リンクされたモジュールの名前(プログラム名と一致していなければなりません)をコマンド・プロンプトで指定することで、アプリケーションを実行できます。また、PDSE の

モジュールとしてプログラムをバインドし、EXEC PGM ステートメントを使用して JCL で実行することもできます。

- 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。
- アプリケーションを以下の環境で実行します。main の方法ジャワクラス（などジャワ従属地域）。
- 標準的な Java プログラミング方式に従う。

[783 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』](#)

関連タスク

[321 ページの『第 17 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』](#)

[775 ページの『ファクトリー・メソッドの定義』](#)

[791 ページの『Java 用の配列およびストリングの宣言』](#)

[569 ページの『第 25 章 COBOL プログラムの開発 \(IMS の場合\)』](#)

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

```
cb1 dll,thread
Identification Division.
Class-id. CBLmain inherits Base.
Environment Division.
Configuration section.
Repository.
    Class Base is "java.lang.Object"
    Class stringArray is "jobjectArray: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

```
cb1 dll,thread,pgmname(longmixed),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 Sbuffer        pic X(65535).  
    01 P              pointer.  
    Linkage section.  
    01 SA             object reference stringArray.  
    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 "GetStringPlatformLength"  
                using by value JNIEnvPtr  
                    SAElement  
                    address of SAElementlen  
                    0  
            Call "GetStringPlatform"  
                using by value JNIEnvPtr  
                    SAElement  
                    address of Sbuffer  
                    length of Sbuffer  
                    0  
            Display Sbuffer(1:SAElementlen)  
        End-perform  
    .  
    End method "main".  
End factory.  
End class Echo.
```

関連タスク

[321 ページの『第 17 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』](#)

[775 ページの『ファクトリー・メソッドの定義』](#)

[785 ページの『第 41 章 Java メソッドとの通信』](#)

第 41 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。

このタスクについて

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。

[797 ページの『例: COBOL で書かれた J2EE クライアント』](#)

[800 ページの『例: バッチ COBOL プログラムから Java を呼び出す』](#)

関連タスク

[321 ページの『第 17 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』](#)

[785 ページの『JNI サービスへのアクセス』](#)

[789 ページの『Java とのデータ共用』](#)

[743 ページの『第 40 章 オブジェクト指向プログラムの作成』](#)

[651 ページの『第 32 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

[JDK 5.0 Documentation](#)

JNI サービスへのアクセス

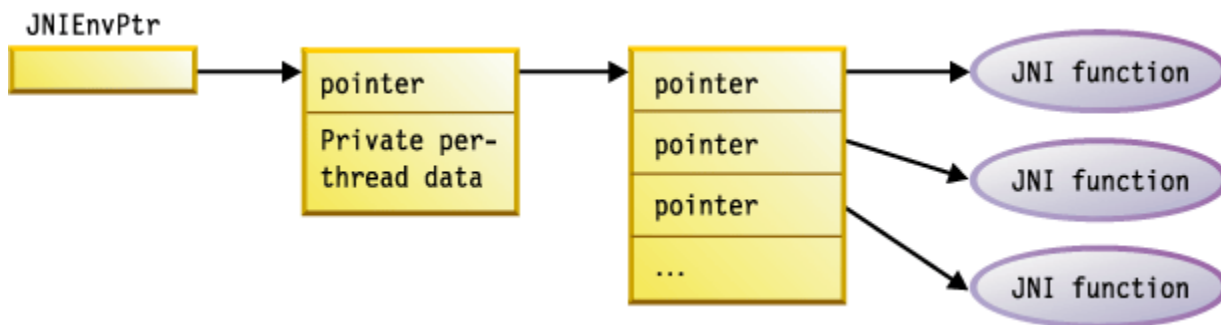
Java Native Interface (JNI) は、COBOL と Java を併用するアプリケーションを開発する際に使用できる多くの呼び出し可能サービスを提供します。このサービスへのアクセスを円滑に行うには、COBOL プログラムの LINKAGE SECTION に JNI.cpy をコピーします。

このタスクについて

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
- JNIInterface、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` は暗黙的に次のように定義される `USAGE POINTER`; 受信データ項目として使用しないでください。 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` は、`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` は、`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。

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`

制約事項: 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

[787 ページの『ローカル参照とグローバル参照の管理』](#)

[786 ページの『Java 例外の処理』](#)

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[759 ページの『クライアントの定義』](#)

関連参照

[906 ページの『JNI.cpy コピーブック』](#)

[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。

このタスクについて

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行いません。

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。

手順

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)。

タスクの結果

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。

[787 ページの『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。

このタスクについて

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の実行中にのみ有効です。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

[785 ページの『JNI サービスへのアクセス』](#)

[788 ページの『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のみ 保管します。

このタスクについて

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 94. 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	リターン値	目的
NewGlobalRef	• 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	• 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	• 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

785 ページの『JNI サービスへのアクセス』

Java アクセス制御

このタスクについて

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 とのデータ共有

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。)

このタスクについて

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

790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』

791 ページの『Java 用の配列およびストリングの宣言』

792 ページの『Java 配列の取り扱い』

795 ページの『Java ストリングの取り扱い』

763 ページの『メソッドの呼び出し (INVOKE)』

623 ページの『第 30 章 データの共用』

OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ

OO COBOL プログラムは、通信時に特定のデータ型のみを使用できます。ジャワ。

このタスクについて

注: このセクションでは、OO COBOL プログラムの COBOL/Java データ型の互換性に関する情報を提供する。非 OO COBOL/Java 相互運用性のための COBOL/Java データ型の互換性については、Enterprise COBOL Language Reference の「「非 OO COBOL と Java データ型のマッピング」を参照のこと。

表 95. 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。 <pre> level-number data-name PIC X. 88 data-name-false value X'00'. 88 data-name-true value X'01' through X'FF'. </pre>
byte ¹	半角英数字:PIC X または PIC A
short	USAGE BINARY、COMP、COMP-4、または COMP-5、と PICTURE フォームの条項 S9(n)、ただし 1<=n<=4
int	USAGE BINARY、COMP、COMP-4、または COMP-5、と PICTURE フォームの条項 S9(n)、ただし 5<=n<=9
long	USAGE BINARY、COMP、COMP-4、または COMP-5、と PICTURE フォームの条項 S9(n)、ただし 10<=n<=18
float ²	USAGE COMP-1
double ²	USAGE COMP-2
char	1 文字初歩的な全国: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 クラス名
1. ブール型とバイト型はそれぞれ次の値に対応しているので区別する必要があります。PIC X。PIC X 示されているように 2 つの条件名を使用して引数またはパラメーターを定義した場合にのみ、ブール値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。 2. Java の浮動小数点データは、IEEE Standard for Binary Floating Point Arithmetic に従ってフォーマットされます。Enterprise COBOL、ただし、16 進浮動小数点表現を使用します。浮動小数点引数を渡す場合、INVOKE ステートメント、または浮動小数点データを受け取るジャワメソッドでは、引数とデータは必要に応じて自動的に変換されます。	

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。次の表に示されている COBOL データ型をコーディングします。

このタスクについて

表 96. 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[]	object reference jbooleanArray
byte[]	object reference jbyteArray
short[]	object reference jshortArray
int[]	object reference jintArray
長さ[]	object reference jlongArray
文字[]	object reference jcharArray
Object[]	object reference jobjectArray
ストリング	object reference jstring

これらのクラスのいずれかを相互運用性のために使用するにはジャワ、エントリをコード化する必要があります REPOSITORY 段落。例:

```
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の分離文字として、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.  
... .
```

783 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 97. 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
浮く[]	object reference jfloatArray
ダブル[]	object reference jdoubleArray

関連タスク

748 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。指定する OCCURS または OCCURS DEPENDING ON 配列に適した句。

このタスクについて

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 98. JNI 配列サービス			
サービス	入力引数	リターン値	目的
GetArrayLength	- 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	- 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 98. <i>JNI</i> 配列サービス (続き)			
サービス	入力引数	リターン値	目的
GetBooleanArrayElements, GetByteArrayElements, GetCharArrayElements, GetShortArrayElements, GetIntArrayElements, GetLongArrayElements	• JNI 環境ポインタ • 配列オブジェクトの参照 • ブール項目へのポインタ。ポインタが 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 <code>ReleasexxxArrayElements</code> 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	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、ポインタがストレージ・バッファに戻される。ポインタは、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	• JNI 環境ポインタ • 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 • 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	• JNI 環境ポインタ • 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	• JNI 環境ポインタ • 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	• JNI 環境ポインタ • 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 98. **JNI 配列サービス** (続き)

サービス	入力引数	リターン値	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。			

[783 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』](#)

[794 ページの『例: Java 整数配列の処理』](#)

関連タスク

[790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』](#)

[791 ページの『Java 用の配列およびストリングの宣言』](#)

[785 ページの『JNI サービスへのアクセス』](#)

例: Java 整数配列の処理

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 整数配列の処理を示しています。

```

cbl thread,dll
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 JNIInterface 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 String・データを Unicode で表します。Java String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 英数字または国別 (Unicode) データを設定するか、オブジェクトから抽出します。

このタスクについて

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。これらのサービスは、ワークステーションとメインフレーム間で移植可能にするアプリケーションに使用します。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行います。

表 99. 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	リターン値
NewString ¹	- JNI 環境ポインター - COBOL 国別データ項目などの、Unicode String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	- 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	- 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	- 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 でない場合、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	- 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. システムがメモリ不足の場合、このサービスは例外を throw します。		

EBCDIC 向けサービス:以下を使用してください z/OSJNI の拡張であるサービスを使用して、jstring オブジェクト参照と COBOL 英数字データ (PIC X(n))。

表 100. jstring 参照と英数字データ間の変換サービス		
サービス	入力引数	リターン値
NewStringPlatform	- JNI 環境ポインター - jstring オブジェクトに変換するヌル終了 EBCDIC 文字ストリングへのポインター - 結果のための jstring オブジェクト参照へのポインター - ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング¹として表す	2 進数フルワード整数としての戻りコード: 0 正常。 -1 誤った形式の入力データまたは正しくない入力文字。 -2 非サポート・エンコード。jstring オブジェクト参照ポインターは NULL に設定されます。
GetStringPlatformLength	- JNI 環境ポインター - 長さのための jstring オブジェクト参照 - 結果のための 2 進数フルワード値へのポインター - ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング¹として表す	2 進数フルワード整数としての戻りコード: 0 正常。 -1 誤った形式の入力データまたは正しくない入力文字。 -2 非サポート・エンコード。jstring オブジェクト参照ポインターは NULL に設定されます。 2 番目の引数が参照するヌル終了バイトなど、変換した Java ストリングを保持するために必要な出力バッファの長さ (バイト単位) を、3 番目の引数に戻します。
GetStringPlatform	- JNI 環境ポインター - ヌル終了ストリングに変換する jstring オブジェクト参照 - 変換済みストリングのための出力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出力バッファの長さ - ストリングの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字ストリング¹として表す	2 進数フルワード整数としての戻りコード: 0 正常。 -1 誤った形式の入力データまたは正しくない入力文字。 -2 非サポート・エンコード。出力ストリングはヌル・ストリングに設定されます。 -3 変換バッファがいっぱいです。
1. ポインタが NULL、エンコードはジャワ file.encoding プロパティが使用されます。		

これらの EBCDIC サービスは、IBM Java ソフトウェア開発キットの一部である DLL としてパッケージされています。サービスの詳細については、IBM Java ソフトウェア開発キットの jni_convert.h を参照してください。

サービスを呼び出すには、CALL *literal* ステートメントを使用してください。呼び出しは、libjvm.x DLL サイド・ファイルを介して解決されます。このファイルは、オブジェクト指向言語を使用する COBOL プログラムのリンク手順に含める必要があります。

例えば、次のコードは、EBCDIC スtring 'MyConverter' から Java スtring・オブジェクトを作成します。(このコード・フラグメントは、797 ページの『例: COBOL で書かれた J2EE クライアント』で詳細に示す、J2EE クライアント・プログラムからのものです。)

```
Move z"MyConverter" to stringBuffer
Call "NewStringPlatform"
  using by value JNIEnvPtr
                address of stringBuffer
                address of jstring1
                0
returning rc
```

EBCDIC サービスが、COBOL プログラムから呼び出す唯一の JNI サービスである場合には、JNI.cpy コピーブックをコピーする必要はありません。また、JNI 環境 ポインターとのアドレス可能性を設定する必要はありません。

UTF-8 サービス Java ネイティブ・インターフェースでは、jstring オブジェクト参照と UTF-8 String との間の変換用のサービスも提供しています。これらのサービスは、COBOL プログラムでの使用にはお勧めしません。z/OS プラットフォームで UTF-8 文字 String を取り扱うことは難しいためです。

関連タスク

785 ページの『JNI サービスへのアクセス』

790 ページの『OO COBOL で相互運用可能なデータ型をコーディングし、ジャワ』

791 ページの『Java 用の配列および String の宣言』

133 ページの『COBOL での国別データ (Unicode) の使用』

321 ページの『第 17 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』

例: COBOL で書かれた J2EE クライアント

次の例では、J2EE 準拠 EJB サーバー上で実行する Enterprise Bean にアクセスできる COBOL クライアント・プログラムを示します。

COBOL クライアントは、「Java 2 Enterprise Edition 開発者ガイド」の『Getting Started』セクションにある J2EE クライアント・プログラムと同等です。インプリメンテーションを比較する上でのユーザーの便宜のために、2 番目の例では上記資料に記載されている同等の Java クライアントを示します。(エンタープライズ Bean は単純な通貨コンバーター エンタープライズ Bean の Java インプリメンテーションで、同資料に記載されています。)

Java Enterprise Bean の他のバージョンとクライアント・コードは、下記で参照されている「Java EE 5 チュートリアル」で検索できます。

COBOL クライアント (ConverterClient.cbl)

```
Process pgmname(longmixed),dll,thread
*****
* Demo J2EE client written in COBOL.                                     *
*                                                                 *
* Based on the sample J2EE client written in Java, which is         *
* given in the "Getting Started" chapter of "The Java(TM) 2        *
* Enterprise Edition Developer's Guide."                             *
*                                                                 *
* The client:                                                         *
*   - Locates the home interface of a session enterprise bean      *
*     (a simple currency converter bean)                             *
*   - Creates an enterprise bean instance                           *
*   - Invokes a business method (currency conversion)              *
*****
Identification division.
Program-id. "ConverterClient" is recursive.
Environment Division.
```

```

Configuration section.
Repository.
  Class InitialContext is "javax.naming.InitialContext"
  Class PortableRemoteObject
    is "javax.rmi.PortableRemoteObject"
  Class JavaObject is "java.lang.Object"
  Class JavaClass is "java.lang.Class"
  Class JavaException is "java.lang.Exception"
  Class jstring is "jstring"
  Class Converter is "Converter"
  Class ConverterHome is "ConverterHome".
Data division.
Working-storage section.
01 initialCtx          object reference InitialContext.
01 obj                 object reference JavaObject.
01 classObj            object reference JavaClass.
01 ex                  object reference JavaException.
01 currencyConverter   object reference Converter.
01 home                object reference ConverterHome.
01 homeObj             redefines home object reference JavaObject.
01 jstring1            object reference jstring.
01 stringBuffer        pic X(500) usage display.
01 len                 pic S9(9) comp-5.
01 rc                  pic S9(9) comp-5.
01 amount              comp-2.
Linkage section.
  Copy JNI.
Procedure division.
  Set address of JNIEnv to JNIEnvPtr
  Set address of JNINativeInterface to JNIEnv

*****
* Create JNDI naming context.
*****
  Invoke InitialContext New returning initialCtx
  Perform JavaExceptionCheck

*****
* Create a jstring object for the string "MyConverter" for use
* as argument to the lookup method.
*****
  Move z"MyConverter" to stringBuffer
  Call "NewStringPlatform"
    using by value JNIEnvPtr
    address of stringBuffer
    address of jstring1
    0
  returning rc
  If rc not = zero then
    Display "Error occurred creating jstring object"
    Stop run
  End-if

*****
* Use the lookup method to obtain a reference to the home
* object bound to the name "MyConverter". (This is the JNDI
* name specified when deploying the J2EE application.)
*****
  Invoke initialCtx "lookup" using by value jstring1
  returning obj
  Perform JavaExceptionCheck

*****
* Narrow the home object to be of type ConverterHome.
* First obtain class object for the ConverterHome class, by
* passing the null-terminated ASCII string "ConverterHome" to
* the FindClass API. Then use this class object as the
* argument to the static method "narrow".
*****
  Move z"ConverterHome" to stringBuffer
  Call "_etoa"
    using by value address of stringBuffer
    returning len
  If len = -1 then
    Display "Error occurred on ASCII conversion"
    Stop run
  End-if
  Call FindClass
    using by value JNIEnvPtr
    address of stringBuffer
    returning classObj
  If classObj = null

```

```

        Display "Error occurred locating ConverterHome class"
        Stop run
    End-if
    Invoke PortableRemoteObject "narrow"
        using by value obj
            classObj
        returning homeObject
    Perform JavaExceptionCheck

*****
* Create the ConverterEJB instance and obtain local object      *
* reference for its remote interface                            *
*****
    Invoke home "create" returning currencyConverter
    Perform JavaExceptionCheck

*****
* Invoke business methods                                      *
*****
    Invoke currencyConverter "dollarToYen"
        using by value +100.00E+0
        returning amount
    Perform JavaExceptionCheck

    Display amount

    Invoke currencyConverter "yenToEuro"
        using by value +100.00E+0
        returning amount
    Perform JavaExceptionCheck

    Display amount

*****
* Remove the object and return.                                  *
*****
    Invoke currencyConverter "remove"
    Perform JavaExceptionCheck

    Goback
    .

*****
* Check for thrown Java exceptions                              *
*****
    JavaExceptionCheck.
    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
    .
End program "ConverterClient".

```

Java クライアント (ConverterClient.java)

```

/*
 *
 * Copyright 2000 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the proprietary information of Sun Microsystems, Inc.
 * Use is subject to license terms.
 *
 */

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import Converter;
import ConverterHome;

public class ConverterClient {

    public static void main(String[] args) {

```



```

try {
    Context initial = new InitialContext();
    Object objref = initial.lookup("MyConverter");

    ConverterHome home =
        (ConverterHome)PortableRemoteObject.narrow(objref,
            ConverterHome.class);

    Converter currencyConverter = home.create();

    double amount = currencyConverter.dollarToYen(100.00);
    System.out.println(String.valueOf(amount));
    amount = currencyConverter.yenToEuro(100.00);
    System.out.println(String.valueOf(amount));

    currencyConverter.remove();

} catch (Exception ex) {
    System.err.println("Caught an unexpected exception!");
    ex.printStackTrace();
}
}
}

```

関連タスク

321 ページの『[第 17 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](#)』

[WebSphere for z/OS: アプリケーション](#)

[Java 2 Enterprise Edition Developer's Guide \(Getting Started\)](#)

[The Java EE 5 Tutorial \(Getting Started with Enterprise Beans\)](#)

例: バッチ COBOL プログラムから Java を呼び出す

Java Batch Launcher and Toolkit for z/OS (JZOS) を使用して、バッチ COBOL プログラムから Java を呼び出すことができます。次の例には、バッチ・ジョブ・ステップで Java プログラムを呼び出す COBOL プログラムの JCL とソースが含まれています。括弧内の番号は、後続の注釈に対応しています。

この環境では、多くの場合、標準の `JavaSystem.out` ファイルおよび `System.err` ファイルを z/OS データ・セットまたはスプール・ファイルに送信することが望ましいと考えられます。これを行うには、Java 仮想マシン (JVM) の始動後に `com.ibm.jzos.ZUtil` クラスの `redirectStandardStreams` メソッドを呼び出します。ZUtil クラスについて詳しくは、*z/OS security and legacy services API Reference* の [ZUtil](#) を参照してください。

この COBOL プログラム例では、`com.ibm.jzos.sample.HelloWorld` クラスから `main()` メソッドが呼び出されますが、これを変更して他の Java クラス・メソッドを呼び出すことができます。

```

//COB2JAV JOB (),'Dovetail',
// MSGCLASS=H, REGION=128M,
// NOTIFY=&SYSUID
//*
//* Tested on z/OS 2.2 with Ent Cobol 5.1 and Java 7.0
// SET COBPRFX='SYSPROG.MNT.COBOL51' (1)
// SET LIBPRFX='CEE'
// SET SYSLIB1=&COBPRFX..SIGYLIB,DISP=SHR (2)
//* See also CLASSPATH below
//*
//COMPILE EXEC PGM=IGYCRCTL,
// PARM='SIZE(5000K)'
//SYSLIB DD DISP=SHR,DSN=&SYSLIB1 (JNI) CPY
//SYSLIN DD DSN=&&OBJECT(TSTHELLO),UNIT=3390,DISP=(NEW,PASS),
// SPACE=(CYL,(1,1,1)),DCB=(LRECL=80,RECFM=FB)
//SYSPRINT DD SYSOUT=*
//STEPLIB DD DSN=&COBPRFX..SIGYCOMP,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSUT1 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT8 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))

```

```

//SYSUT9 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT10 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT11 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT12 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT13 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT14 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT15 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSMDECK DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSIN DD *
    cbl dll,thread
    Identification division.
    Program-id. "TSTHELLO" recursive.
    Environment division.
    Configuration section.
    Repository.
        Class ZUtil          is "com.ibm.jzos.ZUtil"          (3)
        Class HelloWorld     is "com.ibm.jzos.sample.HelloWorld" (4)
        Class JavaException  is "java.lang.Exception"
        Class JavaObject     is "java.lang.Object"
        Class JavaString     is "java.lang.String"
        Class JavaClass      is "java.lang.Class"
        Class stringArray    is "jobjectArray:java.lang.String".

    Data Division.
    Working-storage section.
    01 args          object reference stringArray.
    01 argsLen       pic s9(9) binary value 0.
    01 jstring1      object reference JavaString.
    01 stringClass   object reference JavaClass.
    01 ex            object reference JavaException.
    01 stringBuffer pic X(256) usage display.
    Linkage section.
    COPY "JNI" SUPPRESS.

    Procedure division.
        Display "COBOL program TSTHELLO entered"
        Set address of JNIEnv to JNIEnvPtr
        Set address of JNINativeInterface to JNIENV

    *
    * This static JZOS method will redirect Java stdout/stderr
    * to DD:STDOUT and DD:STDERR, which may be spool files or data sets
    *
        Invoke ZUtil "redirectStandardStreams"          (3)
        Perform ErrorCheck
        Display "Returned from ZUtil.redirectStandardStreams"

    *
    * We invoke com.ibm.jzos.sample.HelloWorld,
    * but this could be any arbitrary Java code
    *
        Perform BuildEmptyArgsArray.
        Invoke HelloWorld "main"          (4)
            using by value args
        Perform ErrorCheck
        Display "Returned from HelloWorld.main"

    Goback.

    ErrorCheck.
        Call ExceptionOccurred
            using by value JNIEnvPtr
            returning ex
        If ex not = null then
            Call ExceptionClear using by value JNIEnvPtr
            Display "Caught a Java exception"
            Invoke ex "printStackTrace"
            Stop run
        End-if.

    BuildEmptyArgsArray.
    * Create a new empty string
        Call NewString
            using by value JNIEnvPtr
            address of stringBuffer
            0
            returning jstring1
        If jstring1 not = null then
            Display "NewString returned OK"
        Else
            Display "NewString returned null!"
            Stop run
        End-if

```

```

* Get a reference to the String class object
  Call GetObjectClass
    using by value JNIEnvPtr jstring1
    returning stringClass
  If stringClass not = null then
    Display "GetObjectClass returned OK"
  Else
    Display "GetObjectClass returned null!"
    Stop run
  End-if

* Create a zero-length String[] array
  move 0 to argsLen
  Call NewObjectArray
    using by value JNIEnvPtr
    argsLen stringClass jstring1
    returning args
  If args not = null then
    Display "NewObjectArray returned OK"
  Else
    Display "NewObjectArray returned null!"
    Stop run
  End-if.

End program "TSTHELLO".

/*
//LKED EXEC PGM=IEWL,COND=(4,LT,COMPILE),
//      PARM='RENT,LIST,LET,DYNAM(DLL),CASE(MIXED)'
//SYSLIB DD DSN=&LIBPRFX..SCEELKED,DISP=SHR
// DD DSN=&LIBPRFX..SCEELKEX,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//SYSLMOD DD DSN=&&GOSET(TSTHELLO),DISP=(MOD,PASS),UNIT=3390,
// SPACE=(CYL,(1,1,1)),DSNTYPE=LIBRARY
//SYSDEFSD DD DUMMY
//OBJMOD DD DSN=&&OBJECT,DISP=(OLD,DELETE)
//SYSLIN DD *
  INCLUDE OBJMOD(TSTHELLO)
  INCLUDE '/usr/lpp/java/J7.0/bin/j9vm/libjvm.x' (5)
  INCLUDE '/usr/lpp/cobol/V5R1/lib/igzjava.x' (5)
/*
/* Note: we expect RC=32 since we should Stop run for exception
/*
//GO EXEC PGM=TSTHELLO,COND=(4,LT,LKED)
//CEEOPDS DD *
* Be careful when editing: quoted ENVARS wrap at col 72
ENVAR( (6)
"PATH=bin:/usr/lpp/java/J7.0/bin",
"LIBPATH=lib:/usr/lib:/usr/lpp/java/J7.0/bin:/usr/lpp/java/J7.0/lib/s390
:/usr/lpp/java/J7.0/lib/s390/j9vm",
"CLASSPATH=/home/g1java1/jzostest/jzos_test.jar")
POSIX(ON) XPLINK(ON)
*
* Add this ENVAR to send stdout/stderr to DD:SYSOUT
* "COBJVMINITOPTIONS=-Djzos.merge.sysout=true", (7)
* Debugging options:
* "COBJVMINITOPTIONS=-Xdump:ceedump -Xcheck:jni:trace -Xjit:verbose")
//STEPLIB DD DSN=*.LKED.SYSLMOD,DISP=(OLD,PASS)
// DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
// DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//SYSOUT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD DUMMY
/*
/* ZUtil.redirectStandardStreams will point to these for sdtout/stderr
/* Unless you add the -Djzos.merge.sysout=true option above.
/* Using that option, both Java stdout/stderr with go to DD:SYSOUT
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
/*
/* JAVAOUT/JAVAERR should not be used unless redirectStandardStreams fails
/* so you may choose to point these to DUMMY
//JAVAOUT DD PATH='/tmp/cob2jav.javaout', (8)
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR,SIRGRP)
//JAVAERR DD PATH='/tmp/cob2jav.javaerr', (8)
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR,SIRGRP)

```

- (1) JCL シンボルを環境に合わせて設定します。
- (2) JNI.cpy ファイルはコンパイラーに付属しており、データ・セット SIGYLIB のメンバー JNI として使用できます。
- (3) ZUtilredirectStandardStreams メソッドは、Java System.out と System.err をそれぞれ DD:STDOUT と DD:STDERR にリダイレクトします。
- (4) com.ibm.jzos.sample.HelloWorld クラスは「Hello World!」と印刷します (stdout) System.out に、そして「Hello World! (stderr) " System.err に。 [JZOS サンプル](#)を使用して com.ibm.jzos.sample.HelloWorld クラスをダウンロードできます。
- (5) Java および COBOL がインストールされている場所を指すように INCLUDE を設定します。
- (6) Java ホーム・ディレクトリーを指すように言語環境プログラム ENVAR を設定します。 個々の環境変数設定は 72 桁で折り返されることに注意してください。
- (7) JVM の Java システム・プロパティーを設定するために、示されているように *COBJVMINITOPTIONS* 環境変数を追加できます。 jzos.merge.sysout=true プロパティーを使用して、System.out と System.err の両方をマージし、DD:SYSOUT に移動することができます。
- (8) ZUtil.redirectStandardStreams() が正しく機能する場合、JAVAOUT DD ステートメントと JAVAERR DD ステートメントは使用されないため、これらのステートメントが DD DUMMY を指すようにすることができます。

JZOS について

Java Batch Launcher and Toolkit for z/OS (JZOS) は、従来のバッチ環境で実行され、z/OS システム・サービスにアクセスする z/OS Java アプリケーションの開発を支援するツールのセットでう。詳しくは、[JZOS インストールおよびユーザーズ・ガイド](#)を参照してください。

第7部 オブジェクト指向 (OO) COBOL フレームワーク の外部での COBOL/Java 相互運用性

Enterprise COBOL は、オブジェクト指向 (OO) COBOL フレームワークによって提供されるものよりも包括的で柔軟な形式の Java 相互相互運用性をサポートします。

スタンドアロン COBOL/Java 相互相互運用性の機能の主な目的は、以下のとおりです。

- Java アプリケーションが既存の COBOL プログラムを呼び出すための簡単な方法を提供します。その際、COBOL プログラムを大幅に変更する必要はなく、新しい構築要件を大幅に適用する必要もありません。代わりに、ユーザー プログラムは動的に呼び出され、通常のユーザー プログラム オブジェクト ライブラリ内に引き続き存在することができます。
- 標準の CALL ステートメントを使用して、静的 Java メソッドを簡単に呼び出すことができるように、既存の COBOL プログラムを簡単に変更する方法を提供します。
- 一般的なさまざまな Java データ・タイプとそれに対応する COBOL データ・タイプとの間の自動変換を包括的にサポートすることにより、パラメーターの引き渡しと戻り値の処理にユーザー・コードでの特別な処理を必要としないようにします。これにより、COBOL プログラマーが Java Native Interface (JNI) プログラミングについて知る必要性が大幅に削減されます。これに対して、OO COBOL フレームワークでは、Java からの着信 String オブジェクトを処理する場合でも、Java スtring・データを PIC X、PIC N、または PIC U 節で定義されたデータ項目に移動できるようにするために、JNI を認識して、場合によっては変換する必要があります。

Java 型に自動的にマップされる COBOL データ型は、Java 互換の COBOL データ型と呼ばれます。詳細は、*Enterprise COBOL for z/OS* 言語解説書の COBOL 型と Java 型の間のマッピングを参照してください。

- 読み取りと書き込みの両方の目的で、Java アプリケーションが COBOL WORKING-STORAGE メモリーに直接簡単にアクセスできる方法を示します。
- AMODE 31 COBOL プログラムと 64 ビット Java アプリケーションとの相互運用を容易にします。

これらの目標を達成するために、Enterprise COBOL のスタンドアロン Java 相互運用性の機能は、以下のエレメントで構成されています。

- 既存の COBOL プログラムを Java によって自動的に呼び出し可能にする JAVA-CALLABLE ディレクティブ。
- Java アプリケーションが読み取りと書き込みの両方でアクセスできるようにする 1 つ以上の 01 /77 レベル WORKING-STORAGE データ項目を識別する JAVA-SHAREABLE ディレクティブ。
- 形式 '*Java.java-class-name.java-static-method-name*' のリテラル CALL ターゲットを静的 Java メソッドの呼び出しとして解釈する、拡張された CALL ステートメント。

関連参照

例: COBPROD アプリケーション

JAVA 呼び出し可能 (*Enterprise COBOL for z/OS* 言語解説書)

JAVA 共有可能 (*Enterprise COBOL for z/OS* 言語解説書)

第 42 章 既存のプログラムを Java から呼び出し可能にする

JAVA-CALLABLE ディレクティブは、既存の COBOL プログラムを Java から呼び出し可能にするために提供されています。

0 個以上の Java 互換パラメータと、オプションで Java 互換でなければならない戻り値から構成される、よく定義されたインタフェースを持つプログラムであれば、JAVA-CALLABLE ディレクティブを使用して、既存の COBOL プログラムを Java から呼び出し可能にすることができます。

JAVA-CALLABLE ディレクティブを、既存プログラムの PROCEDURE DIVISION ヘッダーの前の任意の場所に追加する必要があります。プログラムがコンパイルされると、コンパイラーは、z/OS UNIX ディレクトリーに書き込まれる「COBOL ネイティブ・メソッド・スタブ」プログラムと呼ばれるファイルを生成します。コンパイラー・オプション JAVAIOP の OUTPATH サブオプションを使用して、ディレクトリーの場所を指定できます。詳しくは、[412 ページの『JAVAIOP』](#)を参照してください。

COBOL ネイティブ・メソッド・スタブ・ファイルの主な目的は、以下の通りです。

- 着信 Java 引数を、それらを受け取る COBOL パラメーターの適切なフォーマットに変換します。
- 既存の COBOL プログラムを動的に呼び出します。このプログラムは、プログラムオブジェクトライブラリプログラム実行時に STEPLIB に含まれる
- Java に戻る前に、出力戻り値を COBOL データ・フォーマットから適切な Java フォーマットに変換します。

COBOL プログラムに関連付けられているスタブ・ファイルが、他にもあります。詳細については、*Enterprise COBOL for z/OS* 言語解説書の [を参照してください](#)。

COBOL/Java 相互運用可能アプリケーションには、多くの Java 呼び出し可能 COBOL プログラムが関与することがあります。アプリケーションに必要なすべてのスタブ・ファイルは、z/OS UNIX ファイル・システムの COBOL インストール・ディレクトリーの bin サブ・ディレクトリーに存在する cjbuild ユーティリティを使用して、アプリケーション用の DLL にビルドする必要があります。結果の DLL は、Java から COBOL への呼び出しが行われる前、または Java が COBOL の WORKING-STORAGE メモリーにアクセスする際に、実行時に Java 仮想マシン (JVM) にロードされます。さらに、COBOL プログラムが、CALL ステートメントを使用して、静的 Java メソッドを呼び出すたびにアクセスされます。詳しくは、[331 ページの『相互運用可能な非 OO アプリケーションの構築ジャワ』](#)を参照してください。

例

```
identification division.
program-id. COBPROG.
data division.
working-storage section.
linkage-section.
01 arg1 pic s9(9) comp-5.
01 arg2.
   03 pic s9(18) comp-5 occurs 10 times.
01 retval pic x(20).
>>JAVA-CALLABLE
procedure division using arg1 arg2 arg3
                        returning retval
* COBOL code follows
:
```

この例では、JAVA-CALLABLE ディレクティブにより、このプログラムは Java によって自動的に呼び出し可能になります。

注:

- JAVA-CALLABLE ディレクティブを使用する COBOL プログラムに、Java 互換タイプの 1 つではないパラメーターがある場合、エラーが生成されます。

- 一般的な COBOL グループ項目がパラメータとして指定され、合法的な Java 互換の配列型でない場合は、Java のバイト配列(byte[])にマップされる。この場合、それらのバイトをマップするために使用される COBOL グループのレイアウトに一致するデータでバイト配列を埋める必要がある。

詳細については、*Enterprise COBOL for z/OS* 言語解説書 のを参照してください。

COBPROG プログラムは通常どおりにコンパイルする必要がありますが、JAVAIOP(OUTPATH(zos-unix-directory)) オプションを指定して、アプリケーション用に生成される「スタブ」プログラムが生成される z/OS UNIX ファイル・システム上の出力ディレクトリを指定します。

cjbuild ユーティリティを実行して、このアプリケーションの DLL を構築する場合、Java アプリケーションで COBOL ネイティブ・メソッドが関連付けられる Java パッケージを示すパッケージ名 (-p オプション) を指定することができます。デフォルトは enterprise.COBOL です。

COBPROG と呼ばれる 1 つの COBOL ネイティブ・メソッドだけで構成され、コンパイラによって生成されたスタブ・ファイルが、他のアプリケーションのスタブ・ファイルを含まないカレント・ディレクトリにあり、cjbuild のすべての出力パスをカレント・ディレクトリとするアプリケーションの場合（これはデフォルトであり、ここでは簡単のためにそうしているが、実運用では推奨されない）、cjbuild は以下のように呼び出すことができます：

```
cjbuild -v app1
```

これは、cjbuild に、現在のディレクトリ内のすべてのスタブ ファイルをビルドする DLL に含めるように指示します。文字列「app1」は、結果として得られる DLL の名前を構築するために使用されます。DLL が書き込まれる場合 z/OS ユニックスパスを指定すると、DLL が呼び出されます libapp1.so。DLL が MVS データセットに出力される場合、DLL はそのデータセット内で名前を持つメンバーとして表示されます。LIBAPP1。を参照してください-d 詳細については、cjbuild ユーティリティのオプションを参照してください。

あるいは、複数の相互運用可能なアプリケーションのスタブファイルを同じディレクトリに混在させている場合は、現在のディレクトリにファイルを作成することができます。z/OS ユニックス COBPROG という名前を含む「programs」というディレクトリを作成します。これは、現在のアプリケーションの一部である唯一のプログラムです。その後、次のように cjbuild を呼び出すことができます。

```
cjbuild -v -i COBPROG app1
```

の -i/--progid プログラム ファイルにアプリケーション内の各 COBOL プログラムの名前を含めることができるのと同様に、オプションはアプリケーション内の各 COBOL プログラムに対して 1 回指定できます。

プログラミングのヒント: プログラムファイルを使用するか、1 つ以上の -i/--progid オプションがサポートされているので、すべての COBOL プログラムのスタブファイルを 1 つの z/OS ユニックスディレクトリでは、構築している特定の相互運用可能なアプリケーションに対して、どのサブセットを重視するかを指定できます。ただし、相互運用可能なアプリケーションごとに 1 つのスタブファイルのディレクトリを使用する場合は、コードを整理するための 1 つのアプローチとして、これらのオプションと引数を省略するだけで、cjbuild はアーティファクトディレクトリ内のすべてのスタブファイルを次のように処理します。-c/--coboldir オプションを使用すると、ビルド プロセスが簡素化されるため、推奨されます。

注: DLL が z/OS UNIX ファイル・システムにある場合は、アプリケーションを開始する Java コマンドで JVM の java.library.path プロパティを使用して、DLL の場所を示すことができます。

例えば、クラス TestApp の「main」メソッドで始まり、/home/myid/libs にある関連スタブ・プログラム DLL を使用して COBOL プログラムを呼び出す Java アプリケーションを実行するには、次のコマンドを発行します。

```
java -Djava.library.path="/home/myid/libs" TestApp
```

そうでなければ、DLL がデータ・セットに出力される場合、実行時にデータ・セットが STEPLIB にあることを確認してください。

このネイティブ・メソッドを呼び出すための事実上の Java コードは、以下のとおりです。

```
import enterprise.COBOL.*;

:
int arg1 = 5;
int[] arg2 = new int[10];
:
String retval = enterprise.COBOL.progs.COBPROG(arg1, arg2);
```

注: COBOL ネイティブ・メソッド・プログラムの名前がリテラルを使用して定義されていない場合、つまり名前に大/小文字の区別がない場合は、上記の COBPROG の例のように、Java でその名前を大文字の形式で参照する必要があります。ただし、COBPROG プログラムが PGMNAME (LONGMIXED) オプションを有効にしてコンパイルされている場合は、プログラム名を「cobprog」として指定する必要があります。また、Java では、引用符で囲まれているとおりの名前でもプログラムを参照する必要があります。次の例を参照してください。

```
String retval = enterprise.COBOL.progs.cobprog(arg1, arg2);
```

関連参照

例: COBPROD アプリケーション

JAVA 呼び出し可能 (Enterprise COBOL for z/OS 言語解説書)

第 43 章 COBOL からの静的 Java メソッドの呼び出し

Java アプリケーションが COBOL に対して呼び出しを行い、Java と COBOL のデータ形式の間で引数を自動的に変換する場合と同じ方法で、COBOL は Java に対して呼び出しを行い、同じ自動変換を行うことができます。静的 Java メソッドに対する呼び出しのみがサポートされます。

CALL ステートメントを使用して Java を呼び出すことができます。呼び出しターゲットは次の形式のリテラルでなければなりません。

```
'Java.java-class-name.java-static-method-name'
```

ここで、`java-class-name` は、(パッケージの一部である場合) パッケージ名を含む完全修飾 Java クラス名です。また、`java-static-method-name` は、呼び出されている静的メソッドの名前です。

メソッドが、ネストされたクラスのグループに含まれている場合、ネストされた各クラス名は、リテラル内に \$ 文字で導入する必要があります。

例えば、パッケージ `com.acme` の一部として外部クラス `TestApp` 内のネストされたクラス `Util` に存在するメソッド `myMethod()` を呼び出すには、次の CALL ステートメントを使用します。

```
CALL 'Java.com.acme.TestApp$Util.myMethod' USING ...
```

注：リテラルの「Java.」接頭部には大/小文字の区別はありません。つまり、「JAVA.」および「jAvA.」は両方とも受け入れられますが、リテラルの残りの部分は大/小文字の区別があり、Java で宣言されている名前と一致している必要があります。

例

以下の例は、示されているシグニチャーを持つ静的メソッド `MyMethod` を使用する、Java クラス `TestApp` で構成されています。

```
identification division.  
program-id. COBPROG.  
data division.  
:  
package com.acme;  
import java.math.*;  
  
public class TestApp  
{  
    public static void myMethod(float f1, long l1, BigDecimal[] bdarr) {  
        :  
    }  
}
```

このメソッドは、以下のようにして COBOL から呼び出すことができます。

```
working-storage section.  
01 f1 comp-1.  
01 l1 pic s9(18) comp-5.  
01 bdarr.  
    03 pd pic s9(9)v9(2) comp-3 occurs 3 times.  
Procedure division.  
:  
CALL 'Java.com.acme.TestApp.myMethod' USING f1 l1 bdarr
```

この COBOL アプリケーションの DLL を構築するには、ジャワコンパイラによって生成されたスタブファイルが、他のアプリケーションのスタブファイルを含まない現在のディレクトリにあり、`cjbuild` のすべての出力パスに現在のディレクトリを受け入れる場合 (これはデフォルトで、ここでは簡潔にするためにこれを行っています)、`cjbuild` は次のように呼び出すことができます。

```
cjbuild -v app1
```

これは、cjbuild に、現在のディレクトリ内のすべてのスタブ ファイルをビルドする DLL に含めるように指示します。文字列「app1」は、結果として得られる DLL の名前を構築するために使用されます。DLL が書き込まれる場合 z/OS ユニックスパスを指定すると、DLL が呼び出されます libapp1.so。DLL が MVS データセットに出力される場合、DLL はそのデータセット内で名前を持つメンバーとして表示されます。LIBAPP1。を参照してください-d 詳細については、cjbuild ユーティリティのオプションを参照してください。

あるいは、複数の相互運用可能なアプリケーションのスタブファイルを同じディレクトリに混在させている場合は、現在のディレクトリにファイルを作成することができます。z/OS ユニックス COBPROG という名前を含む「programs」というディレクトリを作成します。これは、現在のアプリケーションの一部である唯一のプログラムです。その後、次のように cjbuild を呼び出すことができます。

```
cjbuild -v programs app1
```

さらに別の方法として、複数の相互運用可能なアプリケーションのスタブファイルを同じディレクトリに混在させている場合は、-i/--progid cjbuild のオプションを使用して、COBPROG が現在のアプリケーション内の唯一の COBOL プログラムであることを示すと、次のように cjbuild を呼び出すことができます。

```
cjbuild -v -i COBPROG app1
```

プログラミングのヒント: プログラムファイルを使用するか、1 つ以上の -i/--progid オプションがサポートされているので、すべての COBOL プログラムのスタブファイルを 1 つの z/OS ユニックスディレクトリでは、構築している特定の相互運用可能なアプリケーションに対して、どのサブセットを重視するかを指定できます。ただし、相互運用可能なアプリケーションごとに 1 つのスタブファイルのディレクトリを使用する場合は、コードを整理するための 1 つのアプローチとして、これらのオプションと引数を省略するだけで、cjbuild はアーティファクトディレクトリ内のすべてのスタブファイルを次のように処理します。-c/--coboldir オプションを使用すると、ビルド プロセスが簡素化されるため、推奨されます。

関連参照

例: COBPROD アプリケーション

第 44 章 COBOL での Java 互換配列型の使用

COBOL 表の定義が明確に定義されたパターンに従っている場合、COBOL 表は Java の一次元配列と互換性があります。

特に、COBOL 表が Java 配列と互換性があると見なされるためには、以下の方法で表を定義する必要があります。

```
01 array.  
  03 array-data <java-compatible-elementary-type> occurs X times.
```

どこ `java-compatible-elementary-type` のアイテムの 1 つですテーブル: ジャワ互換の基本 COBOL 型の中に *Enterprise COBOL for z/OS* 言語解説書。

注: 配列名と配列データ名はプレースホルダーであり、正当な COBOL データ名はすべて受け入れられます。

例えば、Java プリミティブ型 `double (double[])` の値の配列に対応する COBOL 表は、以下のように定義されます。

```
01 array-double.  
  03 double-data comp-2 occurs 10 times.
```

`java.math.BigDecimal` オブジェクトに対応する COBOL 表の例を以下に示します。

```
01 array-packed.  
  03 packed-data pic s9(5)v9(4) usage comp-3 occurs 20 times.
```

注: USAGE COMP-3 または USAGE PACKED-DECIMAL の代わりに USAGE DISPLAY を使用することもできます。つまり、COBOL のパック 10 進数型とゾーン 10 進数型の両方が Java の `java.math.BigDecimal` にマップされます。正しいパック 10 進数またはゾーン 10 進数の定義は、`java.math.BigDecimal` と互換性があります。ネイティブ COBOL パックまたはゾーン 10 進数形式と `java.math.BigDecimal` オブジェクトとの間の変換はすべて、IBM JZOS Web Development Toolkit 製品の一部である変換ルーチンによって処理されます。

`java.lang.String` オブジェクトに対応する配列の例を以下に示します。

```
01 array-string  
  03 string-data pic n(20) occurs 15 times.
```

注: スtring配列の場合は、PIC X、PIC N、または PIC U のいずれも使用できます。しかし、ジャワ文字列は UTF-16 Unicode エンコーディング、PIC N 文字列データを通信する最も効率的な方法となるジャワコードページ変換が不要なため、COBOL でも使用できます。

バイト配列および汎用グループ・パラメーター

COBOL コンパイラーは、Java 互換配列型の定義と一致する可能性があるグループ・パラメーターを、Java-callable プログラムで検索し、グループ引数を Java の呼び出しで検索します。

Java 互換配列タイプの定義を満たしていないグループの場合、COBOL コンパイラーは、Java (`byte[]`) でそのグループをバイトのフラット配列として扱い、Java コードがバイト配列からデータを抽出するか、または COBOL でのデータのレイアウトと一致する方法でバイト配列にデータを書き込むと想定します。Java 互換配列タイプの詳細については、*Enterprise COBOL for z/OS* 言語解説書の「表: Java 互換配列の COBOL タイプ」を参照してください。

注: Java でバイトの配列に対応するものとして扱われるグループ・パラメーターまたは引数については、コンパイラーによって通知メッセージ IGYPA3415I が生成されます。

例えば、以下の COBOL グループについて考えてみます。

```
01 g1.  
03 g1-data1 pic s9(9) comp-5 value 12.  
03 g1-data2 pic s9(18) comp-5 value 15.
```

このグループは、コンパイラによって、Java 内のバイトの配列に対応するものとして識別されます。g1 が JAVA-CALLABLE ディレクティブを使用して COBOL プログラムのパラメーターとして使用される場合、Java からの着信データはバイト配列 (byte[]) であると想定され、配列を正しくセットアップするのは Java プログラムの責任です。

一方、COBOL プログラムが CALL の引数として g1 を静的 Java メソッドに渡す場合、対応する Java メソッドは、グループをバイトの配列として受け取る必要があります。

COBOL から呼び出すことができ、グループ引数 g1 をバイト配列として受け取り、g1 の 2 つの従属データ項目にアクセスできる静的 Java メソッドの例を以下に示します。

```
import java.nio.*;  
  
public class TestApp  
{  
    public static void myMethod(byte[] g1)  
    {  
        ByteBuffer bb = ByteBuffer.wrap(g1, 0, 12);  
        System.out.println("data1 = " + bb.getInt(0));  
        System.out.println("data2 = " + bb.getLong(4));  
    }  
}
```

Java の相互運用可能アプリケーションにおける配列更新のセマンティクスと制限

Java では、配列は、Java プリミティブ型 (short[], int[], long[] など) の配列であっても、常にオブジェクトと見なされます。そのため、Java メソッド間で配列が渡されると、呼び出されたメソッドによって実行された変更は、常に呼び出し元メソッドに反映されます。これは、すべてのオブジェクト引数に対する Java の通常動作です。

この動作は Enterprise COBOL でも保持されます。ジャワ相互運用性モデルプリミティブ型の配列の場合。例えば、Java メソッドが Java プリミティブ型の配列を COBOL に渡し、COBOL がデータを受け取る COBOL テーブルを更新すると、その変更は常に Java 呼び出し元に反映される。同様に、COBOL が Java を呼び出し、呼び出された Java メソッドで受け取ったデータの表を配列オブジェクトとして渡すと、Java で行われた配列データに対する変更は、Java メソッドが戻るときに COBOL 表に反映されます。

注：パフォーマンス上の理由から、COBOL が Java から java.lang.String オブジェクトの配列または java.math.BigDecimal オブジェクトの配列を受信すると、COBOL でそのデータに変更が加えられても、アプリケーションの Java 側には反映されない。同様に、COBOL が PIC X/N/U 項目の表を Java に渡すとき、Java でそのデータを変更しても、アプリケーションの COBOL 側には反映されない。

配列の制限

現時点では、COBOL の固定長の表のみが Java 配列 (ODO 表なし) を受け取ることができ、着信 Java 配列の要素の数と、OCCURS 節で示されている COBOL 表の要素の数が正確に一致している必要があります。そうでない場合は、java.lang.IllegalArgumentException クラスの Java 例外がスローされる。この例外は、Java コードでは try/catch で捕捉でき、COBOL プログラムでは CALL 文の ON EXCEPTION 句を使用して静的 Java メソッドを呼び出すことで捕捉できる。

注：Java では、メソッドの配列パラメーターは常に動的であり、任意の長さの着信配列を受け取ることができます。したがって、COBOL から静的 Java メソッドへの呼び出しは、配列引数の要素数の不一致が起り得ないため、より柔軟である。しかし、Java から COBOL に返される配列では、長さが正確に一致しなければならない。

第 45 章 Java から COBOL WORKING-STORAGE データ項目へのアクセス

このトピックでは、Java から COBOL WORKING-STORAGE のデータ項目にアクセスする方法について説明します。

JAVA-SHAREABLE ON 指令の有効範囲内にある COBOL プログラム内の Java 互換データ項目には、Java からアクセスできます。この指令は、01 /77 レベルの項目にのみ適用されます。そのため、JAVA-SHAREABLE ON 指令がグループの中央にある場合は、後続の 01 /77 レベルの項目に適用されます。ただし、01 レベルのグループ項目が JAVA-SHAREABLE ON 指令の有効範囲内にある場合は、グループ内の 01 レベルの項目に従属するすべてのデータ項目に Java からアクセスできます。

データ項目が Java と共用可能な場合、コンパイラーは、Java コードからデータ項目のメモリーに対する読み取りと書き込みの両方のインターフェースを定義する Java クラスを生成します。インターフェース・クラスは、実行時にインスタンス化され、cjbuidl の実行時に -p オプションによって示されるパッケージに属する strg という特殊なクラスを介してアクセスできます。ここで、enterprise.COBOL はデフォルトのパッケージ名です。strg クラスは、Java と WORKING-STORAGE メモリーを共用するアプリケーション内の COBOL プログラムごとに 1 つのインターフェースを提供し、プログラムごとに、共用可能な WORKING-STORAGE 01 /77 レベル項目ごとに 1 つのインターフェースを提供します。

注：

1. COBOL と Java の間でデータを共用するメカニズムには COBOL の EXTERNAL データ項目が含まれるため、Java 共用可能なデータ項目は、アプリケーションの COBOL 部分全体で固有の名前を持つ必要があります。そうしないと、名前が競合する可能性があります。Java で共有可能なデータ項目の名前も 26 文字以下でなければならない。
2. COBOL プログラムの WORKING-STORAGE メモリーに Java からアクセスする前に、プログラムの WORKING-STORAGE を最初に初期設定する必要があります。これは、プログラムが Java から直接または間接的に呼び出された場合にのみ発生します。
3. Java 互換タイプではない JAVA-SHAREABLE ON ディレクティブの有効範囲内にあるデータ項目は、すべて無視されます。

例

Java 呼び出し可能 COBOL プログラム PROGA に定義されている以下の Java 共用可能グループについて考えます。

```
JAVA-SHAREABLE ON
01 g1.
   03 data1 pic s9(9) comp-5.
   03 data2 pic s9(5)v9(4) comp-3.
01 f1 comp-1.
JAVA-SHAREABLE OFF
```

Java 呼び出し可能 COBOL プログラム PROGB に定義されている以下の Java 共用可能グループについて考えます。

```
JAVA-SHAREABLE ON
01 g2.
   03 data1 pic x(20).
   03 data2 pic comp-2.
01 i1 pic s9(9) comp-5.
JAVA-SHAREABLE OFF
```


cjbuild を使用するアプリケーションの DLL を構築するときに、デフォルト・パッケージ `enterprise.COBOL` が使用されたと想定します。以下の Java コードを使用して、これらのデータ項目にアクセスできます。

```
import enterprise.COBOL.*;
import java.math.*;

public class TestApp
{
    public static void main(String[] args)
    {
        // Make sure COBOL programs have been called at least once
        // to initialize their WORKING-STORAGE
        enterprise.COBOL.progs.PROGA(); // assume no arguments to pass
        enterprise.COBOL.progs.PROGB(); // assume no arguments to pass

        // get data item values from PROGA
        int g1_data1 = enterprise.COBOL.strg.PROGA.G1.data1.get();
        BigDecimal g1_data2 = enterprise.COBOL.strg.PROGA.G1.data2.get();
        float f1 = enterprise.COBOL.strg.PROGA.F1.get();

        // put data items from PROGA
        enterprise.COBOL.strg.PROGA.G1.data1.put(25);
        enterprise.COBOL.strg.PROGA.G1.data2.put(new BigDecimal(25.3645));
        enterprise.COBOL.strg.PROGA.F1.put(3.1415f);

        // get data item values from PROGB
        String g2_data1 = enterprise.COBOL.strg.PROGB.data1.put();
        double g2_data2 = enterprise.COBOL.strg.PROGB.data2.put();
        int i1 = enterprise.COBOL.strg.PROGB.I1.get();

        // put data items from PROGB
        enterprise.COBOL.strg.PROGB.data1.put("Hello, world!");
        enterprise.COBOL.strg.PROGB.data2.put(3.1415926);
        enterprise.COBOL.strg.PROGB.I1.put(26);
    }
}
```

関連参照

JAVA-CALLABLE (*Enterprise COBOL for z/OS* 言語解説書)

JAVA-SHAREABLE (*Enterprise COBOL for z/OS* 言語解説書)

関連タスク

[334 ページの『ステップ 2: Java スタブ・プログラムの DLL を構築するための cjbuild ユーティリティの実行』](#)

第 46 章 Java 相互運用可能 COBOL アプリケーションでのエラー処理

このトピックでは、COBOL/Java インターオペラビリティ・エラー処理に関する提案を提供します。

COBOL が静的な ジャワ メソッド CALL 文を使って を呼び出す場合、アプリケーションの ジャワ 側で例外が発生することがある または Java コール・スタブ・プログラムの中にある。コール中に発生した例外はすべて、CALL ステートメントの ON EXCEPTION フレーズを使って処理できる。例外が発生し、ON EXCEPTION フレーズが指定された場合、ON EXCEPTION フレーズのコードが実行される前に例外がクリアされる。必要であれば、ユーザー例外処理コードは、IGY-JAVAIOP-CALL-EXCEPTION 特殊レジスタを通して、例外の原因となった Java 例外オブジェクトにアクセスできる。呼び出しプログラムのチェーンに例外を伝播するには、JNI 関数 Throw() を使用して例外オブジェクトを再スローできます。例外が発生し、ON EXCEPTION フレーズが指定されない場合、COBOL 実行単位は処理されない例外により停止する。COBOL が静的 Java メソッドを呼び出すときに、アプリケーションの Java 側で発生する例外を処理する別の方法は、静的 Java メソッドが例外をキャッチし、COBOL 呼び出し元に返すリターン・コードに変換することである。COBOL 側では、COBOL アプリケーションは戻りコードに基づいて何を実行するかを決定できます。

Java が COBOL を呼び出す場合、JNI 関数の実行時にネイティブ・メソッド呼び出しスタブ・プログラムで Java 例外が発生しても、スタブ・プログラムは例外をクリアせず、例外をキャッチして適切に処理できる Java 側のアプリケーションに直ちに戻る。例外が処理されない場合、Java アプリケーションは未処理の例外により終了する。

第 47 章 AMODE の考慮事項

このトピックでは、Java と相互運用する非 OO COBOL アプリケーションのさまざまな可能な AMODE 構成について説明します。

Java と相互運用する非 OO COBOL アプリケーションは、以下の 3 つの AMODE 構成のいずれかで実行できます。

1. [819 ページの『AMODE 31 \(31 ビット\) の COBOL および AMODE 31 \(31 ビット\) の Java』](#)
2. [AMODE 31 \(31 ビット\) の COBOL および AMODE 64 \(64 ビット\) の Java](#)
3. [AMODE 64 \(64 ビット\) の COBOL および AMODE 64 \(64 ビット\) の Java](#)

AMODE 31 (31 ビット) の COBOL および AMODE 31 (31 ビット) の Java

AMODE 31 COBOL と AMODE 64 Java で構成される COBOL/Java 相互運用可能アプリケーションを構築するには、LP オプションが指定されていない場合のデフォルトである LP(32) オプションを有効にして、COBOL ユーザー・プログラムをコンパイルし、cjbuild ユーティリティの実行時に -m PURE31 オプションを指定します。31 ビット Java SDK の Java コンパイラーを使用して、アプリケーション内のすべての Java ファイルをコンパイルします。

AMODE 31 (31 ビット) の COBOL および AMODE 64 (64 ビット) の Java

AMODE 31 COBOL と AMODE 64 Java で構成される COBOL/Java 相互運用可能アプリケーションを構築するには、LP オプションが指定されていない場合のデフォルトである LP(32) オプションを有効にして、COBOL ユーザー・プログラムをコンパイルし、cjbuild ユーティリティの実行時に -m MIX_31_64 オプションを指定します。64 ビット Java SDK の Java コンパイラーを使用して、アプリケーション内のすべての Java ファイルをコンパイルします。

AMODE 64 (64 ビット) の COBOL および AMODE 64 (64 ビット) の Java

AMODE 64 COBOL と AMODE 64 Java で構成される COBOL/Java 相互運用可能アプリケーションを構築するには、LP(64) オプションを有効にして COBOL ユーザー・プログラムをコンパイルし、cjbuild ユーティリティの実行時に -m PURE64 オプションを指定します。64 ビット Java SDK の Java コンパイラーを使用して、アプリケーション内のすべての Java ファイルをコンパイルします。

関連タスク

[331 ページの『第 18 章 Java と相互運用する非 OO COBOL アプリケーションのコンパイル、リンク、および実行』](#)

第 8 部 特殊处理

第 48 章 割り込みおよびチェックポイント・リスタート

時間を延長してプログラムを実行していると、ジョブ終了になる前に、割り込みによって処理が停止することがあります。z/OS のチェックポイント・リスタート機能を使用すれば、割り込みを受けたプログラムを、ジョブ・ステップの先頭から、または設定したチェックポイントから再開することができます。

チェックポイント・リスタート機能は、多くの余分な処理を引き起こすので、マシン誤動作、入出力エラー、またはオペレーターの意図的な介入が原因の割り込みであると予想される場合にだけ使用するようになっています。

チェックポイント・ルーチンは、プログラムが入っている COBOL プログラム・オブジェクトから開始されます。プログラムの実行中に、チェックポイント・ルーチンは、COBOL RERUN 節を使用して指定されたポイントでレコードを作成します。チェックポイント・レコードには、プログラムがそのチェックポイントに達したときにレジスターおよび主記憶域の中に保管されていた情報のスナップショットが入れられます。

再始動ルーチンにより、割り込まれたプログラムが再始動されます。プログラムが割り込みを受けた後、いつでも再始動することができます。すなわち、即時に再始動すること（自動再始動）も、後で再始動すること（据え置き再始動）もできます。

関連タスク

[823 ページの『チェックポイントの設定』](#)

[826 ページの『プログラムの再始動』](#)

[829 ページの『再始動用のジョブの再実行依頼』](#)

[z/OS DFSMS: Checkpoint/Restart](#)

関連参照

[825 ページの『チェックポイント・データ・セットの定義用の DD ステートメント』](#)

[826 ページの『チェックポイント時に生成されるメッセージ』](#)

[828 ページの『据え置き再始動の要求用の形式』](#)

チェックポイントの設定

チェックポイントを設定するには、ジョブ制御ステートメントを使用し、ENVIRONMENT DIVISION の RERUN 節を使用してください。それぞれの RERUN 節を特定の COBOL ファイルと関連付ける必要があります。

このタスクについて

RERUN 節は、COBOL ファイル内の特定数のレコードが処理されるたびに、または END OF VOLUME に達したときに、チェックポイント・レコードをチェックポイント・データ・セットに書き込むことを指示します。RERUN 節は、EXTERNAL 属性で定義されたファイルには使用できません。

複数の COBOL ファイルからのチェックポイント・レコードを 1 つのチェックポイント・データ・セットに書き込むことができますが、チェックポイント・レコード専用の別個のデータ・セットを使用する必要があります。チェックポイント・レコードをプログラム・データ・セットの 1 つに組み込むことはできません。

制約事項: チェックポイント・データ・セットは順次編成でなければなりません。チェックポイントを、VSAM データ・セット、または拡張形式 QSAM データ・セットに割り振られたデータ・セットに書き込むことはできません。また、実行単位内のいずれかのプログラムが拡張形式 QSAM データ・セットをオープンしている場合にも、チェックポイントを取ることはできません。

チェックポイント・レコードは、DD ステートメントで定義されたチェックポイント・データ・セットに書き込まれます。DD ステートメントでは、次のチェックポイント方式も選択します。

単一(単一チェックポイントの保管)

ある時点では、チェックポイント・レコードは1つだけ存在します。最初のチェックポイント・レコードが作成されると、それ以降のチェックポイント・レコードは前のレコードにオーバーレイします。

この方式は、ほとんどのプログラムで受け入れられます。チェックポイント・データ・セット上のスペースを節約し、最新のチェックポイントでプログラムを再始動することができます。

複数(複数の連続するチェックポイントを保管します)

チェックポイントは順次記録され、番号が付けられます。それぞれのチェックポイントが保管されます。

最後に取り除かれたチェックポイント以外のチェックポイントでプログラムを再始動したい場合に、この方式を使用します。

85 COBOL 標準に完全に準拠するには、複数チェックポイント方式を使用する必要があります。

ソート操作時のチェックポイントの場合、以下の要件があります。

- ソート操作中にチェックポイントを設定する場合は、実行用のジョブ制御プロシージャの中に SORTCKPT の DD ステートメントを追加します。
- ASCII で照合されるソートでチェックポイント・レコードを設定することができますが、チェックポイント・データ・セットを示す *system-name* に ASCII ファイルを指定してはなりません。

関連タスク

[261 ページの『DFSORT によるチェックポイント・リスタートの使用』](#)

[824 ページの『チェックポイントの設計』](#)

[824 ページの『チェックポイントが成功したかどうかのテスト』](#)

関連参照

[825 ページの『チェックポイント・データ・セットの定義用の DD ステートメント』](#)

チェックポイントの設計

データを容易に再構成できるように、プログラム内の重要ポイントにチェックポイントを設計してください。チェックポイントから再始動までの間に、ファイルの内容を変更してはなりません。

このタスクについて

ディスク・ファイルを使用するプログラムの場合、前に処理されたレコードを識別できるようにプログラムを設計してください。例えば、支払利息が周期的に更新される貸付レコードを含んでいるディスク・ファイルを考えてみましょう。チェックポイントが取られ、レコードが更新された後に、プログラムが割り込まれた場合、最後のチェックポイントの後に更新されたレコードが、プログラムの再始動時に前と同様の更新が行われていないことをテストしたいと思います。これを行うために、各レコードに日付フィールドをセットアップし、レコードが処理されるたびに日付を更新します。次に再始動された後、日付フィールドをテストし、レコードが前に処理されたかどうかを判別します。

印刷ファイルを効率よく位置変更するためには、ページの最後の行を印刷した後にファイル上でチェックポイントを取ってください。

チェックポイントが成功したかどうかのテスト

チェックポイントを出す入力ステートメントまたは出力ステートメントが実行されるたびに、RETURN-CODE 特殊レジスタがチェックポイント・ルーチンからの戻りコードで更新されます。したがって、チェックポイントが成功したかどうかをテストして、再始動に関する条件が適切かどうかを判別できます。

このタスクについて

戻りコードが 4 より大きい場合は、チェックポイントに関するエラーが発生しています。戻りコードを調べ、誤った出力が作成される原因となるような再始動を回避してください。

関連参照

z/OS DFSMS: Checkpoint/Restart (戻りコード)

チェックポイント・データ・セットの定義用の DD ステートメント

チェックポイント・データ・セットを定義するには、DD ステートメントを使用します。

テープの場合:

```
//ddname DD DSN=dataset-name,
//          [VOLUME=SER=volser,]UNIT=device-type,
//          DISP=(NEW|MOD, PASS)
```

直接アクセス装置の場合:

```
//ddname DD DSN=dataset-name,
//          [VOLUME=(PRIVATE, RETAIN, SER=volser),]
//          UNIT=device-type, SPACE=(subparms),
//          DISP=(NEW|MOD, PASS, KEEP)
```

ddname

DD ステートメントへのリンクを提供します。COBOL RERUN 節で使用される *assignment-name* の ddname 部分と同じ。

data-set-name

再始動プロシージャに対してチェックポイント・データ・セットを識別します。チェックポイント・レコードを記録するために使用される、データ・セットに与えられる名前。

volser

通し番号でボリュームを識別します。

device-type

装置を識別します。

subparms

データ・セットに必要なトラック・スペースの大きさを指定します。

MOD

複数連続チェックポイント・メソッドを指定します。

NEW

単一チェックポイント・メソッドを指定します。

PASS

ジョブの最後のジョブ・ステップを除き、ジョブ・ステップの正常完了時に、データ・セットの削除を妨げます。最後のステップであれば、データ・セットは削除されます。

KEEP

ジョブ・ステップが異常終了した場合に、データ・セットを保管します。

825 ページの『例: チェックポイント・データ・セットの定義』

例: チェックポイント・データ・セットの定義

以下の例は、チェックポイント・データ・セットの定義に使用できる、JCL および COBOL コーディングを示しています。

テープを使用した、単一チェックポイント・レコードの作成:

```
//CHECKPT DD DSN=CHECK1, VOLUME=SER=ND0003,
//          UNIT=TAPE, DISP=(NEW, KEEP), LABEL=(, NL)

ENVIRONMENT DIVISION.

REERUN ON CHECKPT EVERY
5000 RECORDS OF ACCT-FILE.
```

ディスクを使用した、単一チェックポイント・レコードの作成:

```
//CHEK      DD DSNAME=CHECK2,  
//          VOLUME=(PRIVATE,RETAIN,SER=DB0030),  
//          UNIT=3380,DISP=(NEW,KEEP),SPACE=(CYL,5)  
  
ENVIRONMENT DIVISION.  
  
        RERUN ON CHEK EVERY  
        20000 RECORDS OF PAYCODE.  
        RERUN ON CHEK EVERY  
        30000 RECORDS OF IN-FILE.
```

テープを使用した、複数の連続チェックポイント・レコードの作成:

```
//CHEKPT    DD DSNAME=CHECK3,VOLUME=SER=111111,  
//          UNIT=TAPE,DISP=(MOD,PASS),LABEL=(,NL)  
  
ENVIRONMENT DIVISION.  
  
        RERUN ON CHEKPT EVERY  
        10000 RECORDS OF PAY-FILE.
```

チェックポイント時に生成されるメッセージ

システム・チェックポイント・ルーチンは、コンソールに通知メッセージを表示することにより、取られたチェックポイントの状況をオペレーターに知らせます。

チェックポイントが正常に完了するたびに、ジョブ名 (*ddname*、*unit*、*volser*) を、取られたチェックポイント (*checkid*) に関連付けるメッセージが表示されます。

制御プログラムは、*checkid* を 8 文字の文字ストリングとして割り当てます。先頭文字は文字 C で、その後チェックポイントを示す 10 進数が続きます。例えば、次のメッセージは、ジョブ・ステップで設定された 4 番目のチェックポイントを示します。

```
checkid C0000004
```

プログラムの再始動

システム再始動ルーチンは、チェックポイント・レコードに記録された情報を取得し、主ストレージおよびすべてのレジスターの内容を復元し、プログラムを再始動します。

このタスクについて

再始動ルーチンは、次のいずれかの方法で開始することができます。

- 割り込みによってプログラムが停止した時に、自動的に
- 後で据え置き再始動として

ジョブ制御言語の RD パラメーターは、再始動のタイプを決定します。RD パラメーターは、JOB または EXEC ステートメントのいずれかに指定することができます。JOB ステートメントにコーディングされた場合、このパラメーターは、EXEC ステートメントの RD パラメーターをオーバーライドします。

再始動とチェックポイント書き込みの両方を抑止する場合は、RD=NC をコーディングします。

制約事項: SORT または MERGE 操作時に COBOL プログラムによって取られたチェックポイントで再始動しようとする、エラー・メッセージが出され、再始動は取り消されます。DFSORT によって取られたチェックポイントだけが有効です。

データ・セットの DD ステートメントに SYSOUT パラメーターが指定されていると、そのデータ・セットは、再始動のタイプに応じてさまざまな方法で処理されます。

チェックポイント・データ・セットがマルチボリュームの場合は、チェックポイント項目が書き込まれたボリュームのシーケンス番号を、VOLUME パラメーターに含めなければなりません。チェックポイントデ

ータセットが非標準ラベルまたはラベルなしの 7 トラックテープ上にある場合、SYSCHK DD 文には次の内容を含める必要があります DCB=(TRTCH=C, . . .)。

関連タスク

[261 ページの『DFSORT によるチェックポイント・リスタートの使用』](#)

[827 ページの『自動再始動の要求』](#)

[827 ページの『据え置き再始動の要求』](#)

自動再始動の要求

自動再始動が行われるのは、最新のチェックポイントが取られたときだけです。割り込みの前にチェックポイントが取られなかった場合、自動再始動は、ジョブ・ステップの先頭から行われます。

このタスクについて

自動再始動が行われる時は、システムはユニット・レコード装置を除くすべての装置の位置変更をします。

自動再始動を行いたい場合は、RD=R または RD=RNC をコーディングします。

- RD=R は、最新のチェックポイントで再始動を行うことを示します。チェックポイントを記録するためには、プログラム中の少なくとも 1 つのデータ・セットに対して RERUN 節をコーディングしてください。割り込みの前にチェックポイントが取られなかった場合、自動再始動は、ジョブ・ステップの先頭から行われます。
- RD=RNC は、チェックポイントを書き込まないこと、およびどの再始動もジョブ・ステップの先頭から行うことを示します。この場合、RERUN 節は不要です。もしあっても、無視されます。

RD パラメーターを省略すると、CHKPT マクロ命令がアクティブのままとなり、処理中にチェックポイントが取られることがあります。最初のチェックポイントの後で割り込みが行われる場合、自動再始動が行われます。

自動的に再始動するためには、プログラムが次の条件を満たしていなければなりません。

- プログラムでは、RD パラメーターを使用するか、またはチェックポイントを取ることによって、再始動を要求しなければなりません。
- ジョブを中止させた異常終了は、再始動を可能にするコードを戻さなければなりません。
- オペレーターが、再始動を許可しなければなりません。

[829 ページの『例: ステップ再始動の要求』](#)

据え置き再始動の要求

据え置き再始動は、取られたチェックポイントが必ずしも最新のものでなくても、任意のチェックポイントで行うことが可能です。ジョブ・ステップの先頭以外のチェックポイントからプログラムを再始動することができます。

このタスクについて

据え置き再始動が正常に完了すると、システムは、ジョブが再始動したことを示すメッセージをコンソールに表示します。それから、制御権がプログラムに与えられます。

据え置き再始動を行いたい場合は、RD パラメーターを RD=NR としてコーディングします。この形式のパラメーターは自動再始動を抑止しますが、RERUN 節がコーディングされていれば、チェックポイント・レコードの書き込みを許可します。

据え置き再始動は、JOB カードで RESTART パラメーターを指定し、チェックポイント・データ・セットを識別する SYSCHK DD ステートメントを使用して、要求してください。もし SYSCHK DD ステートメントがジョブに存在し、JOB 声明には含まれていない RESTART パラメータ、SYSCHK DD ステートメントは無

視されます。CHECKID サブパラメーターの指定なしの RESTART パラメーターをジョブに含める場合は、ジョブの最初の EXEC ステートメントの前に SYSCHK DD ステートメントがあってはなりません。

829 ページの『例: 特定チェックポイント・ステップでのジョブの再始動』

関連参照

828 ページの『据え置き再始動の要求用の形式』

据え置き再始動の要求用の形式

JOB ステートメントおよび SYSCHK DD ステートメントの RESTART パラメーターの形式は次のとおりです。

```
//jobname JOB MSGLEVEL=1,RESTART=(request[,checkid])
//SYSCHK DD DSN=dataset-name,
// DISP=OLD[,UNIT=device-type,
// VOLUME=SER=volser]
```

MSGLEVEL=1 (または MSGLEVEL=(1,y))

MSGLEVEL は必須です。

RESTART=(request,[checkid])

再始動が行われる特定のチェックポイントを識別します。

request

次のいずれかの形式を取ります。

ジョブの先頭から再始動することを示します。

stepname

ジョブ・ステップの先頭から再始動することを示します。

stepname.procstep

ジョブ・ステップ内のプロシージャ・ステップから再始動することを示します。

checkid

再始動が行われるチェックポイントを指示します。

SYSCHK

チェックポイント・データ・セットを制御プログラムに識別するために使用される DD 名。SYSCHK DD ステートメントは、再実行依頼されたジョブの最初の EXEC ステートメントの直前で、かつ JOBLIB ステートメントの後になければなりません。

data-set-name

チェックポイント・データ・セットを識別します。チェックポイントが取られた時に使用された名前と同じでなければなりません。

device-type および **volser**

チェックポイント・データ・セットが入っているボリュームの装置タイプおよび通し番号を識別します。

828 ページの『例: 据え置き再始動の要求』

例: 据え置き再始動の要求

この例は、チェックポイント ID (CHECKID) C0000003 で IGYWCLG プロシージャの GO ステップを再始動する JCL を示しています。

```
//jobname JOB MSGLEVEL=1,RESTART=(stepname.GO,C0000003)
//SYSCHK DD DSN=CHECKPT,
// DISP=OLD[,UNIT=3380,VOLUME=SER=111111]
// . . .
```

再始動用のジョブの再実行依頼

再始動のためにジョブを再実行依頼する場合、再始動されるジョブ・ステップの実行に影響する可能性のある、どの DD ステートメントについても注意してください。再始動ルーチンは、再入力されたジョブの DD ステートメントの情報を使用して、再始動後に使用するファイルをリセットします。

このタスクについて

ジョブ・ステップ終了時にデータ・セットを削除したい場合は、(DELETE ではなく) PASS または KEEP の条件付き後処理を指定してください。この後処理を指定すると、割り込みによって強制的に再始動が実行される場合にそのデータ・セットが使えるようになります。ステップの先頭からジョブを再始動したい場合は、前回の実行で作成したデータ・セット (DD ステートメントで NEW と定義された) をすべて廃棄するか、またはデータ・セットに OLD のマークを付けるように DD ステートメントを変更しなければなりません。

テープまたはディスク上の入力データ・セットは、システムによって自動的に再配置されます。

829 ページの『例: ステップ再始動のためのジョブの再実行依頼』

830 ページの『例: チェックポイント・リスタートのためのジョブの再実行依頼』

例: 特定チェックポイント・ステップでのジョブの再始動

この例は、特定ステップでジョブを再始動するための一連のジョブ制御ステートメントを示しています。

```
//PAYROLL JOB MSGLEVEL=1,REGION=80K,  
//          RESTART=(STEP1,CHECKPT4)  
//JOB LIB DD DSNAME=PRIV.LIB3,DISP=OLD  
//SYSCHK DD DSNAME=CHKPTLIB,  
//          [UNIT=TAPE,VOL=SER=456789,]  
//          DISP=(OLD,KEEP)  
//STEP1 EXEC PGM=PROG4,TIME=5
```

例: ステップ再始動の要求

この例は、異常終了したジョブ・ステップのステップ再始動を要求する RD パラメーターの使用法を示しています。

```
//J1234 JOB 386,SMITH,MSGLEVEL=1,RD=R  
//S1 EXEC PGM=MYPROG  
//INDATA DD DSNAME=INVENT[,UNIT=TAPE],DISP=OLD,  
//          [VOLUME=SER=91468,]  
//          LABEL=RETPD=14  
//REPORT DD SYSOUT=A  
//WORK DD DSNAME=T91468,DISP=(,KEEP),  
//          UNIT=SYSDA,SPACE=(3000,(5000,500)),  
//          VOLUME=(PRIVATE,RETAIN,,6)  
//DDCKPNT DD UNIT=TAPE,DISP=(MOD,PASS,CATLG),  
//          DSNAME=C91468,LABEL=(,NL)
```

DDCKPNT DD ステートメントはチェックポイント・データ・セットを定義します。このステップの場合、RERUN 節の実行後、CHKPT 取り消しが出されなければ、自動チェックポイント・リスタートを行うことができます。

例: ステップ再始動のためのジョブの再実行依頼

次の例は、ステップ再始動のためにジョブを再実行依頼する前に、JCL に対して行える変更を示しています。

```
//J3412 JOB 386,SMITH,MSGLEVEL=1,RD=R,RESTART=*  
//S1 EXEC PGM=MYPROG  
//INDATA DD DSNAME=INVENT[,UNIT=TAPE],DISP=OLD,
```



```
//          [VOLUME=SER=91468,]LABEL=RETPD=14
//REPORT   DD  SYSOUT=A
//WORK      DD  DSNAME=S91468,
//          DISP=(,KEEP),UNIT=SYSDA,
//          SPACE=(3000,(5000,500)),
//          VOLUME=(PRIVATE,RETAIN,,6)
//DDCHKPNT DD  UNIT=TAPE,DISP=(MOD,PASS,CATLG),
//          DSNAME=R91468,LABEL=(,NL)
```

上記の例では以下の変更が行われました。

- 元のジョブと再始動されたジョブを区別するために、ジョブ名が変更されました (J1234 から J3412 へ)。
- RESTART パラメーターが JOB ステートメントに追加されて、最初のジョブ・ステップから再始動を開始するように指示します。
- WORK DD ステートメントには、このデータ・セットに対して KEEP という条件付き後処理が最初に割り当てられていました。
 - 前回ジョブを実行した際にこのステップが正常終了した場合には、データ・セットは削除されているので、このステートメントに変更を加える必要はありません。
 - ステップが異常終了した場合、データ・セットは保管されています。その場合は、新しいデータ・セット (示されているように T91468 ではなく S91468) を定義するか、またはジョブを再実行依頼する前にデータ・セットの状況を OLD に変更しなければなりません。
- 新しいデータ・セット (C91468 ではなく R91468) も、チェックポイント・データ・セットとして定義されました。

829 ページの『例: ステップ再始動の要求』

例: チェックポイント・リスタートのためのジョブの再実行依頼

次の例は、チェックポイント・リスタートのためにジョブを再実行依頼する前に、JCL に対して行える変更を示しています。

```
//J3412     JOB  386,SMITH,MSGLEVEL=1,RD=R,
//          RESTART=(*,C00000002)
//SYSCHK   DD  DSNAME=C91468,DISP=OLD
//S1       EXEC PGM=MYPROG
//INDATA   DD  DSNAME=INVENT,UNIT=TAPE,DISP=OLD,
//          VOLUME=SER=91468,LABEL=RETPD=14
//REPORT   DD  SYSOUT=A
//WORK      DD  DSNAME=T91468,DISP=(,KEEP),
//          UNIT=SYSDA,SPACE=(3000,(5000,500)),
//          VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT  DD  UNIT=TAPE,DISP=(MOD,KEEP,CATLG),
//          DSNAME=C91468,LABEL=(,NL)
```

上記の例では以下の変更が行われました。

- 元のジョブと再始動されたジョブを区別するために、ジョブ名が変更されました (J1234 から J3412 へ)。
- RESTART パラメーターが JOB ステートメントに追加されて、C00000002 という名前のチェックポイント項目で最初のステップから再始動を開始するように指示します。
- DD ステートメントの DDCKPNT では、チェックポイント・データ・セットに CATLG という条件付き後処理が最初に割り当てられました。
 - 前回ジョブを実行した際にこのステップが正常終了した場合には、データ・セットは保管されています。その場合、SYSCHK DD ステートメントには、チェックポイント・データ・セットの検索に必要なすべての情報が含まれていなければなりません。
 - ジョブが異常終了した場合、データ・セットはカタログされています。その場合は、示されているように、SYSCHK DD ステートメントに必要なパラメーターは DSNAME と DISP だけです。

V=R が指定された場合は、実行されているジョブでチェックポイントが取られても、十分なページング不能な動的ストレージが使用可能になるまで、そのジョブは再始動することはできません。

第 49 章 COBOL プログラムからの zlib 圧縮の使用

COBOL プログラムから zlib 圧縮を使用する手順とヒントを含む次の例を参照してください。

例

```
//PROCLIB JCLLIB ORDER=IGYV6R50.SIGYPROC
//EXAMPLE1 EXEC IGYWCLG
//COBOL.STEPLIB DD DISP=SHR,DSNAME=IGYV6R50.SIGYCOMP
//SYSADATA DD SYSOUT=*
//COBOL.SYSIN DD *
    CBL PGMNAME(LONGMIXED)
    *****
    *
    * This is a sample testcase to show how you can call zlib.
    * It initializes compression (deflate) and then deflates some
    * data, then initializes decompression (inflate) and inflates
    * the same data.
    *
    * Some of the key differences from 'typical' COBOL are:
    * - Compiler option PGMNAME(LONGMIXED) is required.
    *   This means that the Program-Id name has to be a literal
    * - This program uses COMP-5 data items so that the program
    *   can work with any setting of the TRUNC compiler option
    * - Rather than passing the zstream structure by REFERENCE like
    *   most COBOL programs, we pass the ADDRESS OF using the
    *   BY VALUE phrase to avoid high-order bit getting set
    * - This sample has DISPLAY statements to tell you if it was
    *   successful or not, and they should be removed for production*
    * - The binder (linkage editor) needs to have LIBRARY
    *   statements in order to access the functions in the
    *   /usr/lpp/hzc/lib/libz.a archive file. See the sample
    *   LKED.SYSIN statements below
    *
    *****
    Id Division.
    Program-id. 'ZLIB'.
    *
    Data Division.

    Working-Storage Section.
    **> Memory for tests. 'data' to be compressed
    01 zinput   pic X(65536) value x'00'.
    01 zoutput  pic X(65536) value x'00'.
    01 z.
    *
    * next_in is a Pointer to zstring
    02 next_in   Pointer.
    02 avail_in  Pic S9(9)   Comp-5.
    02 total_in  Pic S9(9)   Comp-5.
    *
    * next_out is a Pointer to zstring
    02 next_out  Pointer.
    02 avail_out Pic S9(9)   Comp-5.
    02 total_out Pic S9(9)   Comp-5.
    *
    * msg is a Pointer to zstring
    02 msg       Pointer.
    02 state     Pointer.
    02 zalloc    Function-pointer.
    02 zfree     Function-pointer.
    02 opaque    Function-pointer.
    02 data_type Pic S9(9)   Comp-5.
    02 Adler     Pic S9(9)   Comp-5.
    02 reserved  Pic S9(9)   Comp-5.
    01 y.
    *
    * next_in is a Pointer to zstring
    02 next_in   Pointer.
    02 avail_in  Pic S9(9)   Comp-5.
    02 total_in  Pic S9(9)   Comp-5.
    *
    * next_out is a Pointer to zstring
    02 next_out  Pointer.
    02 avail_out Pic S9(9)   Comp-5.
    02 total_out Pic S9(9)   Comp-5.
    *
    * msg is a Pointer to zstring
    02 msg       Pointer.
    02 state     Pointer.
    02 zalloc    Function-pointer.
    02 zfree     Function-pointer.
```



```

02 opaque          Function-pointer.
02 data_type       Pic S9(9)   Comp-5.
02 adler           Pic S9(9)   Comp-5.
02 reserved       Pic S9(9)   Comp-5.

77 rc-disp Pic S9(9) Display Sign leading separate.

Linkage Section.
77 zstring Pic x(1000).
77 rc      Pic S9(9) Comp-5.

Procedure Division returning rc.
**>*****/
**>                                          */
**> Setup the z_stream structure with defaults */
**>                                          */
**>*****/
      Set zalloc of z to Null
      Set zfree  of z to Null
      Set opaque of z to Null

*      *>*****/
*      *>                                          */
*      *> Call deflateInit to initialize the deflate stream */
*      *> service using a GZIP wrapper */
*      *>                                          */
*      *>*****/
      Call 'DEIN2' Using By Value Address of z,
                                By Value -1, 8, 31, 8, 0,
                                By Content '1.2.7',
                                By Value Length Of z

      returning rc

      If rc NOT = 0 Then
        Move rc to rc-disp
        Display 'Error: deflateInit failed with Return Code '
                                rc-disp

        If msg of z NOT = Null Then
          Set Address of zstring to msg of z
          Display 'Message = ' zstring
        Else
          Display 'Message pointer is NULL '
        End-if
        Move -1 To Return-code rc

        Goback
      Else
        Display 'deflatinit Successful ! '
      End-if

*****
* Initialize available input, output, total in for deflate
*****
      Compute avail_in of z = 65536
      Compute avail_out of z = 65536
      Compute total_in of z = 0
*****
* Set input and output pointers
*****
      Set next_out of z to Address of zoutput
      Set next_in of z to Address of zinput

*      *>*****/
*      *>                                          */
*      *> Call deflate to compress the data. We only call it */
*      *> once with a flush mode of Z_FINISH (4) which indicates */
*      *> to end the stream */
*      *>                                          */
*      *>*****/
      Call 'deflate' Using By Value Address of z
                                BY Value 4 Returning rc

      If rc NOT = 1 Then
        Move rc to rc-disp
        Display ' Error: deflate returned ' rc-disp

        If msg of z NOT = Null Then
          Set Address of zstring to msg of z
          Display zstring
        End-if
        Display "avail_in of z= " avail_in of z
        Display "total_in of z= " total_in of z

```

```

        Display "avail_out of z= "   avail_out of z
        Display "total_out of z= "   total_out of z

        Call 'DEEND' Using By Value address of z returning rc

        Move -1 To Return-code rc
        Goback
    Else
        Display 'deflate Successful ! '
    End-if

    Display 'After deflate '
    Display 'avail_in of z= '   avail_in of z
    Display 'total_in of z= '   total_in of z
    Display 'avail_out of z= '   avail_out of z
    Display 'total_out of z= '   total_out of z

* /*****
* /*
* /* Now that we are done , call deflateEnd to cleanup the */
* /* deflate internal state.                               */
* /*
* /* *****/
    Call 'DEEND' Using By Value Address of z returning rc
    If rc NOT = 0 Then
        Move rc to rc-disp
        Display 'Error: deflateEnd returned ' rc-disp

        If msg of z NOT = Null Then
            Move -1 To Return-code rc
            Goback
        End-if
    Else
        Display 'deflateEnd Successful ! '
    End-if

* /*****
* /*
* /* Inflate the data we just deflated. Call inflateInit to */
* /* initialize the inflate stream                          */
* /*
* /* *****/
    Set zalloc of y to Null
    Set zfree of y to Null

    Set opaque of y to Null

    Call 'ININ2' Using By Value Address of y,
                                     By Value 31,
                                     By Content '1.2.7',
                                     By VALUE Length Of y

        returning rc

    If rc NOT = 0 Then
        Move rc to rc-disp
        Display 'Error: inflateInit failed with Return Code '
                                     rc-disp

        If msg of y NOT = Null Then
            Set Address of zstring to msg of y
            Display zstring
        End-if
        Move -1 To Return-code rc
        Goback
    Else
        Display 'inflateInit Successful ! '
    End-if

* /*****
* /*
* /* Set the amount of input based on what deflate returned */
* /* and what we expect the output size to be.             */
* /*
* /* *****/
    Compute avail_in of y = total_out of z
    Compute avail_out of y = 65536
    Compute total_in of y = 0

    Set next_out of y to Address of zinput
    Set next_in of y to Address of zoutput

```

```

* /*****
* /*
* /* Call inflate to decompress the data. Note that we expect */
* /* this to end with a Z_STREAM_END (1)since we provided the */
* /* full stream above. */
* /*
* /*****
  Call 'inflate' Using By Value Address of y
                  By Value 0 Returning rc
  If rc Not = 1 Then
    Move rc to rc-disp
    Display 'Error: inflate returned ' rc-disp

    If msg of z NOT = Null Then
      Set Address of zstring to msg of z
      Display zstring
    End-if

    Display "avail_in of y= " avail_in of z
    Display "total_in of y= " total_in of z
    Display "avail_out of y= " avail_out of z
    Display "total_out of y= " total_out of z

    Call 'INEND' Using By Value Address of y Returning rc
    Move -1 To Return-code rc
    Goback
  Else
    Display 'inflate Successful ! '
  End-if

  Display 'After inflate '
  Display 'avail_in of y= ' avail_in of y
  Display 'total_in of y= ' total_in of y
  Display 'avail_out of y= ' avail_out of y
  Display 'total_out of y= ' total_out of y

* /*****
* /*
* /* Now that we are done call inflateEnd to cleanup the */
* /* internal inflate state of the stream. */
* /*
* /*****
  Call 'INEND' Using By Value Address of y Returning rc
  If rc Not = 0 Then

  If rc Not = 0 Then
    Move rc to rc-disp
    Display 'Error: inflateEnd returned ' rc-disp

    If msg of z NOT = Null Then
      Move -1 To Return-code rc
      Goback
    End-if
  Else
    Display 'inflateEnd Successful ! '
  End-if

  Move zero to return-code rc
  Goback.
/*
//LKED.SYSIN DD *
  LIBRARY '/usr/lpp/hzc/lib/libz.a'
/*
//LKED.SYSLIB DD DSN=CEEZ240.SCEELKED,DISP=SHR
//LKED.SYSLMOD DD DSN=MYLOAD.COBOL.LOAD(ZLIB),DISP=SHR
//GO.STEPLIB DD DSN=CEEV240.SCEERUN,DISP=SHR
//GO.SYSUDUMP DD DUMMY
//GO.SYSSORT DD DUMMY
//GO.SYSOUT DD SYSOUT=*
//

```

第 9 部 パフォーマンスおよび生産性の向上

第 50 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。制御フローが複雑であると、プログラムの理解や保守が困難になり、コードの最適化が妨げられます。

このタスクについて

プログラムのパフォーマンスを向上させるには、少なくとも以下の点を調査してください。

- 基になるアルゴリズム: 最高のパフォーマンスを得るには、健全なアルゴリズムを使用することが不可欠です。例:
 - 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。
 - プログラムが頻繁にデータにアクセスする場合は、データにアクセスするステップの数を減らします。
- データ構造: アルゴリズムに適したデータ構造を使用することが不可欠です。

より優れたコード・シーケンスを生成し、システム・サービスをより効率的に活用するようなプログラムを作成することができます。さらに、以下の点がパフォーマンスに影響します。

- コーディング技法: 最適化プログラムが効率的なデータ型を選択し、表を効率的に処理できるようにするプログラミング・スタイルを使用します。
- 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: 一部のコンパイラー・オプションと言語は、プログラムの効率に影響を与えます。
- ランタイム環境: ランタイム・オプションの選択を考慮します。
- CICS、IMS、または VSAM での実行: さまざまなヒントに留意すると、これらのプログラムをより効率的に実行する上で役立ちます。

関連概念

[843 ページの『最適化』](#)

関連タスク

[837 ページの『最適なプログラミング・スタイルの使用』](#)

[839 ページの『効率的なデータ型の選択』](#)

[841 ページの『テーブルの効率的処理』](#)

[843 ページの『コードの最適化』](#)

[844 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』](#)

[850 ページの『CICS、IMS、または VSAM での効率的な実行』](#)

言語環境プログラミングガイド (ランタイム・オプションの指定)

関連参照

[845 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』](#)

言語環境プログラミングガイド (ストレージ・パフォーマンスの考慮事項)

Enterprise COBOL for z/OS パフォーマンス・チューニング・ガイド

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、一括表示表現、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。

このタスクについて

関連タスク

[838 ページの『構造化プログラミングの使用』](#)

[838 ページの『一括表示表現』](#)

[838 ページの『シンボリック定数の使用』](#)

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。すると、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。

このタスクについて

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ネージ・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
- 明示 GO TO ステートメント
- 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。

このタスクについて

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複数のステートメントにわたる式の因数処理を行いません。詳しくは、「Enterprise COBOL for z/OS パフォーマンス・チューニング・ガイド」の『一括表示表現』を参照してください。

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。

このタスクについて

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。

効率的なデータ型の選択

使用方法 SYNCHRONIZED 句を使用すると、より効率的なコードを生成できます。

このタスクについて

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

[53 ページの『数値データの形式』](#)

関連タスク

[839 ページの『効率的な計算データ項目の選択』](#)

[840 ページの『一貫性のあるデータ型の使用』](#)

[840 ページの『算術式の効率化』](#)

[840 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。

このタスクについて

しかし、固定小数点算術ステートメントの中間結果で精度 (有効数字の数) が大きい場合、コンパイラーは、オペランドをパック 10 進数形式、10 進数浮動小数点形式、またはベクトル 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 データ項目で生成されるコードは、場合によっては (特にステートメントが複雑であるか、丸めを指定している場合は) BINARY データ項目で生成されるコードと同じ速さになることがあります。

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)。
- 15 桁以下の PICTURE 仕様 ARCH(7) 機械。PACKED-DECIMAL データ項目が 31 桁を超えている場合は、ライブラリー・ルーチンが使用されます。のために PACKED-DECIMAL16~31 桁のデータ項目 ARCH(8) またはより高レベルのマシンでは、コンパイラはライブラリ呼び出しよりも効率的な命令を使用しますが、データ項目が 15 桁以下の場合ほど高速ではありません。

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。変換ごとにいくつかの命令が必要です。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。

このタスクについて

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。

このタスクについて

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) として定義します。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あります。

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。

このタスクについて

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 ** 1000000.E+00  
COMPUTE fixed-point1 = fixed-point2 ** 1000000
```

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。

効率的な VOLATILE 節の使用

VOLATILE 節で定義されているデータ項目の最適化は、大幅に制限されます。そのため、VOLATILE 節は適切な場合にのみ使用してください。

このタスクについて

特に、VOLATILE 節がグループ項目で使用されると、コンパイラーでは、そのグループ項目より下位のデータ項目はすべて揮発性として扱われ、その揮発性グループ項目を含む上位グループ項目もすべて揮発性として扱われることを理解しておくことが重要です。グループの特定メンバーを揮発性として扱う必要がある場合は、可能であれば、その項目のデータ記述項目に対してのみ VOLATILE 節を指定してください。

現在、VOLATILE 節を使用する主な理由は、LE 条件ハンドラー内部で設定または参照されているが、LE 条件ハンドラー・プログラム外部で定義されているデータ項目に対して使用するためです。VOLATILE 節は、このような項目が最適化プログラムによって正しく処理されることを保証します。使用時期の詳細については VOLATILE、見る VOLATILE 句の中に *Enterprise COBOL for z/OS* 言語解説書。

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。努力の成果が十分報われる可能性があります。テーブル処理操作がアプリケーションの主要部分である場合には特に当てはまります。

このタスクについて

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。

- 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、コードを改善することで **OCCURS DEPENDING ON (ODO)**。可変長項目が参照されるたびに不要な変換を回避するには、次のように指定します。**BINARY** のために **OCCURS . . . DEPENDING ON** オブジェクト。

- 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

[842 ページの『テーブル参照の最適化』](#)

関連タスク

[76 ページの『テーブル内の項目の参照』](#)

[839 ページの『効率的なデータ型の選択』](#)

関連参照

[446 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。

```
comp_s1 * d1 + comp_s2 * d2 + comp_s3 * d3 + 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ライドは d1、d2、および d3 です。ある特定次元のスライドは、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、歩幅 d2 上記の例の 2 番目の次元は、ELEMENT(S1 1S3) そして ELEMENT(S12S3)。

指標計算は添え字計算に類似していますが、乗算をする必要がないという点が異なります。指標値では、それらの中にスライドが含まれています。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。

可変長項目の最適化

従属項目を含むグループ項目 OCCURS DEPENDING ON データ項目の長さは可変です。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。

直接インデックス ELEMENT (I5, J3, K2) この前処理が必要です:

```
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
```

この処理により、直接インデックスは相対インデックスよりも効率が悪くなります。ELEMENT (I + 5, J - 3, K + 2)。

関連概念

[843 ページの『最適化』](#)

関連タスク

[841 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れば、OPTIMIZE(1|2) 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。ご了承ください IBM すべてのユーザーに使用を推奨 OPT(2)最高のパフォーマンスを実現します。

このタスクについて

開発中に再コンパイルせずにプログラムを頻繁に実行する場合にも、OPTIMIZE(1|2)を使用できます。しかし、頻繁に再コンパイルすると、OPTIMIZE(1|2)アセンブラ言語拡張 (LIST コンパイラオプション)を使用してプログラムを微調整します。

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の方が簡単です。

最適化プログラムがプログラムに対してどのように機能するかを確認するには、異なる最適化レベルでコンパイルを行い、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

[843 ページの『最適化』](#)

関連参照

[414 ページの『LIST』](#)

[429 ページの『OPTIMIZE』](#)

最適化

生成されたコードの効率を向上させるには、OPTIMIZE(1) または OPTIMIZE(2) コンパイラー・オプションを使用できます。

OPTIMIZE(1)COBOL オプティマイザが次の最適化を実行します。

- 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みます。
- PERFORM ステートメントに対するコンパイル済みコードを単純化します。コンパイラーは、リンケージ・コードを回避するために PERFORM を何度も複製します。
- 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。
- 実行できないコードをプログラムから削除し、警告メッセージでそのコードを示します (到達不能コードの除去)。
- 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、STGOPT オプションが指定された場合だけです。)

OPTIMIZE(2)を使用すると、COBOL 最適化プログラムはさらに以下の最適化を行います。

- 演算をより積極的に単純化し、命令をスケジュールします。
- グローバル値伝搬やループ不変コード動作など、ブロック間の最適化を行います。

含まれているプログラム・プロシーチャーの統合

含まれているプログラム・プロシーチャーの統合では、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くなる可能性があります。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。

関連概念

844 ページの『[PERFORM プロシージャ統合](#)』

関連参照

429 ページの『[OPTIMIZE](#)』

PERFORM プロシージャ統合

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。この利点は、結果として生じるプログラムが、PERFORM リンケージのオーバーヘッドなしに、より順序正しい制御フローで、より高速に実行されることです。

プログラム・サイズ: 実行されたプロシージャが複数のステートメントによって呼び出され、それぞれが PERFORM ステートメントを置き換えると、プログラムが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制限し、それ以降はプロシージャを統合しません。

関連参照

407 ページの『[INLINE](#)』

INLINE ディレクティブ (*Enterprise COBOL for z/OS* 言語解説書)

パフォーマンスを向上させるコンパイラ機能の選択

どんなパフォーマンス関連コンパイラ・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。

このタスクについて

カスタマイズ済みシステムは、最適なパフォーマンスを得るために特定のオプションを必要とする場合があります。以下の手順を実行します。

手順

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. システム・プログラマーと一緒に、どのオプションがインストール先のオーバーライド不能オプションとして固定されているかを調べる。
3. インストール時に固定されていないオプションについては、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のオプション。

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。

タスクの結果

考慮すべきもう 1 つのコンパイラ機能は、`USE FOR DEBUGGING ON ALL PROCEDURES` 声明。これは、コンパイラ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。

COBOL ではセグメンテーション言語を使用できますが、それを使用してもストレージ割り振りは改善されません。COBOL はオーバーレイを実行しないからです。

関連概念

[843 ページの『最適化』](#)

関連タスク

[843 ページの『コードの最適化』](#)

[500 ページの『リストの入手』](#)

関連参照

[845 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』](#)

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 101. 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
AFP (NOVOLATILE) (『378 ページの『AFP』を参照)	IBM z/Architecture プロセッサによって提供される追加浮動小数点 (AFP) レジスターのコンパイラーでの使用方法を制御する	AFP (NOVOLATILE) を使用すると、コンパイラーは、浮動小数点演算を使用するプログラムに対して、より効率的なコード・シーケンスを生成できます。	なし	動作不良のアセンブラー・コードは、標準の呼び出し規則に準拠していない可能性があり、浮動小数点レジスターに正しく値を保存できないおそれがあります。AFP (NOVOLATILE) を使用すると、COBOL プログラムはそのようなルーチンを安全に呼び出すことができます。
ARCH (『379 ページの『ARCH』を参照)	実行可能プログラム命令の生成対象となるマシン・アーキテクチャーを指定する	より高い ARCH レベルを指定すると、マシンは、一般的な命令のシーケンスではなく、より新しいより高速な命令を使用するコードを生成します。	なし	アプリケーションは、ARCH オプションに指定されたレベルより低いアーキテクチャー・レベルのプロセッサ上で実行されると、異常終了する可能性があります。
ARITH (EXTEND) (『381 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	なし	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 進データ・タイプの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。
AWO (382 ページの『AWO』を参照)	QSAM ファイルのバッファおよび装置スペースを最適に使用する	このオプションを使用すると、パフォーマンス上の節約になります。これは、入出力を処理するためにデータ管理サービスを呼び出す回数が減るからです。	なし	AWO を使用すると、V モード・レコードを持つプログラム内のすべての QSAM ファイルに対して APPLY WRITE-ONLY 節が有効になります。
BLOCK0 (383 ページの『BLOCK0』を参照)	QSAM 出力ファイルにシステム決定のブロック・サイズを利用する	処理速度を向上させ、QSAM 出力ファイルに関するストレージ要件を最小化できる場合があります。	なし	BLOCK0 を使用すると、ファイル記述項目に BLOCK CONTAINS も RECORDING MODE U も指定していない、プログラム内のすべての QSAM ファイルに対して BLOCK CONTAINS 0 節が活性化されます。
DATA (31) (『391 ページの『データ』を参照)	DFSMS が QSAM バッファを 16 MB 境界より上に割り振るようにします (RENT および DATA (31) コンパイラー・オプションを使用)	拡張形式 QSAM データ・セットは大量のバッファを必要とする可能性があるため、制限のないストレージでバッファを割り振ると、仮想記憶域の制約問題を回避することができます。	なし	DFSMS を使用する z/OS システムでは、アプリケーションがストライプ拡張フォーマット QSAM データ・セットを処理する場合、RENT および DATA (31) の各コンパイラー・オプションを使用して、16 MB 境界より上のストレージから QSAM ファイルの入出力バッファを割り振ります。

表 101. 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
DYNAM (398 ページの『DYNAM』を参照)	サブプログラム (CALL ステートメントを介して呼び出されます) をランタイムに動的にロードします	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。	呼び出しは言語環境ルーチンを介して行う必要があるため、若干パフォーマンスが低下します。	不要になった仮想ストレージを解放するには、CANCEL ステートメントを発行します。
FASTSRT (402 ページの『FASTSRT』を参照)	IBM DFSORT プロダクト (または同等の製品) がすべての入出力を処理することを指定する	各レコードの処理後に Enterprise COBOL に戻るというオーバーヘッドを排除します。	なし	ソート作業ファイルに直接作業ファイルを使用する場合は、FASTSRT が推奨されます。すべてのソートがこのオプションに適格とは限りません。
HGPR (405 ページの『HGPR』を参照)	IBM z/Architecture プロセッサに備わる 64 ビット・レジスターのコンパイラーの使用方法を制御する	HGPR(NOPRESERVE) を指定すると、コンパイラーは、プログラムが使用している 64 ビット GPR の高位半分を保持しないため、パフォーマンスが向上します。	なし	ご使用のプログラムがレジスターの上位半分を変更して保存しない場合に、呼び出し側プログラムが未変更の値に依存していると、アプリケーションから正しくない値が返されるおそれがあります。 除外: これは、このプログラムの呼び出し側が Enterprise COBOL、Enterprise PL/I または z/OS XL C/C++ プログラムである場合には該当しません。
INLINE (407 ページの『INLINE』を参照)	ソース・プログラムで PERFORM ステートメントによって参照されるプロシーチャー (段落またはセクション) のインライン化をコンパイラーが利用するかどうかを制御する	INLINE を指定すると、コンパイラーは、PERFORM ステートメントによって参照されるプロシーチャーをインライン化するかどうかを決定できます。これによって通常は、共通に使用されていて頻繁に実行されるプロシーチャーがアプリケーションに含まれているとパフォーマンスが向上します。	INLINE を指定すると、通常はモジュールのサイズが増加します。頻繁に使用されているがほとんど実行されないプロシーチャーについては、>>INLINE OFF ディレクティブを使用し、コンパイラーがそのプロシーチャーをインライン化しないようにして、モジュールのサイズが増えないようにすることができます。	INLINE コンパイラー・オプションを使用して、プロシーチャーがインライン化に適切であると見なす必要があることを示すことができますが、プロシーチャーを特定の PERFORM ステートメントでインライン化するかどうかの決定は、コンパイラーによって行われます。
INITCHECK (405 ページの『INITCHECK』を参照)	初期設定されずに使用されるデータ項目を検査し、それらが初期化されないまま使用されている場合に警告メッセージを出すよう、コンパイラーに指示します。	なし	INITCHECK オプションを使用すると、コンパイル時間とメモリー消費量が増加する可能性があります。	すべての INITCHECK 分析は、コンパイル時にのみ行われます。INITCHECK オプションは、コンパイル後のプログラムの動作やパフォーマンスには影響しません。
MAXPCF (419 ページの『MAXPCF』を参照)	サイズが大きすぎるため、または複雑すぎるために、余分なコンパイル時間またはストレージ所要量を必要とするプログラムでの最適化を削減する	なし	MAXPCF(n) を指定し、n がゼロでない場合、プログラムの複雑度係数が n を超えると、OPTIMIZE(1) または OPTIMIZE(2) の指定が OPTIMIZE(0) にリセットされ、警告メッセージが生成されます。	なし
NUMCHECK (422 ページの『NUMCHECK』を参照)	データ項目が送信データ項目として使用されている場合に、それらのデータ項目を検証するために追加のコードを生成するよう、コンパイラーに指示します。	なし	NUMCHECK は、COBOL プログラムで使用されるゾーン 10 進数 (数値 USAGE DISPLAY) データ項目、パック 10 進数 (COMP-3) データ項目、および 2 進データ項目の数によっては、NONUMCHECK よりはるかに遅くなります。	なし

表 101. 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
NUMPROC (PFD) (『426 ページの『NUMPROC』を参照)	数値演算の無効な符号処理をバイパスさせる	数値比較に関してはかなり効率のよいコードを生成します。	COMP-3 および DISPLAY 数値データ項目へのほとんどの参照では、NUMPROC (PFD) は、追加のコードが「修正」記号に対して生成されないようにします。この余分のコードは、他のタイプの最適化も妨げる可能性があります。追加のコードは NUMPROC (NOPFD) で生成されます。	NUMPROC (PFD) を使用すると、コンパイラーは、すべての 10 進数項目に優先符号値が含まれていることを前提として要求し、符号「修正」プロセスをバイパスします。ただし、すべての外部データ・ファイルに COMP-3 または DISPLAY 数値データの正しい符号が含まれているわけではなく、プログラムが REDEFINES、グループ移動、または優先符号を保証しない方法で渡されるパラメーターを使用する可能性があるため、NUMPROC (PFD) は多くのプログラムに適していない可能性があります。
OPTIMIZE (0 1 2) (『429 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い。OPTIMIZE (1 2) は、OPTIMIZE (0) よりも多くのコンパイル処理時間を必要とします。	OPTIMIZE (0) は、通常、頻繁なコンパイルが必要なプログラム開発時に使用されます。また、シンボリック・デバッグを可能にします。実稼働環境では、OPTIMIZE (1 2) が推奨されます。
STGOPT (『447 ページの『STGOPT』を参照)	DATA DIVISION におけるストレージ割り振りを最適化します	一般に、ストレージ使用量が少なくなります。	なし	STGOPT は、未使用のデータ項目を削除します。これは、ダンプ読み取りの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。
PARMCHECK (431 ページの『PARMCHECK』を参照)	WORKING-STORAGE にある最後の項目の後に追加のデータ項目を生成するよう、コンパイラーに指示します。次にこのバッファー・データ項目は、呼び出されたサブプログラムが WORKING-STORAGE の端を超えてデータを破壊していないかどうかを検査するために実行時に使用されます。	なし	PARMCHECK を指定すると、コンパイラーは CALL ステートメントを使用するプログラムに対して低速コードを生成します。良好なパフォーマンスを得るには、NOPARMCHECK が有効になっている必要があります。	なし
RENT (435 ページの『RENT』を参照)	再入可能プログラムを生成する	プログラムを共用ストレージ (LPA/ELPA) に入れ、実行をより高速化することができます。	プログラムを再入可能にするために追加のコードが生成されます。	NONE
RMODE (ANY) (『436 ページの『RMODE』を参照)	プログラムをどこにでもロードできるようにする	なし	なし	NONE
SMARTBIN (440 ページの『SMARTBIN』を参照)	IBM Automatic Binary Optimizer (ABO) for z/OS (ABO) 2.2 による最適化を可能にする追加のバイナリー・メタデータを含むモジュールを生成するように、コンパイラーに指示する	COBOL モジュールを将来の ABO 最適化の対象として、再コンパイルとテストを必要とせずに IBM Z ハードウェア機能拡張のメリットを最大限に活用できるようにします	このオプションを使用すると、ディスク上のコンパイル時間とモジュール・サイズが増え、より大きなロード・モジュールが必要になります。メタデータは NOLOAD セグメントで作成されるため、メモリー内のサイズには影響しません。	なし

表 101. 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
SSRANGE (『446 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にあるかどうかを検査する	SSRANGE は、表参照を検査するための追加コードを生成します。NOSSRANGE を使用すると、そのコードは生成されません。	SSRANGE を指定すると、有効な範囲の検査がコンパイラーのパフォーマンスに影響を与えます。	一般に、すべての参照ではなく、表の参照を数回だけ検査する必要がある場合は、SSRANGE 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。パフォーマンスに依存するアプリケーションの場合は、NOSSRANGE が推奨されます。
TEST (『448 ページの『TEST』を参照)	デバッグ・ツールの使用時に完全なデバッグ機能を取得し、CEEDUMP 内のデータ項目のシンボリック・ダンプを取得する。NOTEST (DWARF) を使用して、CEEDUMP 内のデータ項目のシンボリック・ダンプを取得することもできます。	なし	TEST オプションを使用すると、最適化がある程度削減されます。TEST の EJPD サブオプションを使用すると、最適化がさらに削減されます。	実稼働環境では、NOTEST または TEST (NOEJPD) を使用することが推奨されます。 実稼働実行中に、プログラムが異常終了した場合に、定様式ダンプ内のデータ項目のシンボリック・ダンプが必要な場合は、TEST を使用して、または NOTEST (DWARF) を使用してコンパイルします。
THREAD (453 ページの『THREAD』を参照)	複数の POSIX スレッドまたは PL/I タスクを持つ言語環境・エンクレープでプログラムを実行できるようにする。	なし	シリアライゼーション・ロジックのオーバーヘッドがあるため、ちょっとしたパフォーマンス・ペナルティがあります。	スレッド化環境または非スレッド化環境のいずれかで多少のパフォーマンス・ペナルティが発生します。
TRUNC (OPT) (『454 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) の両方とも、BINARY データ項目が変更されるたびに追加のコードを生成します。通常、TRUNC (BIN) はこれらのオプションの中で最も遅いものです。	TRUNC (STD) は 85 COBOL 標準に準拠していますが、TRUNC (BIN) および TRUNC (OPT) は準拠していません。TRUNC (OPT) では、コンパイラーは、データが PICTURE および USAGE 仕様に準拠していると想定します。可能な場合は、TRUNC (OPT) が推奨されます。
TUNE (456 ページの『TUNE』を参照)	実行可能プログラムの最適化の対象のアーキテクチャーを指定する	ハードウェア・アーキテクチャーの具体的な実装方式に合わせて、命令の選択や、実装方式に依存するその他のパフォーマンス向上のためのチューニングを行います。	なし	指定した TUNE レベルが、指定した ARCH レベルより低い場合は、コンパイラーによって、ARCH レベルと一致するように TUNE レベルが調整されます。

関連概念

843 ページの『最適化』

40 ページの『ストレージとそのアドレス可能性』

関連タスク

307 ページの『コンパイラー・メッセージのリストの生成』

849 ページの『パフォーマンスの評価』

11 ページの『バッファーおよび装置スペースの最適化』

844 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』

255 ページの『FASTSORT によるソート・パフォーマンスの向上』

197 ページの『ストライプ拡張形式 QSAM データ・セットの使用』

841 ページの『テーブルの効率的処理』

関連参照

59 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』

197 ページの『QSAM ファイル用のバッファーの割り振り』

371 ページの『COBOL 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。

このタスクについて

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 102. パフォーマンス・チューニング・ワークシート	
コンパイラー・オプション	考慮事項
ARCH	プログラムが実行されるすべての環境で、可能な限り高いアーキテクチャー・レベルを使用していますか。例えば、災害復旧マシンを含め、使用する最低レベルのアーキテクチャーが zEC12 または zBC12 の場合、ARCH(10) を使用していますか。
AWO	可能な場合は AWO オプションを使用しますか。
BLOCK0	QSAM ファイルに BLOCK0 を使用しますか。
DATA	QSAM ストラープ・データ・セットを使用する場合、RENT オプションと DATA(31) オプションを使用しますか。プログラム・オブジェクトは AMODE 31 ですか。ALL31(ON) で実行していますか。
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。
FASTSRT	ソート作業ファイルとして直接作業ファイルを使用する場合、FASTSRT オプションを使用しましたか。
INLINE	可能な場合は INLINE を使用していますか。
INVDATA	可能な場合は INVDATA を使用していますか。
NUMCHECK	NONUMCHECK を実稼働に使用していますか。
NUMPROC	可能な場合は NUMPROC (PFD) を使用していますか。
OPTIMIZE	実稼働にゼロ以外の OPTIMIZE レベルを使用しますか。
PARMCHECK	NOPARMCHECK を実稼働に使用していますか。
SSRANGE	NOSSRANGE を実稼働に使用していますか。
TEST	NOTEST または TEST (NOEJPD) を実稼働に使用しますか。
TRUNC	可能な場合は TRUNC (OPT) を使用していますか。
TUNE	プログラムが最も頻繁に実行される環境では、可能な限り高いアーキテクチャー・レベルを使用していますか。例えば、実稼働環境のマシンが zEC12 または zBC12 の場合、TUNE(10) を使用していますか。

関連概念

40 ページの『ストレージとそのアドレス可能性』

関連タスク

[844 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

[845 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』](#)

CICS、IMS、または VSAM での効率的な実行

以下のヒントに従うなら、CICS または IMS のもとで実行されるオンライン・プログラム、あるいは VSAM を使用するプログラムのパフォーマンスを向上させることができます。

このタスクについて

CICS: アプリケーションが CICS のもとで実行される場合、EXEC CICS LINK コマンドを CALL ステートメントに変換してトランザクション応答時間を改善します。

IMS: アプリケーションが IMS のもとで実行される場合は、アプリケーション・プログラムとライブラリー・ルーチンをプリロードすれば、ロードおよび検索のオーバーヘッドを削減するのに役立ちます。入出力活動も減少する可能性があります。

システムのパフォーマンスを向上させるためには、RENT コンパイラー・オプションを使用して、可能な限りアプリケーションとライブラリー・ルーチンをプリロードします。また、言語環境のライブラリー・ルーチン保存 (LRR) 機能を使用すると、IMS/TM 領域でのパフォーマンスを向上することができます。

VSAM: VSAM ファイルを使用する場合、順次アクセスの場合にはデータ・バッファの数、ランダム・アクセスの場合には索引バッファの数を増やしてください。また、アプリケーションに適した制御インターバル・サイズ (CISZ) を選択してください。CISZ が小さいと、ランダム処理での検索は速くなりますが、挿入が犠牲になります。CISZ が大きいと、順次処理の場合に効率がよくなります。

パフォーマンスを向上させるためには、レコードを順次アクセスし、複数の代替索引の使用を可能な限り避けます。代替索引を使用する場合、アクセス方式サービス・プログラムにより、AIXBLD ランタイム・オプションよりもさらに効率的に索引が作成されます。

関連タスク

[545 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』](#)

[569 ページの『第 25 章 COBOL プログラムの開発 \(IMS の場合\)』](#)

[225 ページの『VSAM パフォーマンスの向上』](#)

言語環境のカスタマイズ

関連参照

言語環境プログラミングガイド (ランタイム・オプションの指定)

静的呼び出しまたは動的呼び出しの選択

ご使用のモジュールが配置可能で、頻繁に相互を呼び出すプログラムが 1 つのモジュール内にある場合は、静的呼び出しのほうが動的呼び出しよりも高速です。一方、動的呼び出しでは、複数のサブプログラム・ユーザーを再バインド (再リンク・エディット) する必要があるため、アプリケーションの更新をさらに容易に行うことができます。

このタスクについて

詳しくは、[611 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』](#)を参照してください。

関連概念

[611 ページの『静的呼び出しと動的呼び出しのパフォーマンスについての考慮事項』](#)

第 51 章 コーディングの単純化

このタスクについて

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、形式 2 SORT ステートメント、COBOL 組み込み関数、言語環境 呼び出し可能サービス COBOL/Java 通信機能、AMODE 31 (31 ビット) と AMODE 64 (64 ビット) の COBOL プログラム間の呼び出し、およびユーザー定義関数を使用することにより、反復コーディングを回避でき、多数の算術計算やその他の複雑なタスクをコーディングする必要がなくなります。さらに、SSRANGE、NUMCHECK、PARMCHECK、および INITCHECK などのコンパイラー・オプションは、無効データの問題を検出するのに役立つため、COBOL 4 以前から COBOL 6 以降へのマイグレーション作業が容易になります。

COPY ステートメント

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・シーケンスを一度作成し、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組み込むことができます。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。詳細については、852 ページの『[反復コーディングの除去](#)』を参照してください。

形式 2 SORT ステートメント

テーブルをソートする場合、形式 2 SORT ステートメントを使用するとコーディングを簡素化することができます。これは、形式 1 の SORT ステートメントと比較して、はるかに単純な方法です。詳しくは、857 ページの『[形式 2 SORT ステートメントを使用したテーブルのソート](#)』を参照してください。

組み込み関数

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。詳しくは、122 ページの『[データ項目の変換 \(組み込み関数\)](#)』および 126 ページの『[データ項目の評価 \(組み込み関数\)](#)』を参照してください。

言語環境 呼び出し可能サービス

言語環境日時の呼び出し可能サービスは、日付をフルワード 2 進整数として保管し、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。詳しくは、853 ページの『[言語環境呼び出し可能サービスの使用](#)』を参照してください。

COBOL/Java のインターオペラビリティ

COBOL プログラムは、オブジェクト指向プログラミング・フレームワークを使用せずに Java プログラムとのインターオペラビリティを実現 (相互運用) できます。そのため、Java を使用して COBOL アプリケーションを簡単に拡張できます。詳しくは、331 ページの『[第 18 章 Java と相互運用する非 OO COBOL アプリケーションのコンパイル、リンク、および実行](#)』および 805 ページの『[第 7 部 オブジェクト指向 \(OO\) COBOL フレームワークの外部での COBOL/Java 相互運用性](#)』を参照してください。

AMODE 31 (31 ビット) および AMODE 64 (64 ビット) の COBOL プログラムのインターオペラビリティ

既存の AMODE 31 COBOL アプリケーションとのインターオペラビリティを実現 (相互運用) できる AMODE 64 COBOL アプリケーションを作成できます。AMODE 64 COBOL アプリケーションは、プログラム論理を変更しなくても、既存の AMODE 31 のデータ・サイズ制限より大きなデータ項目にアクセスできます。詳しくは、596 ページの『[AMODE 31 および AMODE 64 の混合プログラムを使用する COBOL アプリケーション](#)』を参照してください。

ユーザー定義関数

多くの一般的なプログラミング言語と同様に、COBOL はユーザー定義関数をサポートしており、使い慣れた構造を利用できます。ユーザー定義関数の定義を使用して独自の関数を作成し、それらを組み込み関数

のように呼び出せるので、コードのモジュール性と保守容易性が向上します。詳しくは、*Enterprise COBOL for z/OS* 言語解説書 および 659 ページの『第 33 章 ユーザー定義関数の使用』の『関数 IDENTIFICATION DIVISION (Function identification division)』を参照してください。

COBOL マイグレーション中に無効データを検出するためのコンパイラー・オプション

Enterprise COBOL 6 へのマイグレーション時の最も一般的な問題の 1 つは、ソース・コードを検査しても検出できない無効な COBOL データによって引き起こされます。これらの問題をより簡単に識別する際に、2 ステップのコンパイル・プロセスの使用をご検討ください。このプロセスでは、初期コード変更と単体テストのために SSRANGE、NUMCHECK、PARMCHECK、INITCHECK、および OPT(0) の各コンパイラー・オプションを使用し、その後、品質保証テストと実動のために NOSSRANGE、NONUMCHECK、NOPARMCHECK、および OPT(2) を指定して再コンパイルします。詳しくは、*Enterprise COBOL* 移行ガイドの『Enterprise COBOL でのコンパイルに関する FAQ』を参照してください。

詳しい COBOL マイグレーション関連情報については、「[COBOL マイグレーション・ポータル](#)」にアクセスしてください。ここには、お客様事例、COBOL 専門家のインタビュー動画、クラウド・ベースの COBOL マイグレーション・アシスタント (マイグレーション・プロセスのナビゲーション用)、COBOL マイグレーションとパフォーマンス・チューニングの Web セミナー、FAQ、マイグレーションをサポートするその他の IBM 製品、他の多くのリソースが含まれます。

反復コーディングの除去

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には、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・レベルで使用します。COPY ステートメントは任意の深さにネストできます。

このタスクについて

複数のコピー・ライブラリーを使用するには、複数のシステム定義を使用するか、複数の定義と IN/OF 句 (IN/OF *library-name*) の組み合わせを使用します。

MVS バッチ

JCL を使用して、SYSLIB DD ステートメント内のデータ・セットを連結します。あるいは、複数の DD ステートメントを定義し、COPY ステートメントの IN/OF 句を使用します。

tso

ALLOCATE コマンドを使用して、SYSLIB のデータ・セットを連結します。あるいは、複数の ALLOCATE ステートメントを発行し、COPY ステートメントの IN/OF 句を使用します。

z/OS ユニックス

SYSLIB 環境変数を使用して、コピーブックへの複数のパスを定義します。あるいは、複数の環境変数を使用し、COPY ステートメントの IN/OF 句を使用します。

例:

```
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。

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。あ COPY 文自体はデバッグ行にはなり得ません。D、そして WITH DEBUGGING モードが指定されていない場合、COPY ただし、ステートメントは処理されます。

853 ページの『[例: COPY ステートメントの使用](#)』

関連参照

481 ページの『[第 21 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・リストでは、COPY ステートメントは別の行に印刷されます。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 プロシーチ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。

注: コンパイル時間を短縮するには、関連項目をコピーブック内にグループ化します。ただし、関連のない項目を 1 つの大きなコピーブックに入れる必要はありません。

関連タスク

[852 ページの『反復コーディングの除去』](#)

関連参照

[481 ページの『第 21 章 コンパイラー指示ステートメント』](#)

言語環境呼び出し可能サービスの使用

言語環境 呼び出し可能サービスにより、数多くのタイプのプログラミング作業が容易になります。CALL ステートメントを使用して呼び出します。

このタスクについて

言語環境は、以下のタスクに役立てることができます。

- 条件の処理

言語環境の条件処理機能により、COBOL アプリケーションは予期しないエラーに反応できるようになります。言語構造体またはランタイム・オプションを使用して、それぞれの条件を処理するレベルを選択できます。例えば、COBOL プログラム内の特定のエラーを処理し、それを言語環境に処理させるか、またはオペレーティング・システムに処理させることができます。

言語環境の条件処理のサポートでは、COBOL はプロシージャ・ポインター・データ項目を提供します。

- 動的ストレージの管理

これらのサービスにより、ストレージの取得、解放、および再割り振りを行うことができます。また、自分自身のストレージ・プールも作成できます。

- 日時の計算

日時サービスを使用すると、いくつかの形式で現在のローカルの日時を取得でき、日時の変換を実行できます。2つの呼び出し可能サービス CEEQCEN および CEESCEN により、2 桁年 (1991 を表す 91 または 2009 を表す 09 など) を処理する予測可能な方法が提供されます。

- 数学計算

数学呼び出し可能サービスで容易に実行される計算としては、対数関数、指数関数、三角関数、平方根関数、および整数関数があります。

COBOL では組み込み関数もサポートされます。この組み込み関数の中には、呼び出し可能サービスによって提供される数学関数および日付関数と同じ働きをする関数もいくつかあります。言語環境呼び出し可能サービスと組み込み関数は、多少の例外はありますが、同じ結果を戻します。これらの相違点を理解したうえで、どちらを使用するかを決定してください。

- メッセージの処理

メッセージ処理サービスには、メッセージの取得、ディスパッチング、およびフォーマット設定のサービスがあります。非 CICS アプリケーション用のメッセージは、ファイルまたはプリンターに送信できます。CICS メッセージは、CICS 一時データ・キューに送信されます。言語環境は、メッセージを分割して、宛先のレコード長を収容できるようにし、日本語や英語のような正しい各国語でメッセージを表示します。

- 各国語のサポート

これらのサービスにより、アプリケーションは、アプリケーション・ユーザーが必要とする言語をサポートしやすくなります。言語と国を設定すると、デフォルトの日付、時刻、番号、および通貨形式を入手することができます。例えば、日付を「23 June 09」または「6,23,09」のように表示することができます。

- z/OS デバッガの始動や言語環境の定様式ダンプの取得などの一般的なサービス

z/OS デバッガは CICS プログラムのバッチ式デバッグおよび対話式デバッグの両方をはじめ、COBOL アプリケーションに高度なデバッグ機能を提供します。z/OS デバッガホストから COBOL アプリケーションをデバッグしたり、IBM 開発者 z/OS、ワークステーションから。

選択したオプションに応じて、言語環境の定様式ダンプには、データ項目の名前と値のほか、状態、プログラム・トレースバック、制御ブロック、ストレージ、およびファイルに関する情報が入れられます。すべての言語環境ダンプには、共通の十分にラベル付けされた読みやすいフォーマットがあります。

[856 ページの『例: 言語環境の呼び出し可能サービス』](#)

関連概念

[855 ページの『言語環境の呼び出し可能サービスのサンプル・リスト』](#)

関連タスク

[63 ページの『数字組み込み関数の使用』](#)

[64 ページの『数学用の呼び出し可能サービスの使用』](#)

[66 ページの『データ呼び出し可能サービスの使用』](#)

言語環境の呼び出し可能サービスのサンプル・リスト

次の表は、言語環境で使用可能な呼び出し可能サービスの例を示しています。リストされているサービスよりもっと多くのサービスを使用できます。

表 103. 言語環境 呼び出し可能サービス		
関数のタイプ	サービス	目的
条件処理	CEEHDLR	ユーザー条件処理ルーチンを登録する
	CEESGL	条件を発生させる、または通知する
	CEEMRCR	条件処理ルーチンが終了した後でプログラムが実行を再開する場所を指示する
動的ストレージ	CEEGTST	ストレージを獲得する
	CEECZST	これまでに割り振られたストレージ・ブロックのサイズを変更する
	CEEFRST	ストレージを解放する
日時	CEECBLDY	日付を表すストリングを COBOL 整数日付形式に変換する (日付を 1600 年 12 月 31 日以降の日数として表す)
	CEEQCEN、 CEESCEN	言語環境の世紀ウィンドウ (年を表すのに 2 桁を使用するプログラムの場合に有用) を照会および設定する
	CEEGMTO	ローカル・システム時間とグリニッジ標準時の差を計算する
	CEELOCT	3 つの形式を選択して現在の地方時を入手する
数学	CEESIABS	整数の絶対値を計算する
	CEESSNWN	単精度浮動小数点数に最も近い整数を計算する
	CEESSCOS	角の余弦を計算する
メッセージ処理	CEEMOUT	メッセージをディスパッチする
	CEEMGET	メッセージを検索する
各国語サポート	CEE3LNG	現在の各国語を変更または照会する
	CEE3CTY	現在の国を変更または照会する
	CEE3MCS	特定の国のデフォルトの通貨記号を入手する
一般	CEE3DMP	言語環境定様式ダンプを取得する
	CEETEST	デバッグ・ツールなどのデバッグ・ツールを開始する

関連参照

言語環境プログラミング・リファレンス

言語環境・サービスの呼び出し

言語環境サービスを呼び出すには、そ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。CALL ステートメントの変数は、そのサービスに必要な定義を指定して、DATA DIVISION に定義します。

このタスクについて

```
77 argument      comp-1.
77 feedback-code pic x(12) display.
77 result        comp-1.

...
CALL "CEESSQT" using argument, feedback-code, result
```

上記の例では、言語環境サービス CEESSQT は変数の平方根の値を計算します `argument` この値を変数に返す `result`。

フィードバック・コード・パラメーターを指定するかどうかを選択できます。指定した場合、`feedback-code` に戻される値は、サービスが正しく完了されたかどうかを示します。指定する場合 `OMITTED` フィードバックコードの代わりに、サービスが成功しない場合は、言語環境状態は自動的に言語環境コンディショナマネージャー。このような条件は、ユーザー作成条件処理ルーチンに実装されたリカバリー・ロジックによって処理するか、処理されない条件に対してデフォルト言語環境処理を行うことができます。いずれの場合も、ロジックを作成して呼び出しを終えるたびにフィードバック・コードをチェックする必要がなくなります。

言語環境呼び出し可能サービス呼び出し、`OMITTED` をフィードバック・コードに指定すると、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。サービスが成功しなかった場合、このレジスターは変更されません。フィードバック・コードに `OMITTED` を指定しない場合は、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。

856 ページの『例: 言語環境の呼び出し可能サービス』

関連概念

言語環境プログラミングガイド (一般的な呼び出し可能サービス)

関連参照

言語環境プログラミング・リファレンス (一般的な呼び出し可能サービス)

CALL ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

例: 言語環境の呼び出し可能サービス

この例では、言語環境 COBOL の結果から日付をフォーマットして表示するサービス `CEEDAYS` と `CEEDATEACCEPT` 声明。

`CEEDAYS` および `CEEDATE` を使用すると、言語環境がないと必要になるようなコーディングが少なくて済みます。

```
ID DIVISION.
PROGRAM-ID. HOHOHO.
*****
* FUNCTION:  DISPLAY TODAY'S DATE IN THE FOLLOWING FORMAT: *
*           WWWWWWWWW, MMMMMMM DD, YYYY                  *
*           For example: TUESDAY, SEPTEMBER 15, 2009      *
*           *****                                     *
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 LANGUAGE ENVIRONMENT 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 " WWWWWWWWWZ, MMMMMMMMMZ 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.

```

形式 2 SORT ステートメントを使用したテーブルのソート

テーブルをソートするには、形式 2 SORT ステートメントの使用をお勧めします。この方法は、形式 1 SORT ステートメントに比べて以下の利点があります。

表 104. 形式 1 と形式 2 の SORT ステートメントの比較		
特性	形式 1 SORT ステートメント	形式 2 SORT ステートメント
ファイルまたはテーブルのソートに使用できる	はい	いいえ、テーブル専用です。
DFSORT または同等のソート・プログラムが必要	はい	いいえ
CICS でサポートされる	制限付き	はい
UNIX システム・サービスでサポートされる	いいえ	はい
THREAD オプションを使用してコンパイルされたプログラムでサポートされる	いいえ	はい
1 つの SORT ステートメントを使用してテーブルをソートできる (コーディングが簡単)	いいえ、SELECT 節、レコード記述のある SD 項目、および入出力プロシージャが必要です。	はい
ソートのキーはテーブル定義の一部として指定することができ、SEARCH ALL 声明	いいえ、キーは SORT ステートメントで指定しなければなりません。SEARCH ALL を使用したテーブル検索も行う場合は、テーブル定義の一部としてキーを重複して指定することも必要です。	はい、必要であれば SORT ステートメントでキーを指定することもできます。
ソート処理中にテーブル・エレメントをフィルターに掛けるか、前処理を行うことができる	はい、入出力プロシージャを使用します。	いいえ、テーブル・エレメントはすべてそのまま SORT に渡されます。
SORT-CONTROL、SORT-CORE-SIZE、SORT-FILE-SIZE、SORT-MESSAGE、SORT-MODE-SIZE、SORT-RETURN などの特殊レジスターを使用する	はい	いいえ
入出力プロシージャの範囲内で実行できる	いいえ	はい

注: ストレージが制約されている環境で形式 2 SORT を大きなテーブルとともに使用しないでください。
形式 2 SORT ではソートの実行にヒープ・ストレージが使用されるためです。

関連参照

SORT ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

付録 A 付録

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。
- ROUNDED 句を含むステートメント

860 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、COBOL for OS/390 & VM 2.2 より前の IBM COBOL のリリース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

[53 ページの『数値データの形式』](#)

[68 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

[861 ページの『固定小数点データと中間結果』](#)

[866 ページの『浮動小数点データと中間結果』](#)

[867 ページの『非算術ステートメントの算術式』](#)

[381 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、以下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 (*Enterprise COBOL for z/OS* 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

62 ページの『算術式の使用』

関連参照

860 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。

操作	整数桁	小数桁
+ または -	$(i1 \text{ または } i2) + 1$ 、どちらか大きい方	$d1$ または $d2$ 、どちらか大きい方
*	$i1 + i2$	$d1 + d2$
/	$i2 + d1$	$(d2 - d1)$ または $dmax$ 、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、コンパイラー・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
<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

$i + d$ の値	d の値	$i + dmax$ の値	ir 用に保持される桁数
>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、コンパイラー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、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	任意の	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	$op1$ 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	任意の	$op2$ 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	任意の	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	任意の	$op2$ 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 ($|op2| - 1$) 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。 Division-by-0 および累乗 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。

[863 ページの『例: 固定小数点の算術での指数』](#)

関連参照

[860 ページの『中間結果用の用語』](#)

[863 ページの『中間結果での切り捨て』](#)

[864 ページの『バイナリー・データと中間結果』](#)

[866 ページの『浮動小数点データと中間結果』](#)

[864 ページの『固定小数点算術で評価される組み込み関数』](#)

[381 ページの『ARITH』](#)

SIZE ERROR 句 (*Enterprise COBOL for z/OS* 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、内部中間結果 *iir1* が得られる。
2. *iir1* に A を掛けると、内部中間結果 *iir2* が得られる。
3. *iir2* に A を掛けると、内部中間結果 *iir3* が得られる。
4. *iir3* を *ir4* に移動する。

ir4 は *dmax* の小数桁数を持っています。 B は正であるため、*ir4* は Y に移動されます。 ただし、B が -4 であった場合は、5 番目の追加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。

注: 上のステップ 1、2、および 3 で指数計算を実行することによって内部ライブラリー・ルーチンで取得された内部中間結果 (*iir1*、*iir2*、および *iir3*) では、上の *ir4* および *ir5* と同じ 10 進精度は使用されません。 その代わり、これらの中間結果はより正確です。 考えられる最も正確な結果は *ir4* または *ir5* で得られます。

関連参照

[860 ページの『中間結果用の用語』](#)

[861 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

53 ページの『数値データの形式』

関連参照

861 ページの『固定小数点データと中間結果』

381 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

861 ページの『固定小数点データと中間結果』

381 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-INTEG	0	8
DATE-TO-YYYYMMDD	0	8
DAY-OF-INTEG	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	該当なし	9
MOD	0	min(<i>i1 i2</i>)
ORD	該当なし	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、以下の手順を実行する。
 - a. 関数結果の代数值を引数と比較する。
 - 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、以下の手順を実行する。
 - a. 関数結果の代数值を引数と比較する。
 - 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、以下の手順を実行する。
 - 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

[860 ページの『中間結果用の用語』](#)

[861 ページの『固定小数点データと中間結果』](#)

[866 ページの『浮動小数点データと中間結果』](#)

[381 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。
- 指数に小数位が含まれている。
- 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

[860 ページの『中間結果用の用語』](#)

[381 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。
 - 指数に小数位が含まれている。
 - 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。

例:

```
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比較で関係演算子を使用されない場合 (つまり、明示的な比較条件がない場合)、その算術式は、被比較数の属性とは無関係に評価されます。例:

```
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

[68 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

[860 ページの『中間結果用の用語』](#)

[861 ページの『固定小数点データと中間結果』](#)

[866 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 (*Enterprise COBOL for z/OS* 言語解説書)

EVALUATE ステートメント (*Enterprise COBOL for z/OS* 言語解説書)

条件式 (*Enterprise COBOL for z/OS* 言語解説書)

2 バイト文字セット (DBCS) データの変換

言語環境のサービス・ルーチン IGZCA2D および IGZCD2A は、DBCS データを含んでいる英数字データ項目を、純 DBCS データ項目との間で変換するためのものでしたが、これには STRING、UNSTRING、および参照変更などの操作を確実に実行するという目的がありました。

このタスクについて

互換性を保たせるため、これらのサービス・ルーチンは引き続き提供されます。しかし、互換性の目的の場合は、現在、代わりに国別データ項目および国別変換操作を使用することをお勧めしています。

これらのサービス・ルーチンは、コード・ページ引数をサポートしないので、CODEPAGE コンパイラ・オプションで指定されたコード・ページの影響を受けません。DBCS コンパイラ・オプションは操作に影響を与えません。

関連タスク

[142 ページの『国別 \(Unicode\) 表現との間の変換』](#)

[170 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

[868 ページの『DBCS の表記』](#)

[868 ページの『英数字から DBCS データへの変換 \(IGZCA2D\)』](#)

[870 ページの『DBCS から英数字データへの変換 \(IGZCD2A\)』](#)

[385 ページの『CODEPAGE』](#)

DBCS の表記

DBCS データ変換例では、DBCS 項目を記述するために以下の記号が使用されています。

記号	意味
<および>	それぞれ、シフトアウト (SO) およびシフトイン (SI)
D0, D1, D2, ..., Dn	1 バイト EBCDIC 文字に対応する、2 バイト EBCDIC 文字を除く任意の DBCS 文字
.A、.B、.C、...	1 バイト EBCDIC 文字に対応する、任意の 2 バイトの EBCDIC 文字。ピリオド (.) は値 X'42' を表します。
A、B、または s などの単一の文字	任意の 1 バイトの EBCDIC 文字

英数字から DBCS データへの変換 (IGZCA2D)

言語環境の IGZCA2D サービス・ルーチンは、2 バイト文字を含んでいる英数字データを純 DBCS データに変換します。

IGZCA2D の構文

IGZCA2D サービス・ルーチンを使用するには、CALL ステートメントを使用して次の 4 つのパラメーターをこのルーチンに渡します。

parameter-1

変換の送信フィールドで、英数字データ項目として処理されます。

parameter-2

変換の受信フィールドで、DBCS データ項目として処理されます。

parameter-2 で参照変更を使用することはできません。

parameter-3

変換される *parameter-1* の中のバイトの数。

これは、*parameter-1* の LENGTH OF 特殊レジスターにするか、または変換される *parameter-1* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。シフト・コードはそれぞれ 1 バイトとしてカウントされます。

parameter-4

変換されたデータを受け取る *parameter-2* の中のバイトの数。

これは、*parameter-2* の LENGTH OF 特殊レジスターにするか、または変換されるデータを受け取る *parameter-2* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。

使用上の注意

- 合格できる *parameter-1*、*parameter-3*、そして *parameter-4* ルーチンに BY REFERENCE または BY CONTENT、しかし、合格しなければなりません *parameter-2* BY REFERENCE。
- コンパイラーは、これらのパラメーターの構文検査を行いません。パラメーターを正しく設定し、CALL ステートメントを使用して、変換ルーチンに渡すようにしてください。そうでない場合、結果は予測できません。
- *parameter-1* から *parameter-2* を作成する際、IGZCA2D は以下の変更を行います。
 - DBCS データはそのまま変更しないで、シフト・コードを除去します。
 - 単一バイト (非スペース) EBCDIC 文字 X'*nn*' を X'42*nn*' で表す文字に変換します。
 - 単一バイト・スペース (X'40') を、X'4240' ではなく DBCS スペース (X'4040') に変換します。
- IGZCA2D は、*parameter-1*、*parameter-3*、または *parameter-4* の内容を変更しません。
- *parameter-3* の内容および *parameter-4* の内容の有効範囲は、1 から 134,217,727 です。

870 ページの『例: IGZCA2D』

関連参照

869 ページの『IGZCA2D の戻りコード』

IGZCA2D の戻りコード

IGZCA2D は、変換の状況を反映するように RETURN-CODE 特殊レジスターを設定します。

表 105. IGZCA2D 戻りコード	
戻りコード	解説
0	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。
2	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。 <i>parameter-2</i> は、右側に DBCS スペースが埋め込まれました。
4	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。 <i>parameter-2</i> に入れられた DBCS データは、右側が切り捨てられました。
6	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。 X'00' から X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。 有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。
8	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。 X'00' から X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。 有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。 <i>parameter-2</i> は、右側に DBCS スペースが埋め込まれました。
10	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。 X'00' から X'3F' の範囲または X'FF' の 1 バイト文字が検出されました。 有効な 1 バイト文字が範囲外の DBCS 文字に変換されました。 <i>parameter-2</i> の中の DBCS データは、右側が切り捨てられました。
12	<i>parameter-1</i> の中の対になったシフト・コードの間で奇数のバイトが検出されました。変換は行われませんでした。
※13	<i>parameter-1</i> の中で、対になっていないかまたはネストされたシフト・コードが検出されました。変換は行われませんでした。
14	<i>parameter-1</i> と <i>parameter-2</i> がオーバーラップしました。変換は行われませんでした。
15	<i>parameter-3</i> または <i>parameter-4</i> に与えられた値が範囲外でした。変換は行われませんでした。

表 105. IGZCA2D 戻りコード (続き)	
戻りコード	解説
16	<i>parameter-4</i> の中で奇数のバイトがコーディングされました。変換は行われませんでした。

例: IGZCA2D

この CALL ステートメント例は、alpha-item 内の英数字データを DBCS データに変換します。変換の結果は dbcs-item に入れられます。

```
CALL "IGZCA2D" USING BY REFERENCE alpha-item dbcs-item
                     BY CONTENT LENGTH OF alpha-item LENGTH OF dbcs-item
```

変換前の alpha-item および dbcs-item の内容と長さが次のものであるとします。

```
alpha-item = AB<D1D2D3>CD
dbcs-item  = D4D5D6D7D8D9D0
```

```
LENGTH OF alpha-item = 12
LENGTH OF dbcs-item  = 14
```

変換後、alpha-item および dbcs-item には次のものが入ります。

```
alpha-item = AB<D1D2D3>CD
dbcs-item  = .A.BD1D2D3.C.D
```

RETURN-CODE レジスターの内容は 0 です。

関連参照

[868 ページの『DBCS の表記』](#)

DBCS から英数字データへの変換 (IGZCD2A)

言語環境の IGZCD2A ルーチンは、純 DBCS データを、2 バイト文字を含むことのある英数字データに変換します。

IGZCD2A の構文

IGZCD2A サービス・ルーチンを使用するには、CALL ステートメントを使用して次の 4 つのパラメーターをこのルーチンに渡します。

parameter-1

変換の送信フィールドで、DBCS データ項目として処理されます。

parameter-2

変換の受信フィールドで、英数字データ項目として処理されます。

parameter-3

変換される *parameter-1* の中のバイトの数。

これは、*parameter-1* の LENGTH OF 特殊レジスターにするか、または変換される *parameter-1* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。

parameter-4

変換されたデータを受け取る *parameter-2* の中のバイトの数。

これは、*parameter-2* の LENGTH OF 特殊レジスターにするか、または変換されるデータを受け取る *parameter-2* のバイト数を含む 4 バイトの USAGE IS BINARY データ項目にすることができます。シフト・コードはそれぞれ 1 バイトとしてカウントされます。

使用上の注意

- 合格できる *parameter-1*、*parameter-3*、そして *parameter-4* ルーチンに BY REFERENCE または BY CONTENT、しかし、合格しなければなりません *parameter-2* BY REFERENCE。
- コンパイラーは、これらのパラメーターの構文検査を行いません。パラメーターを正しく設定し、変換ルーチンに渡すようにしてください。そうでない場合、結果は予測できません。
- *parameter-1* から *parameter-2* を作成する際、IGZCD2A は以下の変更を行います。
 - 1 バイト EBCDIC 文字に対応しない DBCS 文字の前後にシフト・コードを挿入します。
 - DBCS 文字が 1 バイト EBCDIC 文字に対応する場合は、DBCS 文字を 1 バイト文字に変換します。
 - DBCS スペース (X'4040') を単一バイト・スペース (X'40') に変換します。
- IGZCD2A は、*parameter-1*、*parameter-3*、または *parameter-4* の内容を変更しません。
- 変換されたデータに 2 バイト文字が含まれている場合、シフト・コードは *parameter-2* の長さに含まれます。
- *parameter-3* の内容および *parameter-4* の内容の有効範囲は、1 から 134,217,727 です。

871 ページの『例: IGZCD2A』

関連参照

871 ページの『IGZCD2A の戻りコード』

IGZCD2A の戻りコード

IGZCD2A は、変換の状況を反映するように RETURN-CODE 特殊レジスターを設定します。

表 106. IGZCD2A の戻りコード	
戻りコード	解説
0	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。
2	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。 <i>parameter-2</i> は、右側に 1 バイト・スペースが埋め込まれました。
4	<i>parameter-1</i> が変換され、その結果が <i>parameter-2</i> に入れられました。 <i>parameter-2</i> は、右側が切り捨てられました。 ¹
14	<i>parameter-1</i> と <i>parameter-2</i> がオーバーラップしました。変換は行われませんでした。
15	<i>parameter-3</i> または <i>parameter-4</i> の値が範囲外でした。変換は行われませんでした。
16	<i>parameter-3</i> の中で奇数のバイトがコーディングされました。変換は行われませんでした。
1. 切り捨てが DBCS 文字の中で行われる場合、その切り捨ては偶数バイト境界で行われ、シフトイン (SI) が挿入されます。必要な場合には、シフトインの後で、英数字データに単一バイト・スペースが埋め込まれます。	

例: IGZCD2A

この CALL ステートメント例は、dbcs-item 内の DBCS データを 2 バイト文字を含む英数字データに変換します。変換の結果は alpha-item に入れられます。


```
CALL "IGZCD2A" USING BY REFERENCE dbcs-item alpha-item  
BY CONTENT LENGTH OF dbcs-item LENGTH OF alpha-item
```

変換前の dbcs-item および alpha-item の内容と長さが次のものであるとします。

```
dbcs-item = .A.BD1D2D3.C.D  
alpha-item = ssssssssssss
```

```
LENGTH OF dbcs-item = 14  
LENGTH OF alpha-item = 12
```

変換後、dbcs-item および alpha-item には次のものが入ります。

```
dbcs-item = .A.BD1D2D3.C.D  
alpha-item = AB<D1D2D3>CD
```

RETURN-CODE レジスターの内容は 0 です。

関連参照

[868 ページの『DBCS の表記』](#)

XML 参照資料

ここでは、XML 構文解析中または XML 生成中に返される XML 例外コードについて説明します。

関連参照

[872 ページの『XMLPARSE\(XMLSS\) が有効な場合の XML PARSE 例外』](#)

[874 ページの『XMLPARSE\(COMPAT\) が有効な場合の XML PARSE 例外』](#)

[880 ページの『XML GENERATE 例外』](#)

[XML specification](#)

XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外

z/OS XML System Services パーサー が例外イベントのために処理プロシージャーに制御を渡すときは、XML-CODE 特殊レジスターに例外コードが含まれます。この例外コードは、戻りコードと理由コードから形成されます。

戻りコードと理由コードはそれぞれハーフワード・バイナリー値です。例外コードは、上位ハーフワードの戻りコードと、下位ハーフワードの理由コードの 2 つの値を連結したものです。

戻りコードと理由コードは、「z/OS XML システム・サービス・ユーザー・ガイド&リファレンス」(下記参照) および「[873 ページの表 107](#)」に 16 進数値として記載されています。

多くの例外イベントの後でパーサーは処理を続行しません。XML PARSE ステートメントの末尾にある XML-CODE の値は、パーサーが設定した元の例外コードです。

例外イベント後に処理手順がパーサーに戻ると、制御は ON EXCEPTION フレーズの末尾または XML PARSE ステートメントをコーディングしなかった場合 ON EXCEPTION フレーズ。

検証の例外:

コーディングする場合 XML PARSE ステートメントには、VALIDATING フレーズ、そして z/OSXML システムサービスパーサーは文書が有効でないと判断した場合、戻りコード 24 (16 進数 18、XRC_NOT_VALID)。

Enterprise COBOL に固有の例外

一部の例外は Enterprise COBOL に固有のものであるため、z/OS XML システム・サービス・ユーザー・ガイド&リファレンス には記載されていません (例: XML スキーマの検索中に発生するエラー)。理由コードが 16 進数 800~899 の例外の戻りコードは 4 (16 進数 0004、XRC_WARNING その他の例外の場合、戻り

コードは 16 (16 進数 0010、XRC_FATAL)。例外コード (特殊レジスタの値) XML-CODE) は、この戻りコードと、次の表に示す理由コードのいずれかを連結して形成されます。

表 107. <i>Enterprise COBOL</i> に固有な XML PARSE 例外の理由コード	
理由コード (16 進数)	説明
700	VALIDATING WITH FILE は CICS でサポートされません。
701	読み込まれた最適化された XML スキーマが短すぎるか、またはファイルが空でした。
702	スキーマのファイル ID は、DD 名または環境変数名ではありません。
703	DSN 値で、スペースが許されない位置にスペース文字が含まれています。
704	DSN 値は、一時データ・セットを指定しています。
705	PATH 値に、エスケープされていないスペース文字が含まれています。
706	PATH 値に、絶対パスでないパス名が含まれています。
707	XML スキーマのバッファに対するメモリの割り振りが失敗しました。
708	環境変数が NULL であるか、環境変数にスペースしか含まれていません。
709	環境変数に無効なキーワードが含まれています。
710	DSN 値で、メンバー名の後に無効な文字が含まれています。
711	DSN 値でメンバー名を指定していません。
712	DSN 値でデータ・セット名を指定していないか、または括弧が正しく指定されていません。
713	PATH 値でパス名を指定していないか、または括弧が正しく指定されていません。
714	DSN 値に余分な括弧が含まれています。
715	PATH 値に余分な括弧が含まれています。
716	DSN 値に右括弧が欠落しています。
717	PATH 値に右括弧が欠落しています。
718	DSN 値にエスケープ文字が含まれています。
720	表現不能な文字の文字参照が解決されませんでした。
721	文書タイプ宣言内の表現不能な文字の参照がサポートされていません。
800	宣言されていない接頭部が属性名で使用されました。
801	宣言されていない接頭部が START-OF-ELEMENT 名で使用されました。(END-OF-ELEMENT 名は一致しなければならないため、宣言されていない同じ接頭部が使用されていても、ほかに例外は発生しません。)
900	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。

900 以外の理由コードの場合には、エラーを訂正してプログラムを再試行してください。

関連概念

694 ページの『XML-CODE』

693 ページの『XML イベント』

関連タスク

709 ページの『XML PARSE の例外処理』

関連参照

462 ページの『XMLPARSE』 (コンパイラー・オプション)

XML PARSE ステートメント (Enterprise COBOL for z/OS 言語解説書)

z/OS XML System Services User's Guide and Reference

XMLPARSE(COMPAT) が有効な場合の XML PARSE 例外

例外イベントが発生すると、XML パーサーは提供される Enterprise COBOL 図書館特殊レジスタを設定する XML-CODE 例外を識別する値に設定します。以下で参照されている情報に詳述されているように、XML-CODE の値に応じて、例外の後にパーサーが処理を続行できる場合と、続行できない場合があります。

関連参照

874 ページの『継続を許可する XML PARSE 例外』

877 ページの『継続を許可しない XML PARSE 例外』

継続を許可する XML PARSE 例外

もし、XMLPARSE(COMPAT) コンパイラオプションが有効かどうか XML パーサーは、例外イベント後に処理を続行できるかどうかは、例外コードの値によって異なります。

特殊レジスター XML-CODE 内の例外コードが次のいずれかの範囲にあれば、パーサーは処理を続行することができます。

- 1 から 99
- 100,001 から 165,535

次の表には、それぞれの例外の説明と、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示しています。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。

- 実績の文書化エンコード方式
- 文書エンコード宣言

用語の定義については、XML 入力文書エンコードに関する関連概念を参照してください。

表 108. 継続を許可する XML PARSE 例外		
例外コード (10 進数)	説明	続行に関す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。 空白文字の詳細については、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。	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。
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。

表 108. 継続を許可する XML PARSE 例外 (続き)

例外コード (10 進数)	説明	続行に関するパーサーのアクション
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。
8	パーサーは、エレメント・コンテンツ内に、一致する開始文字シーケンス「<![CDATA[」のない CDATA 終了文字シーケンス「]]>」を検出しました。	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 '>' が付いていない文字シーケンス '--' (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。

表 108. 継続を許可する XML PARSE 例外 (続き)

例外コード (10 進数)	説明	続行に関するパーサーのアクション
※13	パーサーは、10 進文字参照に無効桁を検出しました (形式 &#xdddd;)。	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。
14	パーサーは、10 進文字参照に無効桁を検出しました (形式 &#dddd;)。	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。
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	実際の文書のエンコードは EBCDIC で、CODEPAGE コンパイラオプションはサポートされている EBCDIC コードページを指定しましたが、ドキュメントのエンコード宣言で、サポートされている 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、CODEPAGE コンパイラ・オプションで指定されたエンコードを使用します。
71	実際の文書のエンコーディングは EBCDIC であり、文書のエンコーディング宣言ではサポートされている EBCDIC エンコーディングが指定されていましたが、CODEPAGE コンパイラオプションはサポートされている EBCDIC コードページを指定していませんでした。	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 108. 継続を許可する XML PARSE 例外 (続き)		
例外コード (10 進数)	説明	続行に関す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、CODEPAGE コンパイラ・オプションではサポートされる EBCDIC コード・ページが指定されておらず、文書はエンコード宣言を含んでいませんでした。	パーサーでは、EBCDIC コード・ページ 1140 (米国、カナダ、... ユーロ国別拡張コード・ページ) が使用されます。
73	実際の文書のエンコードは EBCDIC でしたが、CODEPAGE コンパイラオプションまた、ドキュメントのエンコーディング宣言では、サポートされている EBCDIC コード ページも指定されていません。	パーサーでは、EBCDIC コード・ページ 1140 (米国、カナダ、... ユーロ国別拡張コード・ページ) が使用されます。
82	実際の文書エンコードは ASCII ですが、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、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 ですが、文書エンコード宣言ではコード・ページ 813、819、および 920 のいずれも指定さ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。
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。	パーサーでは、コード・ページ 1200 (Unicode UTF-16) が使用されます。
100,001 から 165,535	CODEPAGE コンパイラ・オプション および文書エンコード宣言で、サポート対象の別の EBCDIC コード・ページが指定されました。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、CODEPAGE コンパイラ・オプション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。

関連概念

[694 ページの『XML-CODE』](#)

[705 ページの『XML 入力文書エンコード』](#)

関連タスク

[709 ページの『XML PARSE の例外処理』](#)

関連参照

[462 ページの『XMLPARSE』](#) (コンパイラ・オプション)

継続を許可しない XML PARSE 例外

もし、XMLPARSE (COMPAT) コンパイラオプションが有効になっている場合、以下で説明する例外のいずれかが発生した場合、XML パーサーは処理を続行できません。

処理プロシージャがパーサーに制御を戻す前に XML-CODE をゼロに設定していても、これらの例外について、パーサーからこれ以上のイベントは返されません。ON EXCEPTION 句が指定されている場合、パーサーは、この句のステートメントに制御を渡します。指定されていない場合は、XML PARSE ステートメントの最後に制御を渡します。

表 109. 継続を許可しない XML PARSE 例外 (のために XMLPARSE (COMPAT))

例外コード (10 進数)	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。
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・タグは、'/' の後の '>' で終了されませんでした。
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。

表 109. 継続を許可しない XML PARSE 例外 (のために XMLPARSE (COMPAT)) (続き)	
例外コード (10 進数)	説明
134	エレメント終了タグ名は、'>' で終了されませんでした。
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	処理命令は、終了文字シーケンス '?>' で終了されませんでした。
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。
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 宣言は、正しい文字シーケンス '?>' で終了されませんでした。または、無効な属性が含まれていました。

表 109. 継続を許可しない XML PARSE 例外 (のために XMLPARSE (COMPAT)) (続き)

例外コード (10 進数)	説明
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。
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。
500 - 599	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。

関連概念

694 ページの『XML-CODE』

関連タスク

709 ページの『XML PARSE の例外処理』

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。これらの例外のいずれかが発生した場合、制御は ON EXCEPTION フレーズの末尾または XML GENERATE ステートメントをコーディングしなかった場合 ON EXCEPTION フレーズ。

表 110. XML GENERATE 例外

例外コード (10 進数)	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。
401	DBCS データ名は、Unicode への変換時に XML エレメントまたは属性名では無効な文字を含んでいました。
402	Unicode に変換されたときの DBCS データ名の最初の文字が、XML エレメントまたは属性名の先頭文字として有効ではありませんでした。
403	OCCURS DEPENDING ON 変数の値が 16,777,215 を超えました。
410	の CCSID ページは、CODEPAGE コンパイラオプション Unicode への変換はサポートされていません。
411	CODEPAGE コンパイラ・オプションによって指定される CCSID は、サポート対象の 1 バイト EBCDIC CCSID ではありません。
414	XML 文書で指定される CCSID は、無効であるか、サポートされません。
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。

表 110. XML GENERATE 例外 (続き)	
例外コード (10 進数)	説明
416	XML 名前空間 ID に無効な XML 文字が含まれていました。
417	エレメント文字内容または属性値に XML コンテンツでは正しくない文字が含まれていました。要素タグ名または属性名に「hex」というプレフィックスを付けて、XML 生成が続行されました。および文書内で 16 進数で表される元のデータ値です。 TYPE IS CONTENT 指定はすべて無視され、項目はエレメントとして扱われます。
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	受信側は英数字で、入力には国別または DBCS のデータまたは名前が含まれていましたが、ドキュメントに指定されたエンコードは 1208 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

732 ページの『XML GENERATE 例外の処理』

JSON 参照資料

ここでは、JSON 構文解析中または JSON 生成中に返される JSON 例外コードについて説明します。

関連参照

881 ページの『JSON GENERATE 例外』

882 ページの『JSON PARSE 状態、関連コード、およびランタイム・メッセージ』

882 ページの『非例外状態とそれに対応する JSON-STATUS 値』

883 ページの『例外状態とそれに対応する JSON-CODE 値』

883 ページの『非例外状態ランタイム・メッセージ』

884 ページの『例外状態ランタイム・メッセージ』

JSON GENERATE 例外

JSON の生成時に、いずれかの例外コードが JSON-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、JSON GENERATE ステートメントの末尾に制御が渡されます。

表 111. JSON GENERATE 例外	
例外コード (10 進数)	説明
1	受信は小さすぎて、生成された JSON テキスト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트가格納されています。
2	JSON GENERATE ステートメントの ENCODING 句の CCSID が無効またはサポートされていません。
504	ストレージが不足しています JSON GENERATE 作業エリア。領域サイズを大きくしてアプリケーションを再実行してください。
507	内部エラー。LE ランタイム・レベルが最新でない可能性があります。最新の LE ランタイム PTF をインストールしてください。エラーが解決されない場合は、サービス担当者にエラーについて報告してください。

表 111. JSON GENERATE 例外 (続き)	
例外コード (10 進数)	説明
500 - 599 (上記で定義された例外コードを除く)	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。

JSON PARSE 状態、関連コード、およびランタイム・メッセージ

JSON PARSE ステートメントの実行時に 2 種類の状態が発生することがあります。その場合は、受け取り側が部分的に変更される可能性があります。

- 非例外状態の場合は、理由コードが特殊レジスター JSON-STATUS に設定されますが、ステートメントの実行は終了しません。
- 例外状態の場合は、例外コードが特殊レジスター JSON-CODE に設定され、ステートメントの実行は終了します。

次の表にある JSON-STATUS 理由コード値は加法方式になっています。例えば、JSON PARSE ステートメントが実行されるとコード 1 とコード 4 の状態が発生するとします。この場合、JSON-STATUS 値は合算されて 5 となります。所定の状態が発生して、対応するコードが JSON-STATUS にあるかどうかを判別するには、次のようなステートメントを使用します。

```
IF FUNCTION MOD(JSON-STATUS 2 * code) / code = 1
  DISPLAY 'JSON-STATUS condition ' code ' occurred.'
END-IF
```

code は個々の JSON-STATUS コードのいずれかです。

ランタイム・メッセージが発行されるのは、WITH DETAIL 句が JSON PARSE ステートメントで指定された場合に限られます。特殊レジスター JSON-STATUS および JSON-CODE は常に設定されます。

非例外状態とそれに対応する JSON-STATUS 値

表 112. JSON 非例外状態の理由コード	
JSON-STATUS レジスターでの理由コード	説明
1	1 つ以上のデータ項目が変更されませんでした。適合する JSON 名前/値ペアがなかったためです。
2	1 つ以上の JSON 名前/値ペアがいずれのデータ項目にも適合しませんでした。
4	1 つ以上のデータ項目に、適合する JSON 名前/値ペアが複数あり、値が重複していました。
8	1 つ以上のテーブル・データ項目に、適合する JSON 配列よりも多くのエレメントが含まれていました。
16	1 つ以上の JSON 配列に、適合するデータ項目よりも多くの値が含まれていました。
32	1 つ以上のデータ項目が変更されませんでした。対応する JSON 名前/値ペアの値が null であったためです。
64 百万	1 つ以上のテーブル・データ項目に未変更のエレメントが含まれていました。対応する JSON 値が null だったためです。

表 112. JSON 非例外状態の理由コード (続き)	
JSON-STATUS レジスターでの理由コード	説明
128	1 つ以上の数値割り当てで「SIZE ERROR」状態が検出されました。それにもかかわらずデータ項目は変更されました。
256	1 つ以上の英数字割り当てに、情報が欠落していました。それにもかかわらずデータ項目は変更されました。
512	1 つ以上の JSON 名前/値ペアにおいて、そこに含まれる値が原因で、CODEPAGE コンパイラー・オプションで指定された CCSID に Unicode から変換が行われたときに 1 つ以上の置換文字が発生しました。

例外状態とそれに対応する JSON-CODE 値

表 113. JSON 例外状態の理由コード	
JSON-CODE レジスターでの理由コード	説明
100	JSON テキストが無効でした。
101	この JSON テキストの長さはゼロでした。または、この JSON テキストは空白文字のみで構成されていました。
102	最外部 JSON オブジェクトの右中括弧の後に不要な非空白文字が見つかりました。
103	1 つ以上のデータ項目に、適合する JSON 名前/値ペアが複数あり、それぞれの値が異なっていました。そのデータ項目は、JSON テキストで見つかった左端の値に設定されました。
104	1 つ以上の JSON 名前/値ペアに、適合するデータ項目とは非互換の値が含まれていました。
105	1 つ以上の一致する JSON 名/値のペアの値が true または false。
106	JSON 名前/値ペアはいずれのデータ項目にも適合しませんでした。
107	1 つ以上の一致する JSON 名/値のペアの値が true または false しかし、一致する COBOL データ項目が 'CONVERTING BOOLEAN' 句に指定されていませんでした。
108	NAME OMITTED 句と OCCURS 句のない JSON PARSE ステートメントが、JSON テキスト内の匿名 JSON 配列を解析しようとしていました。
109	JSON PARSE ステートメントの ENCODING 句の CCSID が無効またはサポートされていません。

非例外状態ランタイム・メッセージ

以下のメッセージは、WITH DETAIL 句が指定された場合に発行されます。

IGZ0321I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、データ項目 *data-name* に適合する JSON 名前/値ペアがありませんでした。そのため、このデータ項目は変更されませんでした。

IGZ0321I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある JSON 名 *JSON-name* に適合するデータ項目がありませんでした。

IGZ0323I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、*offset* にある重複 JSON 名前/値ペアがデータ項目 *data-name* に適合しました。重複値は受け入れられました。

IGZ0324I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある *JSON-name* という名前の JSON 配列に含まれるエレメントの数が、適合するテーブル項目 *data-name* のエレメントの数より少ない数でした。追加テーブル・エレメントは変更されませんでした。

IGZ0325I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある *JSON-name* という名前の JSON 配列に含まれる値の数が、適合するテーブル項目 *data-name* の値の数より多い数でした。追加の値は無視されました。

IGZ0326I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある JSON 名前/値ペアの値が特殊値 `null` だったため、データ項目 *data-name* は変更されませんでした。

IGZ0327I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある *JSON-name* という名前の JSON 配列に 1 つ以上の `null` 値が含まれていました。突き合わせテーブル項目 *data-name* において対応するエレメントは変更されませんでした。

IGZ0328I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある JSON 名前/値ペアの値がデータ項目 *data-name* に割り当てられると、有効数字が失われました (「SIZE ERROR」)。

IGZ0329I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある JSON 名前/値ペアの値がデータ項目 *data-name* に割り当てられると、情報が失われました。

IGZ0330I

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある *JSON-name* という名前の JSON 名前/値ペアの値が、CODEPAGE コンパイラ・オプションによって指定された CCSID に Unicode から変換されると、1 つ以上の置換文字になりました。変換された値はデータ項目 *data-name* に割り当てられました。

例外状態ランタイム・メッセージ

以下のメッセージは、WITH DETAIL 句が指定された場合に発行されます。

IGZ0335W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、*data-name* に含まれる JSON テキストが無効であることが判明しました。オフセット *offset* にある JSON テキスト *JSON-text-fragment* の後に *JSON-token* が見つかりましたが、予期されたのは *JSON-tokens* のいずれかでした。

IGZ0336W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、*data-name* に含まれる JSON テキストが無効であることが判明しました。この JSON テキストの長さはゼロでした。または、この JSON テキストは空白文字のみで構成されていました。

IGZ0337W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、*data-name* に含まれる JSON テキストが無効であることが判明しました。最外部 JSON オブジェクトの右中括弧の後に不要な文字 *text* が見つかりました。

IGZ0338W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある JSON 名前/値ペアがデータ項目 *data-name* に対する重複適合でした。ただし、値が、最初 (左端) で適合する名前/値ペアとは異なっていました。最初に適合した JSON 名前/値ペアの値は保存されました。

IGZ0339W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある *JSON-name* という名前の JSON 名前/値ペアの値が、適合するデータ項目とは非互換であることが判明しました。

IGZ0340W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある *JSON-name* という名前の JSON 名前/値ペアの値が特殊値 `true` または `false` のいずれかであることが判明しました。

IGZ0341W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、JSON 名前/値ペアが、受け取り側におけるいずれのデータ項目にも適合しませんでした。受け取り側 *data-name* は変更されませんでした。

IGZ0342W

プログラム *program-name* の *line-number* 行目にある JSON PARSE ステートメントが実行されたときに、オフセット *offset* にある *JSON-name* という名前の JSON 名前/値ペアの値が特殊値 `true` または `false` のいずれかであることが判明しました。しかし、一致する COBOL データ項目 *data-name* が、JSON PARSE ステートメントの 'CONVERTING BOOLEAN' フレーズで指定されていません。

IGZ0343W

行の JSON PARSE ステートメントの実行中行番号プログラムのプログラム名、JSON テキストデータ名匿名 JSON 配列であることが判明しましたが、'NAME *data-name* OMITTED' JSON PARSE 文の句が指定されていないか、データ名 OCCURS 句が含まれていませんでした。

IGZ0344W

行の JSON PARSE ステートメントの実行中行番号プログラムのプログラム名、CCSID *ccsid* 番号 ENCODING 句が無効またはサポートされていないことが判明しました。

EXIT コンパイラー・オプション

EXIT コンパイラー・オプションを使用して、各種コンパイラー関数に代わるユーザー提供モジュールを指定できます。各終了モジュールの処理、終了モジュールのエラー処理、または EXIT オプション付き CICS、SQL そして SQLIMS ステートメントについては、次のトピックを参照してください。

注：すべてのユーザー出口ルーチンは AMODE 31 または AMODE ANY でなければなりません。LP(64) を使用して出口ルーチンをコンパイルしないでください。

関連タスク

[886 ページの『ユーザー出口作業域の使用』](#)

[886 ページの『出口モジュールからの呼び出し』](#)

[CICS、SQL および](#)

[SQLIMS ステートメント](#)

関連参照

[399 ページの『出口』](#)

[887 ページの『INEXIT の処理』](#)

[888 ページの『LIBEXIT の処理』](#)

[891 ページの『PRTEXIT の処理』](#)

[892 ページの『ADEXIT の処理』](#)

ユーザー出口作業域の使用

ユーザー出口を使用するとき、コンパイラーは作業域を提供します。その作業域には、出口モジュールが取得した GETMAIN ストレージのアドレスを保存できます。こうした作業域により、モジュールを再入可能にすることができます。

このタスクについて

ユーザー出口作業域は、フルワード境界に位置する 6 つのフルワードから構成されます。このフルワードは、最初の出口ルーチンが呼び出される前に、2 進ゼロに初期化されます。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。

ユーザー出口作業域のワードは、次の表に示した個別の出口モジュールによって使用されます。

表 114. ユーザー出口作業域のレイアウト	
ワード番号	使用するモジュール
1	INEXIT
2	LIBEXIT
3	PRTEXIT
4	ADEXIT
5	(予約済み)
6	MSGEXIT

関連参照

[887 ページの『INEXIT の処理』](#)

[888 ページの『LIBEXIT の処理』](#)

[891 ページの『PRTEXIT の処理』](#)

[892 ページの『ADEXIT の処理』](#)

[893 ページの『MSGEXIT の処理』](#)

出口モジュールからの呼び出し

出口モジュール内で COBOL プログラムまたはライブラリー・ルーチンを呼び出すには、標準の COBOL リンケージを使用します。呼び出しチェーンを正しくトレースするためには、レジスターの規則を認識する必要があります。

このタスクについて

出口モジュールでプログラムまたはルーチンを呼び出すと、レジスターは次のようにセットアップされます。

R1

呼び出されるプログラムまたはライブラリー・ルーチンに渡されるパラメーター・リストを指す

R13

呼び出し側プログラムまたはルーチンが提供するレジスター保管域を指す

R14

呼び出し側プログラムまたはルーチンの戻りアドレスを保持する

R15

呼び出されるプログラムまたはルーチンのアドレスを保持する

終了モジュールには RMODE 属性 24 そして AMODE 属性 ANY。

INEXIT の処理

INEXIT 終了モジュールは、SYSIN から直接取得するのではなく、プライマリ ソース コードを読み取るために使用されます。

表 115. INEXIT 処理	
コンパイラーのアクション	出口モジュールのアクション
初期化時に出口モジュール (<i>mod1</i>) をロードします	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します	処理するためのソースを準備します。OPEN 要求の状況をコンパイラーに戻します。
ソース・ステートメントが必要になったときは、GET 命令コードを使用してこの出口モジュールを呼び出します	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) データの終わり標識のいずれかを戻します
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その出力に関連のあるすべてのリソースを解放します

INEXIT パラメーター

コンパイラーは、参照で渡される 10 個のパラメーターを使用して、出口モジュールと連絡します。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設定され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。

表 116. INEXIT パラメーター		
パラメーター番号	パラメーター項目	項目の説明
1	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	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	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、要求された操作の成功を示します。 0= 操作は成功しました 4=End-of-data 12= 操作は失敗しました
4	ユーザー出口作業域	ユーザー出口モジュールでできるように、コンパイラーが提供する 6 フルワードの作業域 最初のワード: INEXIT が使用
5	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 (80 でなければなりません)

表 116. INEXIT パラメーター (続き)		
パラメーター番号	パラメーター項目	項目の説明
6	データまたは <i>str1</i>	出口モジュールが設定するフルワードであり、GET 操作の際、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。 <i>str1</i> は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
7	未使用	(LIBEXIT および MSGEXIT の専用)
8	未使用	(LIBEXIT の専用)
9	未使用	(LIBEXIT の専用)
10	未使用	(LIBEXIT の専用)

関連タスク

886 ページの『ユーザー出口作業域の使用』

886 ページの『出口モジュールからの呼び出し』

CICS、SQL および

SQLIMS ステートメント

LIBEXIT の処理

SYSLIB (または *library-name*) データ・セットの代わりに LIBEXIT 出口モジュールが使用されます。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。

表 117. LIBEXIT 処理	
コンパイラーのアクション	出口モジュールのアクション
初期化時に出口モジュール (<i>mod2</i>) をロードします	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します	指定された <i>library-name</i> を処理のために準備します。OPEN 要求の状況をコンパイラーに渡します。
<i>library-name</i> が正常にオープンされた場合に、FIND 命令コードを使用してこの出口モジュールを呼び出します	指定された <i>library-name</i> 内の要求された <i>text-name</i> (または <i>basis-name</i>) のところに、位置決めを行います。この場所がアクティブ・コピーブックになります。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
GET 命令コードを使用してこの出口モジュールを呼び出します	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、アドレス、またはデータの終わり標識のいずれかをコンパイラーに渡します
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その入力に関連のあるすべてのリソースを解放します

ネストされた COPY ステートメントによる LIBEXIT の処理

アクティブ・コピーブックのレコードは COPY ステートメントを含むことができます。

text-name の再帰呼び出しは行えません。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。

次の表は、ネストされていない1つ以上の有効な COPY ステートメントがあるときに、LIBEXIT の処理がどのように変わるかを示しています。

表 118. ネストなしの COPY ステートメントによる LIBEXIT の処理	
コンパイラーのアクション	出口モジュールのアクション
初期化時に出口モジュール (mod2) をロードします	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します	指定された <i>library-name</i> を処理のために準備します。OPEN 要求の状況をコンパイラーに渡します。
<i>library-name</i> が正常にオープンされた場合に、FIND 命令コードを使用してこの出口モジュールを呼び出します	指定された <i>library-name</i> 内の要求された <i>text-name</i> (または <i>basis-name</i>) のところに、位置決めを行います。この場所がアクティブ・コピーブックになります。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
<i>library-name</i> が正常にオープンされた場合に、FIND 命令コードを使用してこの出口モジュールを呼び出します	前のアクティブ・コピーブックのところに、位置決めを設定し直します。位置決めが完了したときに、適切な戻りコードをコンパイラーに渡します。
GET 命令コードを使用してこの出口モジュールを呼び出します。同じレコードが渡されたかどうかを検査します。	このコピーブックから前に渡されたものと同じレコードをコンパイラーに渡します。検査後に、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してアドレス、またはデータの終わり標識のいずれかをコンパイラーに渡します。
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その入力に関連のあるすべてのリソースを解放します

次の表は、ネストされた1つの有効な COPY ステートメントをコンパイラーが検出すると、LIBEXIT の処理がどのように変わるかを示しています。

表 119. ネストされた COPY ステートメントによる LIBEXIT の処理	
コンパイラーのアクション	出口モジュールのアクション
要求された場合ライブラリ名ネストされた COPY ステートメントが以前に開かれていない場合、OPEN 命令コードで終了モジュールを呼び出す。	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押し入れます。要求されたアクション (OPEN) を完了させます。新しく要求された <i>text-name</i> (または <i>basis-name</i>) がアクティブ・コピーブックになります。
要求された新しい <i>text-name</i> のために FIND 命令コードを使用してこの出口モジュールを呼び出します	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押し入れます。要求されたアクション (FIND) を完了させます。新しく要求された <i>text-name</i> (または <i>basis-name</i>) がアクティブ・コピーブックになります。
GET 命令コードを使用してこの出口モジュールを呼び出します	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してアドレス、またはデータの終わり標識のいずれかをコンパイラーに渡します。データの終わりに、スタックからその制御情報をポップします。

LIBEXIT パラメーター

コンパイラーは、参照で渡される 10 個のパラメーターを使用して、出口モジュールと連絡します。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設定され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。

表 120. LIBEXIT パラメーター

パラメーター番号	パラメーター項目	項目の説明
1	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。 2=LIBEXIT
2	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 • 4=END
3	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、要求された操作の成功を示します。 • 0= 操作は成功しました • 4=End-of-data • 12= 操作は失敗しました
4	ユーザー出口作業域	ユーザー出口モジュールでできるように、コンパイラーが提供する 6 フルワードの作業域 2 番目のワード: LIBEXIT が使用
5	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 (80 でなければなりません)
6	データまたは <i>str2</i>	出口モジュールが設定するフルワードであり、GET 操作の際、ユーザー所有のバッファ内のレコードのアドレスが入ります。 <i>str2</i> は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) 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。
7	システム <i>library-name</i>	COPY ステートメントの <i>library-name</i> が入れられる 8 文字の区域。プログラム名の処理規則および変換規則が適用されます。必要に応じて、空白が埋め込まれます。OPEN、CLOSE、および FIND に適用されます。
8	システム <i>text-name</i>	COPY ステートメントの <i>text-name</i> (BASIS ステートメントの <i>basis-name</i>) が入れられる 8 文字の区域。 <i>program-name</i> の処理規則と変換規則が適用されます。必要に応じて、空白が埋め込まれます。FIND にのみ適用されます。
9	ライブラリー名	COPY ステートメントからの完全な <i>library-name</i> が入れられる 30 文字の区域。必要な場合には空白が埋め込まれ、そのまま (英大文字に変換されず) で使用されます。OPEN、CLOSE、および FIND に適用されます。
10	テキスト名	COPY ステートメントからの完全な <i>text-name</i> が入れられる 30 文字の区域。必要な場合には空白が埋め込まれ、そのまま (英大文字に変換されず) で使用されます。FIND にのみ適用されます。

関連タスク

886 ページの『ユーザー出口作業域の使用』

886 ページの『出口モジュールからの呼び出し』

PRTEXIT の処理

SYSPRINT データ・セットの代わりに PRTEXIT 出口モジュールを使用します。

表 121. PRTEXIT 処理	
コンパイラーのアクション	出口モジュールのアクション
初期化時に出口モジュール (mod3) をロードします	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します	処理のためにその出力宛先を準備します。OPEN 要求の状況をコンパイラーに渡します。
行が印刷されるときに、印刷されるレコードのアドレスと長さを与え、PUT 命令コードを使用してこの出口モジュールを呼び出します	PUT 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。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。
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	その出力宛先に関連のあるすべてのリソースを解放します

PRTEXIT パラメーター

コンパイラーは、参照で渡される 10 個のパラメーターを使用して、出口モジュールと連絡します。戻りコード、データ長、およびデータ・バッファ・パラメーターは、出口モジュールによって設定され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。

表 122. PRTEXIT パラメーター		
パラメーター番号	パラメーター項目	項目の説明
1	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。 3=PRTEXIT
2	命令コード	操作のタイプを示すハーフワード。 0=OPEN 1=CLOSE 3=PUT
3	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、要求された操作の成功を示します。 0= 操作は成功しました 12= 操作は失敗しました
4	ユーザー出口作業域	ユーザー出口モジュールでできるように、コンパイラーが提供する 6 フルワードの作業域 3 番目のワード: PRTEXIT が使用
5	データ長	PUT 操作によって提供されるレコードの長さを指定するフルワード (コンパイラーはこの値を 133 に設定します)

表 122. PRTEXIT パラメーター (続き)		
パラメーター番号	パラメーター項目	項目の説明
6	データ・バッファーまたは <i>str3</i>	PUT 操作によって印刷されるレコードをコンパイラーが配置したデータ・バッファー。 <i>str3</i> は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
7	未使用	(LIBEXIT および MSGEXIT の専用)
8	未使用	(LIBEXIT の専用)
9	未使用	(LIBEXIT の専用)
10	未使用	(LIBEXIT の専用)

関連タスク

886 ページの『ユーザー出口作業域の使用』

886 ページの『出口モジュールからの呼び出し』

CICS、SQL および

SQLIMS ステートメント

ADEXIT の処理

ADEXIT モジュールは、SYSADATA 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。

ADEXIT モジュールを使用するには、ADATA オプションを使用してコンパイルして SYSADATA 出力を生成し、SYSADATA DD ステートメントをコーディングする必要があります。

表 123. ADEXIT 処理	
コンパイラーのアクション	出口モジュールのアクション
初期化時に出口モジュール (<i>mod4</i>) をロードします	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します	処理のためにその出力宛先を準備します。OPEN 要求の状況をコンパイラーに渡します。
コンパイラーが SYSADATA レコードを書き込んだときに、その SYSADATA のアドレスと長さを与え、PUT 命令コードを使用してこの出口モジュールを呼び出します	PUT 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
データの終わりが存在するときに、CLOSE 命令コードを使用して出口モジュールを呼び出します	すべてのリソースを解放します

ADEXIT パラメーター

コンパイラーは、参照で渡される 10 個のパラメーターを使用して、出口モジュールと連絡します。戻りコード、データ長、およびデータ・バッファー・パラメーターは、出口モジュールによって設定され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。

表 124. ADEXIT パラメーター		
パラメーター番号	パラメーター項目	項目の説明
1	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。 4=ADEXIT
2	命令コード	操作のタイプを示すハーフワード。 • 0=OPEN • 1=CLOSE • 3=PUT
3	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、要求された操作の成功を示します。 • 0= 操作は成功しました • 12= 操作は失敗しました
4	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 6 フルワードの作業域 4 番目のワード: ADEXIT が使用
5	データ長	PUT 操作によって提供されるレコードの長さを指定するフルワード
6	データ・バッファーまたは <i>str4</i>	コンパイラーが PUT 操作によって印刷されるレコードを入れたデータ・バッファーのアドレスが含まれているフルワード。 <i>str4</i> は OPEN にのみ適用されます。最初のハーフワード (ハーフワード境界上の) にストリングの長さが入り、その後にストリングが続きます。
7	未使用	(LIBEXIT および MSGEXIT の専用)
8	未使用	(LIBEXIT の専用)
9	未使用	(LIBEXIT の専用)
10	未使用	(LIBEXIT の専用)

関連タスク

886 ページの『ユーザー出口作業域の使用』

886 ページの『出口モジュールからの呼び出し』

CICS、SQL および

SQLIMS ステートメント

関連参照

378 ページの『ADATA』

MSGEXIT の処理

MSGEXIT モジュールは、コンパイラー診断メッセージおよび FIPS メッセージのカスタマイズに使用します。このモジュールは、メッセージ重大度を変更するか、またはメッセージを抑制することによって、メッセージをカスタマイズします。

MSGEXIT モジュールが FIPS メッセージに重大度を割り当てると、メッセージは診断メッセージに変換されます。(メッセージは、リスト内の診断メッセージの要約に示されます。)

コンパイラー・リストの終わりにある MSGEXIT の要約に、重大度が変更されたメッセージの数と抑制されたメッセージの数が示されます。

表 125. MSGEXIT の処理	
コンパイラーのアクション	出口モジュールのアクション
初期化時に出口モジュール (mod5) をロードします	
OPEN 命令コード (op コード) を使用してこの出口モジュールを呼び出します	オプションで str5 を処理し、OPEN 要求の状況をコンパイラーに渡します
コンパイラーが診断メッセージまたは FIPS メッセージを出そうとするときに、MSGSEV 命令コード (op コード) を使用して出口モジュールを呼び出します	次のいずれかのアクション。 • (戻りコードに 0 を設定することによって) メッセージをカスタマイズしないことを示します。 • メッセージの新しい重大度 (または抑制) を指定し、戻りコードに 4 を設定します • (戻りコードに 12 を設定することによって) 操作が失敗したことを示します
CLOSE 命令コードを使用してこの出口モジュールを呼び出します	オプションでストレージを解放し、CLOSE 要求の状況をコンパイラーに渡します
コンパイラー終了時に出口モジュール (mod5) を削除します	

MSGEXIT パラメーター

コンパイラーは、参照で渡される 10 個のパラメーターを使用して、出口モジュールと連絡します。戻りコードおよびユーザーが要求した重大度パラメーターは、出口モジュールによって設定され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。

表 126. MSGEXIT パラメーター		
パラメーター番号	パラメーター項目	項目の説明
1	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。 6=MSGEXIT
2	命令コード	操作のタイプを示すハーフワード。 • 0=OPEN • 1=CLOSE • 5=MSGSEV: メッセージ重大度をカスタマイズします
3	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、要求された操作の成功を示します。 命令コード MSGSEV の場合: • 0= メッセージはカスタマイズされませんでした • 4= メッセージが検出され、カスタマイズされました • 12= 操作は失敗しました
4	ユーザー出口作業域	ユーザー出口モジュールでできるように、コンパイラーが提供する 6 フルワードの作業域 6 番目のワード: MSGEXIT が使用

表 126. MSGEXIT パラメーター (続き)

パラメーター番号	パラメーター項目	項目の説明
5	未使用	(他の出口が使用)
6	メッセージ出口データ	(ハーフワード境界に位置する) 3 ハーフワードの領域。 - 最初のハーフワード: カスタマイズするメッセージのメッセージ番号 2 番目のハーフワード: 診断メッセージの場合、デフォルトの重大度。FIPS メッセージの場合、数字コードによる FIPS カテゴリー 3 番目のハーフワード: ユーザーが要求したメッセージ重大度 (-1 は抑制を示します)
7	str5	(ハーフワード境界に位置する) 最初のハーフワード: スtring の長さの後に、String が続きます
8	未使用	(LIBEXIT の専用)
9	未使用	(LIBEXIT の専用)
10	未使用	(LIBEXIT の専用)

898 ページの『例: MSGEXIT ユーザー出口』

関連タスク

886 ページの『ユーザー出口作業域の使用』

886 ページの『出口モジュールからの呼び出し』

895 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

CICS、SQL および

SQLIMS ステートメント

コンパイラー・メッセージの重大度のカスタマイズ

コンパイラー・メッセージの重大度を変更したり、コンパイラー・メッセージ (FIPS メッセージを含む) を抑制したりするには、以下のステップを行います。

このタスクについて

1. ERRMSG という COBOL プログラムをコーディングしてコンパイルします。関連タスクに説明されているように、プログラムに必要なのは、PROGRAM-ID 段落のみです。
2. コンパイラー・メッセージとそのメッセージ番号、重大度、およびメッセージ・テキストの全リストが入っている ERRMSG のリストを確認します。
3. カスタマイズするメッセージを決定します。

可能なカスタマイズについては、カスタマイズ可能なコンパイラー・メッセージ重大度に関する関連参照を参照してください。

4. カスタマイズを実装する MSGEXIT モジュールをコーディングします。サンプルを参照
MSGEXITCOBOL コンパイラに含まれている SIGYSAMP データセットと呼ばれる IGYMSGXT。詳しくは、898 ページの『例: MSGEXIT ユーザー出口』を参照してください。

- a. 命令コードのパラメーターがメッセージ重大度のカスタマイズを指示することを確認します。
- b. メッセージ出口データ・パラメーターで、メッセージ番号と、診断メッセージの場合にはデフォルト重大度、または FIPS メッセージの場合には FIPS カテゴリーの 2 つの入力値を検査します。

FIPS カテゴリーは、数字コードで表されます。詳細については、カスタマイズ可能なコンパイラー・メッセージ重大度に関する関連参照を参照してください。

- c. カスタマイズするメッセージについて、メッセージ出口データ・パラメーターのユーザー要求の重大度が、以下のいずれかを示すように設定します。
 - 新しいメッセージ重大度。重大度 0、4、8、または 12 をコーディングします。
 - メッセージの抑制。重大度 -1 をコーディングします。
 - d. 戻りコードを次のいずれかの値に設定します。
 - 0: メッセージがカスタマイズされなかったことを示します。
 - 4: メッセージが見つかり、カスタマイズされたことを示します。
 - 12: 操作が失敗し、コンパイルが終了することを示します。
5. MSGEXIT モジュールをコンパイルし、リンクします。
 6. STEPLIB または JOBLIB DD ステートメントを使用して、MSGEXIT モジュールを含むデータ・セットをコンパイラー連結に追加します。
 7. コンパイラー・オプション EXIT(MSGEXIT(*msgmod*)) を使用してプログラム ERRMSG を再コンパイルします。ここで、*msgmod* は MSGEXIT モジュールの名前です。
 8. リストを確認し、以下を検査します。
 - 更新されたメッセージ重大度
 - 抑制されたメッセージ (重大度の代わりに XX で示されます)
 - サポートされていない重大度変更、またはサポートされていないメッセージ抑制 (重大度 U の診断メッセージと、戻りコード 16 のコンパイラー終了で示されます)

関連タスク

[307 ページの『コンパイラー・メッセージのリストの生成』](#)

関連参照

[308 ページの『コンパイラー診断メッセージの重大度コード』](#)

[896 ページの『カスタマイズ可能なコンパイラー・メッセージ重大度』](#)

[897 ページの『メッセージ・カスタマイズがコンパイル戻りコードに及ぼす影響』](#)

[903 ページの『出口モジュールでのエラー処理』](#)

カスタマイズ可能なコンパイラー・メッセージ重大度

コンパイラー・メッセージの重大度をカスタマイズするには、コンパイラー診断メッセージに可能な重大度、FIPS メッセージのレベルまたはカテゴリ、およびメッセージ重大度に関して可能なカスタマイズについて理解する必要があります。

コンパイラー診断メッセージに可能な重大度コードについては、重大度コードに関する関連参照に説明があります。

次の表に、FIPS (FLAGSTD) メッセージの 8 つのカテゴリを示します。FIPS メッセージのカテゴリは、数字コードで MSGEXIT モジュールに渡されます。2 番目の列にその数字コードが示されています。

表 127. FIPS (FLAGSTD) メッセージのカテゴリ		
FIPS レベルまたはカテゴリ	数字コード	説明
D	81	デバッグ・モジュール・レベル 1
E	82	拡張 (IBM)
H	83	上位レベル
I	84	中間レベル
N	85	分割モジュール・レベル 1
O	86	廃止されるエレメント
Q	87	上位レベルの廃止されるエレメント

表 127. FIPS (FLAGSTD) メッセージのカテゴリ (続き)		
FIPS レベルまたはカテゴリ	数字コード	説明
S	88	分割モジュール・レベル 2

FIPS メッセージの暗黙の重大度はゼロ (重大度 I) です。

メッセージ重大度の可能なカスタマイズ:

コンパイラ・メッセージの重大度は、次のように変更できます。

- 重大度 I および重大度 W のコンパイラ診断メッセージ、および FIPS メッセージは、I から S までの重大度に変更できます。

FIPS メッセージに重大度を割り当てると、FIPS メッセージは、割り当てられた重大度の診断メッセージに変換されます。

例として、以下のことを行うことができます。

- 最適化プログラムの警告を重大度 I に下げます。
- メッセージ 1154 の重大度を上げることによって、小さい項目のより大きい項目による REDEFINING を禁止します。
- FIPS メッセージ 8235 をカテゴリ E の FIPS メッセージから重大度 S のコンパイラ診断メッセージに変更することによって、複雑な OCCURS DEPENDING ON を禁止します。
- 重大度 E のメッセージは、プログラムでエラー条件が発生しているため、重大度 S に上げることはできませんが、重大度 I または W に下げることはできません。
- 重大度 S および重大度 U のメッセージは、別の重大度に変更できません。

コンパイラ・メッセージの抑制は、次のように要求できます。

- I、W、および FIPS のメッセージは抑制できます。
- E および S のメッセージは抑制できません。

関連参照

[308 ページの『コンパイラ診断メッセージの重大度コード』](#)

[403 ページの『FLAGSTD』](#)

[897 ページの『メッセージ・カスタマイズがコンパイル戻りコードに及ぼす影響』](#)

メッセージ・カスタマイズがコンパイル戻りコードに及ぼす影響

MSGEXIT モジュールを使用した場合、プログラム・コンパイルの最終戻りコードが以下のような影響を受けることがあります。

メッセージの重大度を変更すると、コンパイルの戻りコードも変わる場合があります。例えば、コンパイルで 1 件の診断メッセージが生成され、それが重大度 E のメッセージである場合、コンパイルの戻りコードは通常 8 になります。しかし、MSGEXIT モジュールでそのメッセージの重大度が重大度 S に変更されると、コンパイルの戻りコードは 12 になります。

メッセージを抑制した場合、コンパイルの戻りコードは、そのメッセージの重大度に影響されなくなります。例えば、コンパイルで 1 件の診断メッセージが生成され、それが重大度 W のメッセージである場合、コンパイルの戻りコードは通常 4 になります。しかし、MSGEXIT モジュールでそのメッセージが抑制されると、コンパイルの戻りコードは 0 になります。

関連タスク

[895 ページの『コンパイラ・メッセージの重大度のカスタマイズ』](#)

関連参照

[308 ページの『コンパイラ診断メッセージの重大度コード』](#)

例: MSGEXIT ユーザー出口

次の例は、メッセージ重大度を変更し、メッセージを抑制する MSGEXIT ユーザー出口モジュールを示しています。この例は、メンバー IGYMSGXT で、SIGYSAMP データ・セット内の Enterprise COBOL 6 に組み込まれています。

メッセージ出口モジュールの使用に役立つヒントについては、コード内のコメントを参照してください。

```
*****
* IGYMSGXT - Sample COBOL program for MSGEXIT *
*****
*
* IBM Enterprise COBOL for z/OS *
* バージョン 6 リリース 5 変更点 0 *
*
* LICENSED MATERIALS - PROPERTY OF IBM. *
*
* 5655-EC6 COPYRIGHT IBM CORP. 2024 *
* ALL RIGHTS RESERVED *
*
* US GOVERNMENT USERS RESTRICTED RIGHTS - USE, *
* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA *
* ADP SCHEDULE CONTRACT WITH IBM CORP. *
*
*****
*****
* Function: This is a SAMPLE user exit for the MSGEXIT *
* suboption of the EXIT compiler option. This exit *
* can be used to customize the severity of or *
* suppress compiler diagnostic messages and FIPS *
* messages. This example program includes several *
* sample customizations to show how customizations *
* are done. If you do not want the sample *
* customizations then either delete the unwanted *
* lines of code or comment them out with a comment *
* indicator in column 7 (*). *
* ----- *
*
* USAGE NOTE: To use this user exit program, make the *
* link-edited load module available to your *
* compilers that will use the MSGEXIT suboption of *
* the EXIT compiler option. Also, the name should *
* be changed, since IBM recommends that you avoid *
* having programs with names that start with IGY. *
* Sample steps to take: *
* 1) Make your customizations *
* 2) Change program name (E.G. MYEXIT) *
* 3) Compile and link into a dataset *
* 4) Include that dataset in your compile *
* JCL concatenation for the compile step. *
* If you link into USER.COBOLLIB: *
*
* //COBOL.STEPLIB DD DSNAME=SYS1.SIGYCOMP,DISP=SHR *
* // DD DSNAME=USER.COBOLLIB,DISP=SHR *
*
* 5) Finally, compile your programs with the *
* EXIT compiler option, EG: *
* EXIT(MSGEXIT(MYEXIT)) *
*
* COMPILE NOTE: Compile this program with NOEXIT. *
*
*****
Id Division.
Program-Id. IGYMSGXT.
Data Division.

Working-Storage Section.

*****
*
* Local variables. *
*****

77 EXIT-TYPEN PIC 9(4).
77 EXIT-DEFAULT-SEV-FIPS PIC X.
```

```

*****
*
* Definition of the User-Exit Parameter List, which is
* passed from the COBOL compiler to the user-exit module.
*
*****

Linkage Section.
01 EXIT-TYPE          PIC 9(4)    COMP.
01 EXIT-OPERATION     PIC 9(4)    COMP.
01 EXIT-RETURNCODE    PIC 9(9)    COMP.
01 EXIT-WORK-AREA.
    02 EXIT-WORK-AREA-PTR OCCURS 6 POINTER.
01 EXIT-DUMMY         POINTER.
01 EXIT-MESSAGE-PARMS.
    02 EXIT-MESSAGE-NUM PIC 9(4)    COMP.
    02 EXIT-DEFAULT-SEV PIC 9(4)    COMP.
    02 EXIT-USER-SEV   PIC S9(4)    COMP.
01 EXIT-STRING.
    02 EXIT-STR-LEN PIC 9(4)    COMP.
    02 EXIT-STR-TXT PIC X(64).

*****
*****
*
* Begin PROCEDURE DIVISION
*
* Check parameters and perform the operation requested.
*
*****
*****

Procedure Division Using EXIT-TYPE          EXIT-OPERATION
                        EXIT-RETURNCODE    EXIT-WORK-AREA
                        EXIT-DUMMY         EXIT-MESSAGE-PARMS
                        EXIT-STRING        EXIT-DUMMY
                        EXIT-DUMMY         EXIT-DUMMY.

    Compute EXIT-RETURNCODE = 0

    Evaluate TRUE

*****
* Handle a bad invocation of this exit by the compiler.
* This could happen if this routine was used for one of the
* other EXITS, such as INEXIT, PRTEXIT or LIBEXIT.
*****
    When EXIT-TYPE Not = 6
        Move EXIT-TYPE to EXIT-TYPEN
        Display '**** Invalid exit routine identifier'
        Display '**** EXIT TYPE = ' EXIT-TYPE
        Compute EXIT-RETURNCODE = 16

*****
* Handle the OPEN call to this exit by the compiler
* Display the exit string (str5 in syntax diagram) from
* the EXIT(MSGEXIT('str5',mod5)) option specification.
*****
    When EXIT-OPERATION = 0
        Display 'Opening MSGEXIT'
        If EXIT-STR-LEN Not Zero Then
            Display ' str5 len = ' EXIT-STR-LEN
            Display ' str5 = ' EXIT-STR-TXT(1:EXIT-STR-LEN)
        End-If
        Continue

*****
* Handle the CLOSE call to this exit by the compiler
*****
    When EXIT-OPERATION = 1
        Display 'Closing MSGEXIT'
        Goback

*****
* Handle the customize message severity call to this exit
* Display information about every customized severity.
*****
    When EXIT-OPERATION = 5
        Display 'MSGEXIT called with MSGSEV'
        If EXIT-MESSAGE-NUM < 8000 Then
            Perform Error-Messages-Severity

```

```

Else
  Perform FIPS-Messages-Severity
End-If

* If EXIT-RETURNCODE = 4 Then
*   Display '>>>> Customizing message ' EXIT=MESSAGE-NUM
*   ' with new severity ' EXIT-USER-SEV ' <<<<'
*   If EXIT-MESSAGE-NUM > 8000 Then
*     Display 'FIPS sev =' EXIT-DEFAULT-SEV-FIPS '<<<<'
*   End-If
* End-If

*****
* Handle a bad invocation of this exit by the compiler. *
* The compiler should not invoke this exit with EXIT-TYPE = 6 *
* and an opcode other than 0, 1, or 5. This should not happen *
* and IBM service should be contacted if it does. *
*****
When Other
  Display '**** Invalid MSGEXIT routine operation '
  Display '**** EXIT OPCODE = ' EXIT-OPERATION
  Compute EXIT-RETURNCODE = 16

End-Evaluate

Goback.

*****
* ERROR MESSAGE PROCESSOR *
*****
Error-Messages-Severity.

* Assume message severity will be customized...
Compute EXIT-RETURNCODE = 4

Evaluate EXIT-MESSAGE-NUM

*****
* Change severity of message 1154(W) to 12 ('S') *
* This is the case of redefining a large item *
* with a smaller item, IBM Req # MR0904063236 *
*****
When(1154)
  Compute EXIT-USER-SEV = 12

*****
* Modify the severity of RULES messages to enforce coding *
* standards or highlight coding that you want to avoid. *
* Here are the message numbers and what they flag: *
* 1158 RULES(NOOMITODOMIN) Missing min idx in ODO table def *
* 1348 RULES(NOEVENPACK) Even digit packed-decimal items *
* 1353 RULES(NOSLACKBYTES) Slack bytes within records *
* 1379 RULES(NOSLACKBYTES) Slack bytes between records *
* 2159 RULES(NOENDPERIOD) Cond stmt terminated by period *
* 2262 RULES(NOUNREFALL) Unref'd items (source/copybook) *
* 2262 RULES(NOUNREFSOURCE) Unref'd items (source only) *
* 2224 RULES(NOLAXPERF) Ineff. type for PERFORM VARYING *
* 2246 RULES(NOLAXPERF) Ineff. type for subscript *
* 2247 RULES(NOLAXPERF) Compiler option NOAWO in effect *
* 2248 RULES(NOLAXPERF) Option ARITH(EXTEND) in effect *
* 2249 RULES(NOLAXPERF) Option NOBLOCK0 in effect *
* 2250 RULES(NOLAXPERF) Option NOFASTSRT in effect *
* 2251 RULES(NOLAXPERF) Option NUMPROC(NOPFD) in effect *
* 2252 RULES(NOLAXPERF) Option OPTIMIZE(0) in effect *
* 2253 RULES(NOLAXPERF) Option SSRANGE in effect *
* 2254 RULES(NOLAXPERF) Option THREAD in effect *
* 2255 RULES(NOLAXPERF) Option TRUNC(STD) in effect *
* 2256 RULES(NOLAXPERF) Option TRUNC(BIN) in effect *
* 3084 RULES(NOLAXPERF) Ineff. type for arith sender *
* 3123 RULES(NOLAXPERF) Lots of padding in alph MOVE *
*
*****
When(1158) *> Disallow omitting ODO table min
  Compute EXIT-USER-SEV = 12
When(1348) *> Disallow even-digit Comp-3
  Compute EXIT-USER-SEV = 12
When(1353) When(1379) *> Disallow slack bytes
  Compute EXIT-USER-SEV = 12
When(2159) *> Disallow period-termination
  Compute EXIT-USER-SEV = 12 *> of conditional stmts
When(2262) *> Disallow unref'd data items
  Compute EXIT-USER-SEV = 12

```

```

* Highlight poorly performing COBOL features
  When(2224)      *> Ineff. type for PERFORM VARYING
  When(2246)      *> Ineff. type for subscript
  When(2247)      *> Compiler option NOAWO in effect
  When(2248)      *> Option ARITH(EXTEND) in effect
  When(2249)      *> Option NOBLOCK0 in effect
  When(2250)      *> Option NOFASTSRT in effect
  When(2251)      *> Option NUMPROC(NOPFD) in effect
  When(2252)      *> Option OPTIMIZE(0) in effect
  When(2253)      *> Option SSRANGE in effect
  When(2254)      *> Option THREAD in effect
  When(2255)      *> Option TRUNC(STD) in effect
  When(2256)      *> Option TRUNC(BIN) in effect
  When(3084)      *> Ineff. type for arith sender
  When(3123)      *> Lots of padding in alph MOVE
    Compute EXIT-USER-SEV = 8

*****
* Change severity of messages 3178(I) to highlight File *
* Definitions that could lead to wrong-length read conditions. *
* Message 3178 is issued when the length of the shortest *
* record description is less than the FROM integer in the *
* RECORD IS VARYING clause, and when the length of the *
* longest record description is greater than the TO integer *
* in the RECORD IS VARYING clause. *
*****
  When(3178)
    Compute EXIT-USER-SEV = 8

*****
* Change severity of messages 3188(W) and 3189(W)
* to 12 ('S'). This is to force a fix for all
* SEARCH ALL cases that might behave differently
* between COBOL compilers previous to Enterprise
* COBOL release 3.4 and later compilers such as
* Enterprise COBOL 4.2.
* Another way to handle this migration is to analyze all of
* the warnings you get and then change them to I-level when
* the analysis is complete.
*****
  When(3188) When(3189)
    Compute EXIT-USER-SEV = 12

*****
* Change severity of message 4019 to suppress this message.
* Message 4019 says 'Compiler option "OPTFILE" was specified, "
* but file SYSOPTF could not be opened. No options from a
* "SYSOPTF" file were used.
* This customization would allow users to specify OPTFILE
* as a default option and it would only have an affect
* if there is a SYSOPTF file present
*****
  When(4019)
    Compute EXIT-USER-SEV = -1

*****
* Change severity of 'optimization' messages to suppress them
* so that compilation Return Code can be zero (RC=0)
* 7300: The code from lines &2 in program '&1' can never
* be executed and was therefore discarded.
* 7301: A zero base was raised to a zero power in a numeric
* literal exponentiation. The result was set to 1.
* 7302: A zero base was raised to a negative power in a numeric
* literal exponentiation. The result was set to 0.
* 7304: An exception "&1" occurred while processing numeric
* literals. The result of the operation was set to zero.
* 7307: This statement may cause a program exception at execution
* time.
* 7309: There may be a loop from the "PERFORM" statement at "
* "PERFORM (line &1)" to itself.
* 7312: Procedure starting at 'proc-name' (line 'num') was
* copied one or more times to be inlined for PERFORM
* statements. xxx total bytes were generated from
* copying that procedure.
*****
  When(7300) When(7301) When(7302) When(7304)
  When(7307) When(7309) When(7312)
    Compute EXIT-USER-SEV = -1  *> Suppress the messages

*****
* Change severity of message 7311(W) to 12 ('S'). This is the *
* case of INITCHECK messages about uninitialized data items. *

```

```

*****
      When(7311)
          Compute EXIT-USER-SEV = 12

*****
*   Message severity Not customized
*****
      When Other
          Compute EXIT-RETURNCODE = 0

      End-Evaluate
      .
*****
*   FIPS MESSAGE PROCESSOR
*****
      Fips-Messages-Severity.

*   Assume message severity will be customized...
      Compute EXIT-RETURNCODE = 4

*   Convert numeric FIPS(FLAGSTD) 'category' to character
*   See the Programming Guide for description of FIPS category

      EVALUATE EXIT-DEFAULT-SEV
          When 81
              MOVE 'D' To EXIT-DEFAULT-SEV-FIPS
          When 82
              MOVE 'E' To EXIT-DEFAULT-SEV-FIPS
          When 83
              MOVE 'H' To EXIT-DEFAULT-SEV-FIPS
          When 84
              MOVE 'I' To EXIT-DEFAULT-SEV-FIPS
          When 85
              MOVE 'N' To EXIT-DEFAULT-SEV-FIPS
          When 86
              MOVE 'O' To EXIT-DEFAULT-SEV-FIPS
          When 87
              MOVE 'Q' To EXIT-DEFAULT-SEV-FIPS
          When 88
              MOVE 'S' To EXIT-DEFAULT-SEV-FIPS
          When Other
              Continue
      End-Evaluate

*****
*   Example of using FIPS category to force coding
*   restrictions. This is not a recommendation!
*   Change severity of all OBSOLETE item FIPS
*   messages to 'S'
*****
*   If EXIT-DEFAULT-SEV-FIPS = 'O' Then
*       Display '>>> Default customizing FIPS category '
*       EXIT-DEFAULT-SEV-FIPS ' msg ' EXIT-MESSAGE-NUM '<<<<'
*       Compute EXIT-USER-SEV = 12
*   End-If

      Evaluate EXIT-MESSAGE-NUM
*****
*   Change severity of message 8062(O) to 8 ('E')
*   8062 = GO TO without proc name
*****
      When(8062)
          Compute EXIT-USER-SEV = 8

*****
*   Change severity of message 8193(E) to 0('I')
*   8193 = GOBACK
*****
      When(8193)
          Compute EXIT-USER-SEV = 0

*****
*   Change severity of message 8235(E) to 8 (Error)
*   to disallow Complex Occurs Depending On
*   8235 = Complex Occurs Depending On
*****
      When(8235)
          Compute EXIT-USER-SEV = 08

*****
*   Change severity of message 8270(O) to -1 (Suppress)
*   8270 = SERVICE LABEL

```

```

*****
      When(8270)
        Compute EXIT-USER-SEV = -1

*****
*      Message severity Not customized
*****
      When Other
        For the default set '0' to 'S' case...
        *      If EXIT-USER-SEV = 12 Then
        *        Compute EXIT-RETURNCODE = 4
        *      Else
        *        Compute EXIT-RETURNCODE = 0
        *      End-If

      End-Evaluate

      END PROGRAM IGYMSGXT.

```

出口モジュールでのエラー処理

ユーザー出口の処理中に、以下に記述した条件が発生することがあります。

出口ロードの失敗:

ユーザー出口の LOAD 要求が失敗すると、メッセージ IGYSI5207-U がオペレーター宛てに書き込まれます。

```
An error occurred while attempting to load user exit exit-name.
```

出口オープンの失敗:

ユーザー出口の OPEN 要求が失敗すると、メッセージ IGYSI5208-U がオペレーター宛てに書き込まれます。

```
An error occurred while attempting to open user exit exit-name.
```

PRTEXT PUT の失敗:

- メッセージ IGYSI5203-U がリストに書き込まれます。

```
A PUT request to the PRTEXT user exit failed with return code nn.
```

- メッセージ IGYSI5217-U がオペレーター宛てに書き込まれます。

```
An error occurred in PRTEXT user exit exit-name. Compiler terminated.
```

SYSIN GET の失敗:

以下のメッセージがリストに書き込まれることがあります。

- IGYSI5204-U:

```
The record address was not set by the exit-name user exit.
```

- IGYSI5205-U:

```
A GET request from the INEXIT user exit failed with return code nn.
```

- IGYSI5206-U:

```
The record length was not set by the exit-name user exit.
```

ADEXIT PUT の失敗:

- メッセージ IGYSI5225-U がオペレーター宛てに書き込まれます。

```
An error occurred in ADEXIT user exit exit-name. Compiler terminated.
```


- メッセージ IGYSI5226-U がリストに書き込まれます。

A PUT request to the ADEXIT user exit failed with return code *nn*.

MSGEXIT の失敗:

カスタマイズの失敗: サポートされていない重大度変更、またはサポートされていないメッセージ抑制が試行されると、メッセージ IGYPP5293-U がリストに書き込まれます。

MSGEXIT user exit *exit-name* specified a message severity customization that is not supported. The message number, default severity, and user-specified severity were: *mm*, *ds*, *us*. Change MSGEXIT user exit *exit-name* to correct this error.

一般障害: MSGEXIT モジュールが 4 以外の非ゼロ値を戻りコードに設定した場合、メッセージ IGYPP5064-U がリストに書き込まれます。

A call to the MSGEXIT user exit routine *exit-name* failed with return code *nn*.

MSGEXIT メッセージで、*PP* という 2 文字は、MSGEXIT モジュールを呼び出すことになったメッセージを出したコンパイラー・フェーズを示します。

関連タスク

895 ページの『コンパイラー・メッセージの重大度のカスタマイズ』

CICS、SQL および SQLIMS の各ステートメントで EXIT コンパイラー・オプションを使用する

のサブオプションを使用してコンパイルすると、EXIT コンパイラオプション、およびプログラムに EXEC CICS、EXEC SQL、または EXEC SQLIMS ステートメントでは、終了モジュールで実行できるアクションは、別の CICS 翻訳者と Db2 プリコンパイラ、または統合された CICS 翻訳者と Db2 コプロセッサ。

このタスクについて

プログラムに EXEC SQLIMS ステートメントが含まれている場合、出口モジュールで実行できるアクションは、組み込みの変換プログラムについてリストされるアクションです。

次の表では、組み込みの変換プログラムを使用するか、または分離型の変換プログラムを使用するかによって、出口モジュールで可能なアクションを示しています。

表 128. CICS、SQL および SQLIMS ステートメントに対して出口モジュールで可能なアクション			
サブオプションを使用したコンパイラ	統合された、または分離した CICS および Db2 の変換プログラムによって変換されているか	可能なアクション	コメント
INEXIT	組み込み	INEXIT モジュールでは、EXEC CICS、EXEC SQL、および EXEC SQLIMS ステートメントを処理できます	INEXIT モジュールでは、EXEC ステートメントに生成された COBOL ステートメントの制御が得られません。
	分離型	INEXIT モジュールでは、EXEC ステートメントに生成された COBOL ステートメントを処理できます	生成されたステートメントは INEXIT モジュールで変更することができます。ただし、この変更について、IBM ではサポートしません。

表 128. CICS、SQL および SQLIMS ステートメントに対して出口モジュールで可能なアクション (続き)			
サブオプションを使用したコンパイラ	統合された、または分離した CICS および Db2 の変換プログラムによって変換されているか	可能なアクション	コメント
LIBEXIT	組み込み	LIBEXIT モジュールで、EXEC SQL INCLUDE そして EXEC SQLIMS INCLUDE 声明。処理可能 EXEC CICS LIBEXIT モジュール内のソースステートメント。	EXEC SQL INCLUDE そして EXEC SQLIMS INCLUDE 文は COBOL のように処理される COPY 声明。
	分離型	LIBEXIT モジュールでは、EXEC CICS ステートメントに生成された COBOL ステートメントを処理できます	EXEC SQL INCLUDE および SQLIMS INCLUDE ステートメントによってもたらされた入力ステートメントを処理できるのは、INEXIT サブオプションを使用した場合のみです。
PRTEXT	組み込み	PRTEXT モジュールにおいて SOURCE リストにある EXEC CICS、EXEC SQL、および EXEC SQLIMS ソース・ステートメントを処理できます	PRTEXT モジュールは、生成された COBOL ステートメントにアクセスできません。
	分離型	PRTEXT モジュールでは、EXEC ステートメントに生成された COBOL SOURCE リスト・ステートメントを処理できます	
ADEXIT	組み込み	ADEXIT モジュールでは、EXEC CICS、EXEC SQL、および EXEC SQLIMS ソース・ステートメントを処理できます	ADEXIT モジュールは、生成された COBOL ステートメントにアクセスできません。
	分離型	ADEXIT モジュールでは、EXEC ステートメントに生成された COBOL SYSADATA ステートメントを処理できます	
MSGEXIT	組み込み	MSGEXIT モジュールで CICS および Db2 メッセージを処理できます	
	分離型	MSGEXIT モジュールで CICS および Db2 メッセージを処理できません	CICS からのメッセージは、分離型の CICS 変換プログラム・リストに表示されています。Db2 からのメッセージは Db2 プリコンパイラ・リストに表示されています。

関連概念

[551 ページの『組み込みの CICS 変換プログラム』](#)

[557 ページの『Db2 コプロセッサ』](#)

関連タスク

[549 ページの『CICS オプションを使用したコンパイル』](#)

[561 ページの『SQL オプションを使用したコンパイル』](#)

関連参照

[887 ページの『INEXIT の処理』](#)

[888 ページの『LIBEXIT の処理』](#)

[891 ページの『PRTEXIT の処理』](#)

[892 ページの『ADEXIT の処理』](#)

[893 ページの『MSGEXIT の処理』](#)

JNI.cpy コピーブック

このリスト JNI.cpy コピーブックが示しています。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 はの中に z/OS UNIX ファイルシステムの中に includeCOBOL インストールディレクトリのサブディレクトリ（通常/usr/lpp/cobol/include）。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.

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.
02 NewDirectByteBuffer	function-pointer.
02 GetDirectBufferAddress	function-pointer.
02 GetDirectBufferCapacity	function-pointer.
02 GetObjectRefType	function-pointer.
02 GetModule	function-pointer.

関連タスク

[321 ページの『z/OS ユニックスのもとでのオブジェクト指向アプリケーションのコンパイル』](#)

[785 ページの『JNI サービスへのアクセス』](#)

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、追加のプログラム・データを含んだファイル (SYSADATA 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。

[912 ページの『例: SYSADATA』](#)

関連参照

[378 ページの『ADATA』](#)

[910 ページの『SYSADATA ファイルに影響するコンパイラ・オプション』](#)

[911 ページの『SYSADATA レコード・タイプ』](#)

[913 ページの『SYSADATA レコード記述』](#)

SYSADATA ファイルに影響するコンパイラ・オプション

いくつかのコンパイラ・オプションは、SYSADATA ファイルの内容に影響を与える可能性があります。

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。

LANGUAGE

LANGUAGE は、メッセージ・テキスト (大文字英語、英大 / 小文字混合、または日本語) を制御します。日本語を選択すると、そのために DBCS 文字がエラー識別レコードに書き込まれることになる場合があります。

NUM

NUM を使用すると、コンパイラは、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。

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。

以下の表に示されている 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
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、FLAGSTD または 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。

[912 ページの『例: SYSADATA』](#)

関連参照

[911 ページの『SYSADATA レコード・タイプ』](#)

[388 ページの『COMPILE』](#)

[399 ページの『出口』](#)

[413 ページの『LANGUAGE』](#)

[422 ページの『番号』](#)

[448 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 129. SYSADATA レコード・タイプ	
レコード・タイプ	アクション
916 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
917 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
917 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
917 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
930 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
932 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
959 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
985 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
985 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
986 ページの『COPY REPLACING レコード: X'0039'』	一致の結果としてのテキスト置換のインスタンスを説明します。COPY. . .REPLACINGoperand-1 コピーブックのテキスト付き
986 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。

表 129. SYSADATA レコード・タイプ (続き)	
レコード・タイプ	アクション
999 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
1001 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
1001 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するライブラリー・ファイルとメンバーを記述します
1002 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
1002 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

タイプ	説明
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

913 ページの『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

914 ページの『共通ヘッダー・セクション』

916 ページの『ジョブ識別レコード: X'0000』

917 ページの『ADATA 識別レコード: X'0001』

917 ページの『コンパイル単位の開始|終了レコード: X'0002』

917 ページの『オプション・レコード: X'0010』

930 ページの『外部シンボル・レコード: X'0020』

932 ページの『構文解析ツリー・レコード: X'0024』

959 ページの『トークン・レコード: X'0030』

985 ページの『ソース・エラー・レコード: X'0032』

985 ページの『ソース・レコード: X'0038』

986 ページの『COPY REPLACING レコード: X'0039』

986 ページの『記号レコード: X'0042』

999 ページの『記号相互参照レコード: X'0044』

1001 ページの『ネストされたプログラム・レコード: X'0046』

1001 ページの『ライブラリー・レコード: X'0060』

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・ユーティリティーによって、ストリップされます。

表 130. 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 High Level Assembler
		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 130. **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・レベル	XL1	3 ヘッダー構造の定義レベル
フラグ	XL1	1. 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

表 130. SYSADATA 共通ヘッダー・セクション (続き)		
フィールド	サイズ	説明
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 131. 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
BLD レベル	CL8	関連データ・ファイルを生成したプロダクトのビルド・レベル情報 (形式: PYYMMDD)
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・プロシージャーの MVS プロシージャー・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
...入力ファイル名の長さ	HL2	次の入力ファイル名の長さ
...ボリュームシリアル番号の長さ	HL2	ボリューム通し番号の長さ
...メンバー名の長さ	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

表 131. SYSADATA ジョブ識別レコード (続き)		
フィールド	サイズ	説明
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。次のレコードには、入力ファイルの残りが含まれます。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 132. 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。
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 133. 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。以下のオプションのどちらかを使用します。 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 134. **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	1... ビット 1 = COPYLOC、ビット 0 = NOCOPYLOC .1... ビット 1 = DEFINE、ビット 0 = NODEFINE ..1. ビット 1 = SOURCE (HEX)、ビット 0 = SOURCE (DEC) または NOSOURCE ...1 ビット 1 = INVDATA、ビット 0 = NOINVDATA 1... ビット 1 = JAVAIOPI、ビット 0 = NOJAVAIOPI 1. ビット 1 = CONDCOMP (SKIPSR)、ビット 0 = CONDCOMP (NOSKIPSR) 11.1 将来の利用のために予約済み
オプション・バイト 1	XL1	1... ビット 1 = DECK、ビット 0 = NODECK .1... ビット 1 = ADATA、ビット 0 = NOADATA ..1. ビット 1 = COLLSEQ (EBCDIC)、ビット 0 = COLLSEQ (LOCALE BINARY) (AIX のみ) ...1 ビット 1 = SEPOBJ、ビット 0 = NOSEPOBJ (AIX のみ) 1... ビット 1 = NAME、ビット 0 = NONAME 1.. ビット 1 = OBJECT、ビット 0 = NOOBJECT 1. ビット 1 = SQL、ビット 0 = NOSQL 1 ビット 1 = CICS、ビット 0 = NOCICS

表 134. **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 2	XL1	1... ビット 1 = OFFSET、ビット 0 = NOOFFSET .1... ビット 1 = MAP、ビット 0 = NOMAP ..1. ビット 1 = LIST、ビット 0 = NOLIST ...1 ビット 1 = DBCSXREF、ビット 0 = NODBCSXREF 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。 XREF (FULL) は、このフラグがオフであり、ビット 7 のフラグ がオンであることによって示されます。 1.. ビット 1 = SOURCE、ビット 0 = NOSOURCE 1. ビット 1 = VBREF、ビット 0 = NOVBREF 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。
オプション・バイト 3	XL1	1... ビット 1 = 指定された FLAG 組み込み診断レベル (値 Y は FLAG(X,Y) のように指定される) .1... ビット 1 = FLAGSTD、ビット 0 = NOFLAGSTD ..1. ビット 1 = NUM、ビット 0 = NONUM ...1 ビット 1 = SEQUENCE、ビット 0 = NOSEQUENCE 1... ビット 1 = SOSI、ビット 0 = NOSOSI (AIX のみ) 1.. ビット 1 = NSYMBOL (NATIONAL)、ビット 0 = NSYMBOL (DBCS) 1. ビット 1 = PROFILE、ビット 0 = NOPROFILE (AIX のみ) 1 ビット 1 = WORD、ビット 0 = NOWORD

表 134. **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 4	XL1	1... .. ビット 1 = ADV、ビット 0 = NOADV .1... .. ビット 1 = APOST、ビット 0 = QUOTE ..1. ビット 1 = DYNAM、ビット 0 = NODYNAM ...1 ビット 1 = AWO、ビット 0 = NOAWO 1... ビット 1 = RMODE 指定済み、ビット 0 = RMODE (AUTO) 1.. ビット 1 = RENT、ビット 0 = NORENT 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。 1 ビット 1 = RMODE (24)、ビット 0 = RMODE (ANY)
オプション・バイト 5	XL1	1... .. ビット 1 = SQLCCSID、ビット 0 = NOSQLCCSID .1... .. ビット 1 = OPT (1 2)、ビット 0 = OPT (0) ..1. ビット 1 = SQLIMS、ビット 0 = NOSQLIMS ...1 ビット 1 = DBCS、ビット 0 = NODBCS 1... ビット 1 = AFP (VOLATILE)、ビット 0 = AFP (NOVOLATILE) 1.. ビット 1 = SSRANGE、ビット 0 = NOSSRANGE 1. ビット 1 = TEST、ビット 0 = NOTEST 1 ビット 1 = PROBE、ビット 0 = NOPROBE (Windows のみ)

表 134. 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 6	XL1	1... ビット 1 = SRCFORMAT(EXTEND)、ビット 0 = SRCFORMAT(COMPAT) ..1. ビット 1 = NUMPROC(PFD)、ビット 0 = NUMPROC(NOPFD) ...1 ビット 1 = NUMCLS(ALT)、ビット 0 = NUMCLS(PRIM) 1.. ビット 1 = BINARY(S390)、ビット 0 = BINARY(NATIVE) (AIX のみ) 1. ビット 1 = TRUNC(STD)、ビット 0 = TRUNC(OPT) 1 ビット 1 = ZWB、ビット 0 = NOZWB .1.. 1... 将来の利用のために予約済み
オプション・バイト 7	XL1	1... ビット 1 = ALLOWCBL、ビット 0 = NOALLOWCBL .1.. ビット 1 = TERM、ビット 0 = NOTERM ..1. ビット 1 = DUMP、ビット 0 = NODUMP 1. ビット 1 = CURRENCY、ビット 0 = NOCURRENCY ...1 11.1 将来の利用のために予約済み
オプション・バイト 8	XL1	1... ビット 1 = RULES、ビット 0 = NORULES .1.. ビット 1 = OPTFILE、ビット 0 = OPTFILE ではない ..1. ビット 1 = ADDR(64)、ビット 0 = ADDR(32) (AIX のみ) ビット 1 = LP(64)、ビット 0 = LP(32) (z/OS のみ) 1... ビット 1 = BLOCK0、ビット 0 = NOBLOCK0 1. ビット 1 = DISPSIGN(SEP)、ビット 0 = DISPSIGN(COMPAT) 1 ビット 1 = STGOPT、ビット 0 = NOSTGOPT ...1 .1.. 将来の利用のために予約済み

表 134. **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 9	XL1	1... ビット 1 = DATA(24)、ビット 0 = DATA(31) .1... ビット 1 = FASTSRT、ビット 0 = NOFASTSRT 1.. ビット 1 = THREAD、ビット 0 = NOTHREAD ...1 1.11 将来の利用のために予約済み
オプション・バイト A	XL1	1... ビット 1 = HGPR(PRESERVE)、ビット 0 = HGPR(NOPRESERVE) .1... ビット 1 = XMLPARSE(XMLSS)、ビット 0 = XMLPARSE(COMPAT) ..1. ビット 1 = MAP(DEC)、ビット 0 = MAP(HEX) ...1 ビット 1 = SPOPT、ビット 0 = NOSPOPT 1.. ビット 1 = SUPPRESS、ビット 0 = NOSUPPRESS 1. ビット 1 = VSAMOPENFS(SUCC)、ビット 0 = VSAMOPENFS(COMPAT) 1..1 将来の利用のために予約済み
オプション・バイト B	XL1	1111 1111 将来の利用のために予約済み
オプション・バイト C	XL1	1... ビット 1 = NCOLLSEQ(LOCALE) (AIX のみ) .1... 将来の利用のために予約済み ..1. ビット 1 = INTDATE(LILIAN)、ビット 0 = INTDATE(ANSI) ...1 ビット 1 = NCOLLSEQ(BINARY) (AIX のみ) 1... ビット 1 = ZACHECK 1.. ビット 1 = FLOAT(HEX)、ビット 0 = FLOAT(NATIVE) (AIX のみ) 1. ビット 1 = COLLSEQ(BINARY) (AIX のみ) 1 ビット 1 = COLLSEQ(LOCALE) (AIX のみ)

表 134. **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	1... .. ビット 1 = DLL、ビット 0 = NODLL .1... .. ビット 1 = EXPORTALL、ビット 0 = NOEXPORTALL ..1. ビット 1 = CODEPAGE 1 ビット 1 = SOURCEFORMAT(EXTEND)、ビット 0 = SOURCEFORMAT(COMPAT) (AIX のみ) 1... ビット 1 = INITCHECK、ビット 0 = NOINITCHECK 1.. ビット 1 = INLINE、ビット 0 = NOINLINE 1. ビット 1 = WSCLEAR、ビット 0 = NOWSCLEAR (AIX のみ) 1 ビット 1 = BEOPT、ビット 0 = NOBEOPT (AIX のみ)
オプション・バイト E	XL1	1..... ビット 1 = VLR(COMPAT)、ビット 0 = VLR(STANDARD) .1... .. ビット 1 = DIAGTRUNC、ビット 0 = NODIAGTRUNC 1.. ビット 1 = LSTFILE(UTF-8)、ビット 0 = LSTFILE(LOCALE) (AIX のみ) 1. ビット 1 = MDECK、ビット 0 = NOMDECK 1 ビット 1 = MDECK(NOCOMPILE) ..11 1... 将来の利用のために予約済み

表 134. **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 F	XL1	1... .. ビット 0 = INITIAL、ビット 1 = NOINITIAL .1... .. ビット 0 = COPYRIGHT、ビット 1 = NOCOPYRIGHT ..1... .. ビット 0 = QUALIFY (COMPAT)、ビット 1 = QUALIFY (EXTEND) ...1... .. ビット 0 = NOSERVICE、ビット 1 = SERVICE 1... ビット 1 = ZONEDATA (MIG) 1.. ビット 1 = ZONEDATA (NOPFD) 1. ビット 0 = NONUMCHECK、ビット 1 = NUMCHECK (ABD MSG BIN PAC ZON) 1 ビット 0 = NOPARMCHECK、ビット 1 = PARMCHECK (ABD MSG)
オプション・バイト G	XL1	1... .. ビット 0 = NUMCHECK (NOZON)、ビット 1 = NUMCHECK (ZON) .1... .. ビット 0 = NUMCHECK (NOPAC)、ビット 1 = NUMCHECK (PAC) ..1... .. ビット 0 = NUMCHECK (NOBIN)、ビット 1 = NUMCHECK (BIN) ...1... .. ビット 0 = NUMCHECK (ABD)、ビット 1 = NUMCHECK (MSG) 1... ビット 0 = NUMCHECK (ZON (ALPHNUM))、ビット 1 = NUMCHECK (ZON (NOALPHNUM)) 1.. ビット 0 = NUMCHECK (ZON (STRICT))、ビット 1 = NUMCHECK (ZON (LAX)) 1. ビット 0 = NUMCHECK (BIN (TRUNCBIN))、ビット 1 = NUMCHECK (BIN (NOTRUNCBIN)) 1 将来の利用のために予約済み
オプション・バイト H	XL1	1... .. ビット 0 = PARMCHECK (MSG)、ビット 1 = PARMCHECK (ABD) (PARMCHECK が有効な場合) .111 1111 将来の利用のために予約済み

表 134. **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 I	XL1	1... ビット 1 = RULES(ENDPERIOD)、ビット 0 = RULES(NOENDPERIOD) .1... ビット 1 = RULES(EVENPACK)、ビット 0 = RULES(NOEVENTPACK) ..1. ビット 1 = RULES(LAXPERF)、ビット 0 = RULES(NOLAXPERF) ...1 ビット 1 = RULES(SLACKBYTES)、ビット 0 = RULES(NOSLACKBYTES) 1... ビット 1 = RULES(OMITODOMIN)、ビット 0 = RULES(NOOMITODOMIN) 1.. ビット 1 = RULES(UNREFALL)、ビット 0 = RULES(NOUNREFALL) 1. ビット 1 = RULES(UNREFSRC)、ビット 0 = RULES(NOUNREFSRC) 1 ビット 1 = RULES(LAXREDEF)、ビット 0 = RULES(NOLAXREDEF)
オプション・バイト J	XL1	1... ビット 1 = SSRANGE(ZLEN,x)、ビット 0 = SSRANGE(NOZLEN,x) .1... ビット 1 = SSRANGE(x,ABD)、ビット 0 = SSRANGE(x,MSG) ..11 1111 将来の利用のために予約済み
オプション・バイト K	XL1	1... ビット 1 = INVDATA(FORCENUMCMP,x) .1.. .1.. ビット 1 = INVDATA(NOCLEANSIGN,x)
オプション・バイト L	XL1	1... ビット 1 = JAVAIO(OUTPATH,x,x) ..1. ビット 1 = JAVAIO(x,x,JAVA64)

表 134. **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' Flag(I) X'04' Flag(W) X'08' Flag(E) X'0C' Flag(S) X'10' Flag(U) X'FF' Noflag
組み込み診断レベル	XL1	X'00' Flag(I) X'04' Flag(W) X'08' Flag(E) X'0C' Flag(S) X'10' Flag(U) X'FF' Noflag
FLAGSTD (FIPS) 指定	XL1	1... .. 最小 .1.. .. 中間 ..1. 高 ...1 IBM extensions 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

表 134. **SYSADATA** オプション・レコード (続き)

フィールド	サイズ	説明
コンパイラ・モード	XL1	X'00' 無条件 Noccompile、Nocompile(I) X'04' Nocompile(W) X'08' Nocompile(E) X'0C' Nocompile(S) X'FF' Compile
スペース値	CL1	
3 値オプション用のデータ	XL1	1... .. NAME (ALIAS) 指定 .1... .. 将来の利用のために予約済み ..1. TRUNC (BIN) 指定 ...1 PARMCHECK (ABD) (PARMCHECK が有効な場合) 1... INITCHECK (STRICT) 指定 111 将来の利用のために予約済み
TEST サブオプション	XL1	1... .. TEST (EJPD) .1... .. TEST (SOURCE) ..1. TEST (SEPARATE) ...1 TEST NOTEST (DWARF) 1... TEST (SEPARATE (DSNAME)) 111 Reserved for future use
OUTDD 名長	HL2	OUTDD 名の長さ
RWT ID 長	HL2	予約語テーブル ID の長さ
BLD LEVEL	CL8	製品ビルド・レベル情報 (形式 PYYMMDD)

表 134. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
PGMNAME サブオプション	XL1	1... .. ビット 1 = PGMNAME (COMPAT) .1... .. ビット 1 = PGMNAME (LONGUPPER) ..1. ビット 1 = PGMNAME (LONGMIXED) ...1 1111 将来の利用のために予約済み
記入項目インターフェース・サブオプション	XL1	1... .. ビット 1 = EntryInterface(System) (Windows のみ) .1... .. ビット 1 = EntryInterface(OptLink) (Windows のみ) ..11 1111 将来の利用のために予約済み
CALLINTERFACE サブオプション	XL1	1... .. ビット 1 = CALLINTERFACE(DLL) .1... .. ビット 1 = CALLINTERFACE(DYNAMIC) ..11 1111 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	CL5	将来の利用のために予約済み
最適化レベル	XL1	最適化レベル $0 \leq n \leq 2$
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	このフィールドは、LANGUAGE オプションの 2 文字の省略形 (EN、UE、JA、または JP のいずれか) を保持します。
INEXIT 名長	HL2	SYSIN ユーザー出口名の長さ
PRTEXT 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

表 134. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
ADEXIT 名長	HL2	ADATA ユーザー出口名の長さ
CURROPT	CL5	CURRENCY オプション値
ARCH	XL1	ARCH レベル番号
TUNE	XL1	TUNE レベル番号
予約済み	CL1	将来の利用のために予約済み
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
BUFSIZE	FL4	BUFSIZE 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 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 IGYCDIAG 1 ビット 1 = ユーザー領域内の IGYCDGEN 1... 11.. 将来の利用のために予約済み

表 134. SYSADATA オプション・レコード (続き)		
フィールド	サイズ	説明
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	CL4	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
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 135. 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	X'00' PROGRAM-ID 名 (メインエントリー・ポイント名) X'01' ENTRY 名 (2 次エントリー・ポイント名) X'02' 外部参照 (参照先の外部エントリー・ポイント) X'04' COBOL の場合は適用外 X'05' COBOL の場合は適用外 X'06' COBOL の場合は適用外 X'0A' COBOL の場合は適用外 X'12' 内部参照 (参照先の内部サブプログラム) X'C0' 外部クラス名 (OO COBOL クラス定義) X'C1' METHOD-ID 名 (OO COBOL メソッド定義) X'C6' メソッド参照 (OO COBOL メソッド参照) X'FF' COBOL の場合は適用外 X'D0' FUNCTION-ID 名 タイプ X'12'、X'C0'、X'C1' および X'C6' は COBOL 専用です。
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
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。

注: コンパイラーは同じノードに対して複数の SYSADATA レコードを出力することがある。この場合、2 つのレコードのデータを組み合わせる必要がある。例えば、OCCURS や RECORD IS VARYING の DEPENDING ON オブジェクトのノードがそうである。2 番目のレコードには通常、最初のレコードの SYMBOL-ID が追加される。

表 136. 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 004 ユーザー定義関数 005 関数のプロトタイプ
		101 IDENTIFICATION DIVISION 102 ENVIRONMENT DIVISION 103 DATA DIVISION 104 PROCEDURE DIVISION 105 終了プログラム／方法／クラス／機能
		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

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		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 512 関数名 513 関数属性
		601 記述 602 データ記述節 603 データ入力項目 604 ファイル記述節 605 データ入力項目名 606 データ入力項目レベル 607 EXEC 記入項目

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		701 EVALUATE サブジェクト句 702 EVALUATE WHEN 句 703 EVALUATE WHEN OTHER 句 704 SEARCH WHEN 句 705 INSPECT CONVERTING 句 706 INSPECT REPLACING 句 707 INSPECT TALLYING 句 708 PERFORM UNTIL 句 709 PERFORM VARYING 句 710 PERFORM AFTER 句 711 ステートメント・ブロック 712 範囲終了符号 713 INITIALIZE REPLACING 句 714 EXEC CICS コマンド 715 INITIALIZE WITH FILLER 716 INITIALIZE TO VALUE 717 INITIALIZE TO DEFAULT 718 ALLOCATE INITIALIZED 719 ALLOCATE LOC 720 DATA DIVISION 句

表 136. SYSADATA 構文解析ツリー・レコード (続き)		
フィールド	サイズ	説明
		801 句 802 ON 句 803 NOT 句 804 THEN 句 805 ELSE 句 806 条件 807 式 808 相対索引付け 809 EXEC CICS オプション 810 予約語 811 INITIALIZE REPLACING カテゴリー

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		901 セクションまたは段落名 902 ID 903 英字名 904 クラス名 905 条件名 906 ファイル名 907 指標名 908 簡略名 910 シンボリック文字 911 リテラル 912 関数 ID 913 データ名 914 特殊レジスター 915 プロシーチャー参照 916 算術演算子 917 全プロシーチャー 918 INITIALIZE リテラル (トークンなし) 919 ALL リテラルまたは形象定数 920 キーワード・クラス・テスト名 921 ID レベルの予約語 922 単項演算子 923 比較演算子

表 136. SYSADATA 構文解析ツリー・レコード (続き)		
フィールド	サイズ	説明
		1001 添え字 1002 参照変更
ノード・サブタイプ	HL2	ノードのサブタイプ。 セクション・タイプの場合: 0001 「構成 (Configuration)」 セクション 0002 INPUT-OUTPUT セクション 0003 FILE SECTION 0004 WORKING-STORAGE SECTION 0005 LINKAGE SECTION 0006 LOCAL-STORAGE SECTION 0007 REPOSITORY セクション

表 136. **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 0015 FUNCTION-ID 段落

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		ENVIRONMENT DIVISION 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 0012 XML-SCHEMA

表 136. **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 利用方法 0011 価値 0012 VOLATILE 0013 DYNAMIC LENGTH 0023 GLOBAL 0024 EXTERNAL 0025 TYPEDEF 0026 タイプ

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		ファイル記述文節タイプの場合: 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

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		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 136. **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 追加
		0005 ALTER
		0006 CALL
		0007 CANCEL
		0008 CLOSE
		0009 COMPUTE
		0010 CONTINUE
		0011 削除
		0012 表示
		0013 DIVIDE (INTO)
		0113 DIVIDE (BY)
		0014 ENTER
		0015 エントリー
		0016 EVALUATE
		0017 出口
		0018 GO
		0019 GOBACK
		0020 IF
		0021 INITIALIZE
		0022 INSPECT

表 136. **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 サービス 0037 SET

表 136. SYSADATA 構文解析ツリー・レコード (続き)		
フィールド	サイズ	説明
		0038 SORT 0039 開始 0040 「STOP」 0041 STRING 0042 SUBTRACT 0043 UNSTRING 0044 EXEC SQL 0144 EXEC CICS 0045 WRITE 0046 XML 0047 ALLOCATE 0048 無料 0049 JSON

表 136. **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. .すべて 0005 INSPECT. .LEADING 0006 セット。 。 。 今後 0007 セット。 。 。 UP 0008 セット。 。 。 DOWN 0009 実行する。 。 。 TIMES 0010 DIVIDE. .REMAINDER 0011 INSPECT. .FIRST 0012 検索。 。 。 VARYING 0013 MORE-LABELS 0014 SEARCH ALL 0015 検索。 。 。 AT END 0016 検索。 。 。 TEST INDEX 0017 GLOBAL 0018 ラベル

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		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 0031 END-OF-PAGE 0032 USING

表 136. **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 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 キー 0042 DELIMITER 0043 POINTER 0044 COUNT 0045 METHOD 0046 PROGRAM 0047 INPUT 0048 OUTPUT (出力) 0049 I-O 0050 EXTEND 0051 RELOAD 0052 ASCENDING 0053 DESCENDING

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		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 0066 XML PARSE VALIDATING 0067 XML GENERATE NAME 0068 XML GENERATE TYPE 0069 XML GENERATE SUPPRESS 0070 BYTE-LENGTH 0071 LIMIT

表 136. **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 ログ
		0003 MAX
		0004 最小
		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

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		0019 長さ
		0020 MEDIAN
		0021 NUMVAL
		0022 RANDOM
		0023 ANNUITY
		0024 整数
		0025 ORD-MAX
		0026 ORD-MIN
		0027 REVERSE
		0028 MIDRANGE
		0029 NUMVAL-C
		0030 VARIANCE
		0031 FACTORIAL
		0032 LOWER-CASE

表 136. **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 0049 DISPLAY-OF 0050 NATIONAL-OF 0051 UPOS 0052 UVALID 0053 UWIDTH 0054 ULENGTH 0055 USUBSTR

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		0056 USUPPLEMENTARY 0057 HEX-OF 0058 BIT-OF 0059 E 0060 TRIM 0061 PI 0062 ABS 0063 BYTE-LENGTH 0064 EXP 0065 EXP10 0066 BIT-TO-CHAR 0067 NUMVAL-F 0068 HEX-TO-CHAR 0069 SIGN 0070 TEST-NUMVAL 0071 TEST-NUMVAL-C 0072 TEST-NUMVAL-F

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		0073 UUID4 0074 FORMATTED-DATE 0075 FORMATTED-TIME 0076 TEST-DAY-YYYYDDD 0077 COMBINED-DATETIME 0078 FORMATTED-DATETIME 0079 TEST-DATE-YYYYMMDD 0080 SECONDS-PAST-MIDNIGHT 0081 FORMATTED-CURRENT-DATE 0082 TEST-FORMATTED-DATETIME 0083 INTEGER-OF-FORMATTED-DATE 0084 SECONDS-FROM-FORMATTED-TIME
		特殊レジスター・タイプの場合: 0001 ADDRESS OF 0002 LENGTH OF

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		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
		予約語タイプの場合: 0001 TRUE 0002 FALSE 0003 任意 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

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		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 0009 UTF-8
		プロシージャー名タイプの場合: 0001 SECTION 0002 PARAGRAPH

表 136. **SYSADATA** 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		ID レベルの予約語タイプの場合: 0001 ROUNDED 0002 TRUE 0003 オン 0004 オフ 0005 サイズ 0006 日付 0007 DAY 0008 DAY-OF-WEEK 0009 時刻 0010 WHEN-COMPILED 0011 PAGE 0012 DATE YYYYMMDD 0013 DAY YYYYDDD 0014 属性 0015 エレメント 0016 内容 0017 数字 0018 非数値 0019 間隔 0020 日時

表 136. **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 および 0015 または 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。

表 136. SYSADATA 構文解析ツリー・レコード (続き)		
フィールド	サイズ	説明
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。 - 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。
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 (/) が指定されているソース行
- 以下のコンパイラー指示ステートメント:
 - *CBL (*CONTROL)
 - BASIS
 - COPY
 - DELETE
 - EJECT
 - INSERT
 - REPLACE
 - SKIP1
 - SKIP2

- SKIP3
- TITLE
- デバッグ行。これは桁 7 に D が指定されているソース行です (WITH DEBUGGING MODE が指定されていない場合)。

表 137. 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。
トークン・コード	HL2	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、コンパイラー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。 0001 ACCEPT 0002 追加 0003 ALTER 0004 CALL 0005 CANCEL 0007 CLOSE 0009 COMPUTE 0011 削除 0013 表示 0014 DIVIDE 0017 READY 0018 END-PERFORM 0019 ENTER 0020 エントリー

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0021 出口 0022 EXEC EXECUTE 0023 GO 0024 IF 0025 INITIALIZE 0026 INVOKE 0027 INSPECT 0028 MERGE 0029 MOVE 0030 MULTIPLY 0031 OPEN 0032 PERFORM 0033 READ 0035 RELEASE 0036 RETURN 0037 REWRITE 0038 SEARCH 0040 SET

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0041 SORT 0042 開始 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

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		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 0070 EVALUATE 0071 RESET 0072 サービス 0073 END-INVOKE 0074 END-EXEC 0075 XML 0076 END-XML 0077 ALLOCATE 0078 無料 0079 JSON 0080 END-JSON

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0099 FOREIGN-VERB 0101 DATA-NAME 0105 DASHED-NUM 0106 10 進数 0107 DIV-SIGN 0108 EQ 0109 EXPONENTIATION 0110 GT 0111 整数 0112 LT 0113 LPAREN 0114 MINUS-SIGN 0115 MULT-SIGN 0116 NONUMLIT 0117 PERIOD 0118 PLUS-SIGN

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0121 RPAREN 0122 SIGNED-INTEGER 0123 QUID 0124 COLON 0125 IEOF 0126 EGCS-LIT 0127 COMMA-SPACE 0128 SEMICOLON-SPACE 0129 プロシージャー名。 0130 FLT-POINT-LIT 0131 言語環境 0132 GE 0133 IDREF 0134 EXPREF 0136 CICS 0137 新規 0138 NATIONAL-LIT 0139 UTF8-LIT 0140 クオレオペ

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0200 ADDRESS 0201 ADVANCING 0202 AFTER 0203 すべて 0204 ALPHABETIC 0205 ALPHANUMERIC 0206 任意 0207 および 0208 ALPHANUMERIC-EDITED 0209 BEFORE 0210 BEGINNING 0211 関数 0212 CONTENT 0213 CORR CORRESPONDING 0214 DAY 0215 日付 0216 DEBUG-CONTENTS 0217 DEBUG-ITEM 0218 DEBUG-LINE 0219 DEBUG-NAME 0220 DEBUG-SUB-1

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0221 DEBUG-SUB-2 0222 DEBUG-SUB-3 0223 DELIMITED 0224 DELIMITER 0225 DOWN 0226 NUMERIC-EDITED 0227 XML-EVENT 0228 END-OF-PAGE EOP 0229 EQUAL 0230 エラー 0231 XML-NTEXT 0232 EXCEPTION 0233 EXTEND 0234 FIRST 0235 FROM 0236 GIVING 0237 GREATER 0238 I-O 0239 IN 0240 INITIAL

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0241 INTO 0242 INVALID 0243 SQL 0244 LESS 0245 LINAGE-COUNTER 0246 XML-TEXT 0247 LOCK 0248 GENERATE 0249 NEGATIVE 0250 NEXT 0251 いいえ 0252 なし 0253 NUMERIC 0254 KANJI 0255 または 0256 OTHER 0257 OVERFLOW 0258 PAGE 0259 CONVERTING 0260 POINTER

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		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

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0280 THAN 0281 UNTIL 0282 UP 0283 UPON 0284 VARYING 0285 RELOAD 0286 TRUE 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

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0302 OCCURS 0303 SYNC SYNCHRONIZED 0304 MORE-LABELS 0305 JUST JUSTIFIED 0306 SHIFT-IN 0307 ブランク 0308 価値 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 0316 索引 0317 利用方法 0318 SIGN 0319 LEADING 0320 SEPARATE

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0321 INDEXED 0322 LEFT 0323 RIGHT 0324 PIC PICTURE 0325 値 0326 GLOBAL 0327 EXTERNAL 0328 BINARY 0329 PACKED-DECIMAL 0330 EGCS 0331 PROCEDURE-POINTER 0332 COMP-5 COMPUTATIONAL-5 0333 FUNCTION-POINTER 0334 タイプ 0335 JNIENVPTR 0336 NATIONAL 0337 GROUP-USAGE 0338 XML-NNAMESPACE 0339 XML-NAMESPACE 0340 XML-NNAMESPACE-PREFIX

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0341 XML-NAMESPACE-PREFIX 0342 VOLATILE 0343 UTF-8 0344 TYPEDEF 0401 HIGH-VALUE HIGH-VALUES 0402 LOW-VALUE LOW-VALUES 0403 QUOTE QUOTES 0404 SPACE SPACES 0405 ZERO 0406 ZEROES ZEROS 0407 ヌル NULLS 0501 BLOCK 0502 BOTTOM 0505 CHARACTER 0506 コード 0507 CODE-SET

表 137. SYSADATA トークン・レコード (続き)		
フィールド	サイズ	説明
		0514 FILLER 0516 FOOTING 0520 ラベル 0521 長さ 0524 LINAGE 0526 OMITTED 0531 RENAMES 0543 TOP 0545 TRAILING 0549 RECORDING 0500 POINTER-32 0551 BYTE-LENGTH 0555 LIMIT 0601 INHERITS 0603 RECURSIVE 0604 PROTOTYPE

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0701 ACCESS 0702 ALSO 0703 ALTERNATE 0704 領域 AREAS 0705 ASSIGN 0707 COLLATING 0708 COMMA 0709 CURRENCY 0710 CLASS 0711 DECIMAL-POINT 0712 DUPLICATES 0713 DYNAMIC 0714 EVERY 0716 MEMORY 0717 モード 0718 MODULES 0719 MULTIPLE 0720 NATIVE

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		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

表 137. SYSADATA トークン・レコード (続き)		
フィールド	サイズ	説明
		0741 APPLY 0742 WRITE-ONLY 0743 COMMON 0744 ALPHABET 0745 PADDING 0746 SYMBOLIC 0747 STANDARD-2 0748 OVERRIDE 0750 PASSWORD 0751 XML-SCHEMA

表 137. **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 0807 COPY 0808 COUNT 0809 DEBUGGING 0810 DEPENDING 0811 DESCENDING 0812 DIVISION 0814 FOR 0815 ORDER 0816 INPUT 0817 REPLACE 0818 キー 0819 LINE LINES 0820 XML-INFORMATION

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0821 OF 0822 オン 0823 OUTPUT (出力) 0825 レコード 0826 RECORDS 0827 REEL 0828 SECTION 0829 サイズ 0830 状況 0831 THROUGH THRU 0832 時刻 0833 TIMES 0834 今後 0836 UNIT 0837 USING 0838 WHEN 0839 WITH 0840 SQLIMS

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		0841 デフォルト 0900 DETAIL 0901 PROCEDURE 0902 DECLARATIVES 0903 END 1001 データ 1002 ファイル 1003 FD 1004 SD 1005 WORKING-STORAGE 1006 LOCAL-STORAGE 1007 LINKAGE

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		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 インストール 1205 DATE-WRITTEN 1206 DATE-COMPILED 1207 セキュリティー 1208 CLASS-ID 1209 METHOD-ID 1210 METHOD

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		1211 FACTORY 1212 OBJECT 1213 FUNCTION-ID 2020 TRACE 2046 SUPPRESS 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

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		5001 PARSE 5002 AUTOMATIC 5003 PREVIOUS 5004 ENCODING 5005 NAMESPACE 5006 NAMESPACE-PREFIX 5007 XML-DECLARATION 5008 ATTRIBUTES 5009 VALIDATING 5010 UNBOUNDED 5011 ATTRIBUTE 5012 エレメント 5013 NONNUMERIC 5014 名前 5015 CYCLE 5016 PARAGRAPH 5017 LOC 5018 INTRINSIC 5020 AS

表 137. **SYSADATA** トークン・レコード (続き)

フィールド	サイズ	説明
		5021 INITIALIZED 5022 BYTE-LENGTH 5023 FIXED 5024 BOOL 5025 BOOLEAN 5026 エンタリインタフェース 5027 DLL 5029 STATIC 5030 ENTRY-NAME 5031 COMPAT 5032 LONGMIXED 5033 LONGUPPER 5034 CODEPAGE 5035 無視 5036 指示 9999 COBOL の
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	トークンに関する情報は、以下を参照してください。 X'80' トークンは継続 X'40' 継続されるトークンの最後のピース 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。

表 137. SYSADATA トークン・レコード (続き)		
フィールド	サイズ	説明
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・ストリング

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 138. SYSADATA ソース・エラー・レコード		
フィールド	サイズ	説明
行番号	FL4	エラーのある行の行番号
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 139. 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 <i>n</i> 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 COPY BASIS 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 <i>n</i> 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。COPY BASIS 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 <i>n</i> 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 COPY BASIS 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 <i>n</i> フィールドを参照してください)。

表 139. SYSADATA ソース・レコード (続き)		
フィールド	サイズ	説明
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。

次の表は、COPY REPLACING 記録。

表 140. 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 141. 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。

表 141. SYSADATA 記号レコード (続き)		
フィールド	サイズ	説明
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。
記号タイプ	XL1	X'68' クラス名 (クラス ID) X'58' メソッド名 X'48' ユーザー定義関数 X'40' データ名 X'30' タイプ名 X'20' プロシージャ名。 X'10' 簡略名 X'08' プログラム名 X'81' 予約済み 以下は、上記タイプの ORed です (該当する場合)。 X'04' 外部 X'02' グローバル

表 141. **SYSADATA** 記号レコード (続き)

フィールド	サイズ	説明
記号属性	XL1	X'01' 数字 X'02' 次のいずれかのクラスの基本文字: • 英字 • 英数字 • DBCS • 国別 • UTF-8 X'03' グループ X'04' ポインター X'05' 指標データ項目 X'06' 指標名 X'07' 条件 X'0F' ファイル X'10' ソート・ファイル X'17' クラス名 (リポジトリ) X'18' オブジェクト参照 X'19' 通貨記号 X'1A' XML スキーマ名

表 141. **SYSADATA** 記号レコード (続き)

フィールド	サイズ	説明
節	XL1	記号定義で指定されている文節。 のためにシンボル属性が数値のシンボル (X'01')、基本文字 (X'02')、グループ (X'03')、ポインタ (X'04')、インデックスデータ項目 (X'05')、またはオブジェクト参照 (X'18'): 1... .. 値 .1... .. 索引付き ..1. 再定義 ...1 名前変更 1... 発生 1.. Occurs キーあり 1. ケースにより発生 1 親で発生 ファイル・タイプの場合: 1... .. 選択 .1... .. 割り当て ..1. 再実行 ...1 同一領域 1... 同一レコード域 1.. 記録モード 1. 予約済み 1 レコード

表 141. **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 コンソール

表 141. SYSADATA 記号レコード (続き)		
フィールド	サイズ	説明
		20 SYSIN SYSIPT 22 SYSOUT SYSLST SYSLIST 24 SYSPUNCH SYSPCH 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
データ・フラグ 1	XL1	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: 1... .. 再定義 .1... .. 名前変更 ..1. 同期化 ...1 暗黙的に再定義 1... 揮発性 1.. 暗黙の再定義 1. FILLER 1 レベル 77

表 141. **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 (の USAGE DISPLAY または USAGE NATIONAL) ..1. 内部浮動小数点 ...1 圧縮 1... 外部小数点 (の USAGE DISPLAY または USAGE NATIONAL) 1.. 負の位取り 1. 数値編集 (の USAGE DISPLAY または USAGE NATIONAL) 1 バイナリ (の USAGE COMP-5) 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 141. **SYSADATA** 記号レコード (続き)

フィールド	サイズ	説明
		ファイル・タイプの場合: 1... レコード内の ODO のオブジェクト .1... レコード内の ODO のサブジェクト ..1. 順次アクセス ...1 ランダム・アクセス 1... 動的アクセス 1.. 位置指定モード 1. レコード域 1 将来の利用のために予約済み
データ・フラグ 3	XL1	1... すべてのレコードが同じ長さ .1... 固定長 ..1. 可変長 ...1 未定義 1... スパン 1.. ブロック 1. 書き込みのみ適用 1 同一ソート・マージ領域

表 141. **SYSADATA** 記号レコード (続き)

フィールド	サイズ	説明
ファイル編成および属性	XL1	1... .. 物理順次 (ホスト上、QSAM) .1... .. ASCII ..1. 標準ラベル ...1 ユーザー・ラベル 1... 順次編成 1.. 索引編成 1. 相対編成 1 行順次
USAGE 節	FL1	X'00' USAGE IS DISPLAY X'01' USAGE IS COMP-1 X'02' USAGE IS COMP-2 X'03' USAGE IS PACKED-DECIMAL または USAGE IS COMP-3 X'04' USAGE IS BINARY、USAGE IS COMP、USAGE IS COMP-4、 または USAGE IS COMP-5 X'05' USAGE IS DISPLAY-1 X'06' USAGE IS POINTER X'07' USAGE IS INDEX X'08' USAGE IS PROCEDURE-POINTER X'09' USAGE IS OBJECT-REFERENCE X'0A' FUNCTION-POINTER X'0B' NATIONAL X'0C' POINTER-32 X'0D' UTF-8

表 141. SYSADATA 記号レコード (続き)		
フィールド	サイズ	説明
Sign 節	FL1	X'00' SIGN 節なし X'01' SIGN IS LEADING X'02' SIGN IS LEADING SEPARATE CHARACTER X'03' SIGN IS TRAILING X'04' SIGN IS TRAILING SEPARATE CHARACTER
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 141. **SYSADATA** 記号レコード (続き)

フィールド	サイズ	説明
ストレージ・タイプ	FL1	00 該当なし 01 ファイル 02 作業用ストレージ 03 リンケージ・セクション (Linkage Section) 05 特殊レジスター 07 変数別索引付き 10 UPSI スイッチ 13 可変位置項目 14 外部データ 15 英数字 FUNC 16 英数字 EVAL 17 オブジェクト・データ 19 ローカル・ストレージ 20 ファクトリー・データ 21 XML-TEXT と XML-NTEXT 22 TYPEDEF
データ・フラグ 6	XL1	各種シンボル・フラグ: 1... .. 動的長基本項目 .1... .. GROUP 動的長項目 ..1. 固定バイト長 UTF-8 項目

表 141. 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 4	XL1	数値 (X'01') の記号属性を持つ記号の場合: 1... .. 数値国別 基本文字 (X'02') の記号属性を持つ記号の場合: 1... .. 国別 .1... .. 国別編集 グループ (X'03') の記号属性を持つ記号の場合: 1... .. 国別グループ (Group-Usage National) .1... .. 無制限長グループ
データ・フラグ 5	XL1	OCCURS フラグ: 1... .. UNBOUNDED
ベース・ロケータ・セル	FL2	ベース・ロケータ・セル番号
シンボル ID	FL4	シンボルを識別する番号
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

表 141. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
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す。 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。
動的長項目制限	FL4	動的長項目に対する DYNAMIC LENGTH 節の LIMIT 句の値。これは、項目が動的長項目でなかったり LIMIT 句が指定されなかったりする場合はゼロ。
タイプ名 シンボル ID	FL4	TYPE 句で定義されたデータ項目の場合、これは対応する TYPEDEF のシンボル ID である。 グループタイプインスタンス内の従属アイテムの場合、関連するグループタイプ定義内の対応する従属アイテムのシンボル ID である。
予約済み	CL4	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
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、シンボル ID ASSIGN USING データ名; ゼロの場合 ASSIGN TO 指定された
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。何も指定しない場合はゼロ
記号名	CL(n)	

表 141. 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・ストリング 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 文字ストリング。文字ストリングには、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
鍵	(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
...フィラー	CL3	予約済み
...鍵 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
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、INITIAL VALUE このシンボルの句。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。
...最初の値の長さ	HL2	最初の値の長さ
...最初の値データ	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。
...2 番目の値の長さ	HL2	2 番目の値の長さ — THRU 値の組でない場合は、ゼロ
...2 番目の値データ	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 142. SYSADATA 記号相互参照レコード		
フィールド	サイズ	説明
記号名の長さ	HL2	記号名の文字数
ステートメント定義	FL4	シンボルが定義または宣言されている行番号 ステートメント XREF の場合のみ: ステートメント・カウント - 当該ステートメントに対する参照の総数。
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
相互参照タイプ	XL1	X'01' プログラム X'02' 手順 X'03' ステートメント 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)
...行番号	XL4	シンボルまたはステートメントが参照される行番号

表 142. SYSADATA 記号相互参照レコード (続き)

フィールド	サイズ	説明
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、ステートメントの場合には、ステートメント番号が 10 回発生します。 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。		

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 143. 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	シンボルが定義または宣言されている行番号
ネスト・レベル	XL1	プログラム・ネスト・レベル
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 144. 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

表 144. SYSADATA ライブラリー・レコード (続き)		
フィールド	サイズ	説明
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
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 145. 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 146. SYSADATA EVENTS TIMESTAMP レコード・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 147. SYSADATA EVENTS PROCESSOR レコード・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 148. SYSADATA EVENTS FILE END レコード・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

表 148. **SYSADATA EVENTS FILE END** レコード・レイアウト (続き)

フィールド	サイズ	説明
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 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 149. **SYSADATA EVENTS PROGRAM** レコード・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 150. **SYSADATA EVENTS FILE ID** レコード・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	
ブランク・セパレーター	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・セパレーター	CL1	

表 150. SYSADATA EVENTS FILE ID レコード・レイアウト (続き)		
フィールド	サイズ	説明
参照標識	XL1	
ブランク・セパレーター	CL1	
ソース・ファイル名の長さ	H2	
ブランク・セパレーター	CL1	
ソース・ファイル名	CL(n)	

表 151. SYSADATA EVENTS ERROR レコード・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・セパレーター	CL1	
改訂レベル	XL1	
ブランク・セパレーター	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・セパレーター	CL1	
Annot クラス	XL1	Annot クラス・メッセージの配置
ブランク・セパレーター	CL1	
エラー入力レコード番号	XL10	
ブランク・セパレーター	CL1	
エラー開始行番号	XL10	
ブランク・セパレーター	CL1	
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	

表 151. SYSADATA EVENTS ERROR レコード・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)	

サンプル・プログラムの使用

サンプル・プログラムはお手持ちのプロダクト・テープに収録されており、COBOL の多数の言語エレメントおよび概念を実例によって説明します。

この情報には以下の項目が含まれています。

- プログラムの概要 (2 つのサンプルに関するプログラム図表を含む)
- 入力データの形式とサンプル
- 作成される報告書のサンプル
- プログラムの実行方法についての情報
- 例示されている言語エレメントと概念のリスト

これらのプログラムに関する疑似コードとその他のコメントは、プログラムのプロローグに収められており、プログラム・リストで入手できます。

次の 3 つのサンプル・プログラムがあります。

- IGYTCARA は、QSAM ファイルと VSAM 索引付きファイルの使用例を提供し、多くの COBOL 組み込み関数の使用方法を示します。
- IGYTCARB は、IBM 対話式システム生産性向上機能 (ISPF) を使用する例です。
- IGYTSALE は、言語環境呼び出し可能サービスの機能のいくつかの使用例です。

関連概念

[1006 ページの『IGYTCARA: バッチ・アプリケーション』](#)

[1010 ページの『IGYTCARB: 対話式プログラム』](#)

[1013 ページの『IGYTSALE: ネストされたプログラム・アプリケーション』](#)

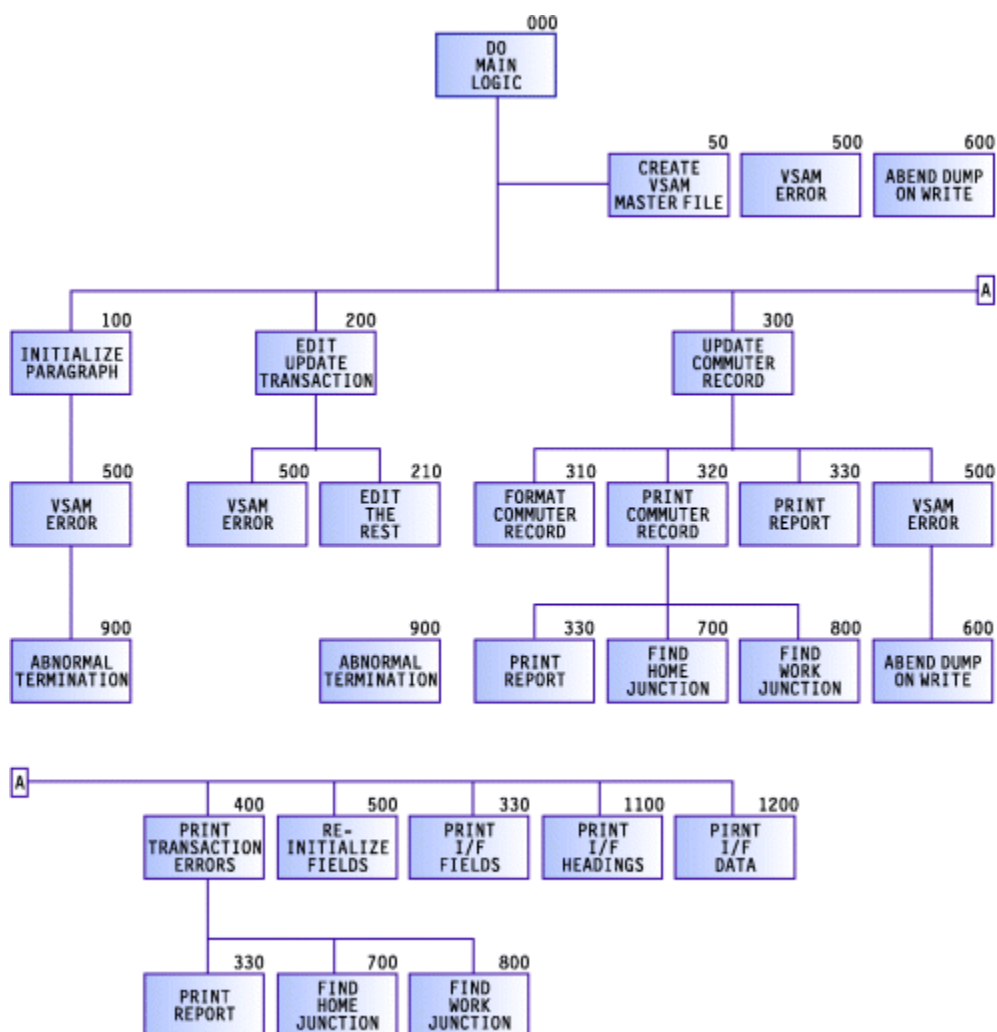
IGYTCARA: バッチ・アプリケーション

いくつかの地方事業所を持つ会社が従業員のための相乗り通勤を設定したいとします。アプリケーション IGYTCARA は、トランザクション・ファイル項目を妥当性検査し (QSAM 順次ファイル処理)、メインファイルを更新します (VSAM 索引付きファイル処理)。

このバッチ・アプリケーションでは、次の 2 つのタスクが実行されます。

- 同じ居住地域から同じ事業所まで同乗できる従業員の報告書を作成します
- 相乗り通勤データを更新します
 - 新しい従業員のデータを追加します
 - 関係する従業員の情報を変更します
 - 従業員レコードを削除します
 - 無効な更新要求をリストします

次の図は、アプリケーションの各部分とそれらがどのように編成されているかを示しています。



関連タスク

1009 ページの『IGYTCARA の実行準備』

関連参照

1007 ページの『IGYTCARA の入力データ』

1008 ページの『IGYTCARA によって作成される報告書』

1020 ページの『示されている言語エレメントおよび概念』

IGYTCARA の入力データ

プログラムへの入力として、会社は関連のある従業員から情報を集め、その情報をコーディングして、入力ファイルを作成しました。入力ファイルの形式の例 (フィールド相互間のスペースは、入力ファイルの場合と同様、省かれています) を示し、その後に各項目を説明しています。

A10111	ROBERTS	AB1021	CRYSTAL	COURTSAN	FRANCISCO	CA9990141555501904155551387H1W1D
↑↑↑↑	↑			↑	↑	↑↑↑↑↑
12 3 4	5			6	7	8 9 10 11

1. トランザクション・コード
2. シフト
3. ホーム・コード
4. 業務コード

- 通勤者名
- 自宅の住所
- 自宅の電話番号
- 勤務先電話
- 居住地域コード
- 勤務地域コード
- 運転状況コード

以下は、入力ファイルのセクションの例です。

```
A10111ROBERTS AB1021 CRYSTAL COURTSAN FRANCISCOCA9990141555501904155551387H1W1D
A20212KAHN DE789 EMILY LANE SAN FRANCISCOCA9992141555518904155552589H2W2D
P48899 99ASDFG0005557890123ASDFGHJ T
R10111ROBERTS AB1221 CRYSTAL COURTSAN FRANCISCOCA9990141555501904155551387H1W1D
A20212KAHN DE789 EMILY LANE SAN FRANCISCOCA9992141555518904155552589H2W2D
D20212KAHN DE
D20212KAHN DE
A20212KAHN DE789 EMILY LANE SAN FRANCISCOCA9992141555518904155552589H2W2D
A10111BONNICK FD1025 FIFTH AVENUE SAN FRANCISCOCA9990541555595904155557895H8W3
A10111PETERSON SW435 THIRD AVENUE SAN FRANCISCOCA9990541555546904155553717H3W4
. . .
```

IGYTCARA によって作成される報告書

IGYTCARA によって作成される出力報告書の最初のページを、以下のサンプルに示しています。実際の出力は、システムによって形式が多少変わることがあります。

1REPORT #: IGYTCAR1 COMMUTER FILE UPDATE LIST RUN DATE: 11/24/2003 PAGE #: 1									
-PROGRAM #: IGYTCAR1 RUN TIME: 01:40									
TRANS CODE	RE- CORD TYPE	SHIFT HOME CODE WORK CODE	COMMUTER NAME	HOME ADDRESS	HOME PHONE WORK PHONE	HOME LOCATION WORK LOCATION	JUNCTION	STA- TUS CODE	TRANS. ERROR
A	NEW	1 01 11	ROBERTS	AB 1021 CRYSTAL COURT SAN FRANCISCO	(415) 555-0190 (415) 555-1387	RODNEY/CRYSTAL BAYFAIR PLAZA		D	
A	NEW	2 02 12	KAHN	DE 789 EMILY LANE SAN FRANCISCO	(415) 555-1890 (415) 555-2589	COYOTE 14TH STREET/166TH AVENUE		D	
P		4 88 99			(000) 555-7890 99 ASDFG (123) ASD-FGHJ	HOME CODE ' ' NOT FOUND. WORK CODE ' ' NOT FOUND.		T	TRANSACTION CODE SHIFT CODE HOME LOC. CODE WORK LOC. CODE LAST NAME INITIALS ADDRESS CITY STATE CODE ZIPCODE HOME PHONE WORK PHONE HOME JUNCTION WORK JUNCTION DRIVING STATUS
R	OLD	1 01 11	ROBERTS	AB 1021 CRYSTAL COURT SAN FRANCISCO	(415) 555-0190 (415) 555-1387	RODNEY/CRYSTAL BAYFAIR PLAZA		D	
	NEW	1 01 11	ROBERTS	AB 1221 CRYSTAL COURT SAN FRANCISCO	(415) 555-0190 (415) 555-1387	RODNEY/CRYSTAL BAYFAIR PLAZA		D	
A		2 02 12	KAHN	DE 789 EMILY LANE SAN FRANCISCO	(415) 555-1890 (415) 555-2589	COYOTE 14TH STREET/166TH AVENUE		D	DUPLICATE REC.
D	OLD	2 02 12	KAHN	DE 789 EMILY LANE SAN FRANCISCO	(415) 555-1890 (415) 555-2589	COYOTE 14TH STREET/166TH AVENUE		D	
D		2 02 12	KAHN	DE					REC. NOT FOUND
A	NEW	2 02 12	KAHN	DE 789 EMILY LANE SAN FRANCISCO	(415) 555-1890 (415) 555-2589	COYOTE 14TH STREET/166TH AVENUE		D	
A	NEW	1 01 11	BONNICK	FD 1025 FIFTH AVENUE SAN FRANCISCO	(415) 555-9590 (415) 555-7895	RODNEY 17TH FREEWAY SAN LEANDRO			
A	NEW	1 01 11	PETERSON	SW 435 THIRD AVENUE	(415) 555-4690	RODNEY/THIRD AVENUE			

IGYTCARA の実行準備

IGYTCARA プログラム (IGYTCARA、IGYTCODE、および IGYTRANX) に必要なすべてのファイルは、プロダクト・インストール・テープで提供されます。これらのファイルは、IGY.V6R5M0.SIGYSAMP データ・セット内にあります。

このタスクについて

データ・セット名とプロシージャ名は、インストール時に変更することができます。これらの名前の妥当性について、システム・プログラマーにお問い合わせください。

IGYTCARA のソース・ファイルの中の、CBL ステートメントの以下のオプションは変更しないでください。

- NOADV
- NODYNAM
- NONAME
- NONUMBER
- QUOTE
- SEQUENCE

これらのオプションが有効な場合、プログラムは診断メッセージを出しません。ソース・ファイル内のシーケンス番号ストリングを使用すれば、使用されている言語エレメントを見つけることができます。

関連概念

[1006 ページの『IGYTCARA: バッチ・アプリケーション』](#)

関連タスク

[1009 ページの『IGYTCARA の実行』](#)

関連参照

[1007 ページの『IGYTCARA の入力データ』](#)

[1008 ページの『IGYTCARA によって作成される報告書』](#)

[1020 ページの『示されている言語エレメントおよび概念』](#)

IGYTCARA の実行

次のプロシージャで、IGYTCARA プログラムのコンパイル、リンク・エディット、および実行が行われます。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、IGYWCLG カタログ式プロシージャを変更しなければなりません。

このタスクについて

z/OS のもとで IGYTCARA を実行するには、JCL を使用して VSAM クラスターを定義し、プログラムをコンパイルします。小文字で示されているフィールド (会計情報、ボリューム通し番号、装置名、クラスター接頭部) には、システムおよびインストール先に固有の情報を挿入します。以下の例では、名前 IGYTCAR.MASTFILE を使用しましたが、別の名前を使用しても構いません。

手順

1. 次の JCL を使用して、必要な VSAM クラスターを作成します。

```
//CREATE JOB (acct-info),'IGYTCAR CREATE VSAM',MSGLEVEL=(1,1),
// TIME=(0,29)
//CREATE EXEC PGM=IDCAMS
//VOL1 DD VOL=SER=your-volume-serial,UNIT=your-unit,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
DELETE your-prefix.IGYTCAR.MASTFILE -
FILE(VOL1) -
PURGE
DEFINE CLUSTER -
(NAME(your-prefix.IGYTCAR.MASTFILE) -
```

```
VOLUME(your-volume-serial) -
FILE(VOL1) -
INDEXED -
RECSZ(80 80) -
KEYS(16 0) -
CYLINDERS(2))
/*
```

既存のクラスターを除去するために、VSAM クラスターの作成前に、DELETE が出されます。

2. 次の JCL を使用して、IGYTCARA プログラムをコンパイル、リンク・エディット、および実行します。

```
//IGYTCARA JOB (acct-info), 'IGYTCAR', MSGLEVEL=(1,1), TIME=(0,29)
//TEST EXEC IGYWCLG
//COBOL.SYSLIB DD DSN=IGY.V6R5M0.SIGYSAMP, DISP=SHR
//COBOL.SYSIN DD DSN=IGY.V6R5M0.SIGYSAMP(IGYTCARA), DISP=SHR
//GO.SYSOUT DD SYSOUT=A
//GO.COMMUTR DD DSN=your-prefix.IGYTCAR.MASTFILE, DISP=SHR
//GO.LOCCODE DD DSN=IGY.V6R5M0.SIGYSAMP(IGYTCODE), DISP=SHR
//GO.UPDTRANS DD DSN=IGY.V6R5M0.SIGYSAMP(IGYTRANX), DISP=SHR
//GO.UPDPRINT DD SYSOUT=A, DCB=BLKSIZE=133
//
```

タスクの結果

関連タスク

201 ページの『[第 10 章 VSAM ファイルの処理](#)』

関連参照

283 ページの『[コンパイル、リンク・エディット、実行用のプロシーチャー \(IGYWCLG\)](#)』

IGYTCARB: 対話式プログラム

IGYTCARB には、IBM 対話式システム生産性向上機能 (ISPF) を使用してダイアログ・マネージャーと Enterprise COBOL を呼び出して相乗り通勤データを入力するための対話式プログラムが入っています。IGYTCARB は、IGYTCARA のような突き合わせプログラムまたは駐車場リストへの入力として使用できるファイルを作成します。

IGYTCARB の入力データは、IGYTCARA の場合と同じです。IGYTCARB により、ISPF パネルを介して入力ファイルに情報を追加することができます。IGYTCARB によって使用されるパネルの例を、以下に示します。

```
----- CARPOOL DATA ENTRY -----
New Data Entry                                Previous Entry
Type =====> -                               A, R, or D      A
Shift =====> -                               1, 2, or 3      1
Home Code ==> --                               2 Chars         01
Work Code ==> --                               2 Chars         11
Name =====> -----                          9 Chars         POPOWICH
Initials ==> --                               2 Chars         AD
Address =====> -----                       18 Chars        134 SIXTH AVENUE
City =====> -----                          13 Chars        SAN FRANCISCO
State =====> --                               2 Chars         CA
Zip Code ==> ----                              5 Chars         99903
Home Phone => -----                           10 Chars        4155553390
Work Phone => -----                           10 Chars        4155557855
Home Jnc code > --                             2 Chars         H3
Work Jnc Code > --                             2 Chars         W7
Commuter Stat > -                             D, R or blank
```

関連タスク

1011 ページの『[IGYTCARB の実行準備](#)』

IGYTCARB の実行準備

対話式システム生産性向上機能 (ISPF) のもとで、IGYTCARB プログラムを実行します。IGYTCARB (IGYTCARB、IGYTRANB、および IGYTPNL) に必要なすべてのファイルは、IGY.V6R5M0.SIGYSAMP データ・セットのプロダクト・インストール・テープで提供されます。

このタスクについて

データ・セット名とプロシージャ名は、インストール時に変更することができます。名前を確認するには、システム・プログラマーにお問い合わせください。

IGYTCARB のソース・ファイルの中の、CBL ステートメントの以下のオプションは変更しないでください。

- NONUMBER
- QUOTE
- SEQUENCE

これらのオプションが有効な場合、プログラムは診断メッセージを出しません。ソース・ファイル内のシーケンス番号ストリングを使用すれば、言語エレメントを見つけることができます。

関連概念

[1010 ページの『IGYTCARB: 対話式プログラム』](#)

関連タスク

[1011 ページの『IGYTCARB の実行』](#)

関連参照

[1020 ページの『示されている言語エレメントおよび概念』](#)

IGYTCARB の実行

次のプロシージャで、IGYTCARB プログラムのコンパイル、リンク・エディット、および実行が行われます。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、このプロシージャを変更しなければなりません。

このタスクについて

z/OS で IGYTCARB を実行するには、次の手順を行います。

手順

1. ISPF エディターを使用して、IGYTCARB 呼び出しを含めるよう、ISPF/PDF 基本オプション・パネル (ISR@PRIM) または他のパネルを変更します。パネル ISR@PRIM は、設置場所の PDF パネル・データ・セット (通常は ISRPLIB) に入っています。

次の例は、2 つの指示された位置で IGYTCARB 呼び出しを含めるよう変更された ISR@PRIM パネルを示しています。パネル定義の上部にオプションを追加 (または変更) する場合は、パネルの下部にも対応する行を追加 (または変更) しなければなりません。

```
%----- ISPF/PDF PRIMARY OPTION PANEL -----  
%OPTION ==>_ZCMD  
%  
%      0 +ISPF PARMS - Specify terminal and user parameters +USERID - &ZUSER  
%      1 +BROWSE     - Display source data or output listings +TIME   - &ZTIME  
%      2 +EDIT       - Create or change source data          +TERMINAL - &ZTERM  
%      3 +UTILITIES  - Perform utility functions             +PF KEYS  - &ZKEYS  
%      4 +FOREGROUND - Invoke language processors in foreground  
%      5 +BATCH      - Submit to batch for language processing  
%      6 +COMMAND    - Enter TSO or Workstation commands  
%      7 +DIALOG TEST - Perform dialog testing  
%      8 +LM UTILITIES- Perform library management utility functions  
%      C +IGYTCARB   - Run IGYTCARB UPDATE TRANSACTION PROGRAM (1)  
%      T +TUTORIAL   - Display information about ISPF/PDF  
%      X +EXIT       - Terminate using console, log, and list defaults  
%
```

```
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP = ISR00003
  &ZPRIM = YES           /* ALWAYS A PRIMARY OPTION MENU */
  &ZHTOP = ISR00003      /* TUTORIAL TABLE OF CONTENTS */
  &ZHINDEX = ISR91000    /* TUTORIAL INDEX - 1ST PAGE */
  VPUT (ZHTOP,ZHINDEX) PROFILE
)PROC
  &Z1 = TRUNC(&ZCMD,1)
  IF (&Z1 &notsym.= '.' )
    &ZSEL = TRANS( TRUNC (&ZCMD, '.' )
      0, 'PANEL (ISPOPTA)'
      1, 'PGM (ISRBRO) PARM (ISRBRO01)'
      2, 'PGM (ISREDIT) PARM (P, ISREDM01)'
      3, 'PANEL (ISRUTIL)'
      4, 'PANEL (ISRFPA)'
      5, 'PGM (ISRJB1) PARM (ISRJPA) NOCHECK'
      6, 'PGM (ISRPPC)'
      7, 'PGM (ISRYXDR) NOCHECK'
      8, 'PANEL (ISRLPRIM)'
      C, 'PGM (IGYTCARB)'
      T, 'PGM (ISPTUTOR) PARM (ISR00000)'
      , ' '
      , ' '
      X, 'EXIT'
      *, '? ' )
    &ZTRAIL = .TRAIL
  IF (&Z1 = '.' ) .msg = ISPD141
)END
```

この例の **(1)** で示されているように、以下を入力して、パネルの上部に IGYTCARB を追加します。

```
%      C +IGYTCARB      - Run IGYTCARB UPDATE TRANSACTION PROGRAM
```

(2)で示されているように、以下を入力して、パネルの下部に対応する行を追加します。

C, 'PGM(IGYTCARB)'

2. `ISR@PRIM` (または他の変更したパネル) と `IGYTPNL` をライブラリーに入れ、このライブラリーを `ISPLLIB` 連結の最初のライブラリーにします。
3. C コメントシーケンス行 `IB2200` シーケンス行のコメントを解除します `IB2210 IGYTCARB` で。(z/OS のもとでは、`OPEN EXTEND` ステートメントがサポートされます。)
4. `IGYTCARB` をコンパイルおよびリンク・エディットし、結果として生じたプログラム・オブジェクトを `LOADLIB` に入れます。
5. 次のコマンドを使用して、`ISPLLIB` を割り振ります。

```
ALLOCATE FILE(ISPLLIB) DATASET(DSN1, SYS1.COBLIB, DSN2) SHR REUSE
```

ここで、*DSN1* はステップ 4 からの *LOADLIB* のライブラリー名です。 *DSN2* はインストール済みの *ISPLLIB* です。

6. 次のコマンドを使用して、入力データ・セットと出力データ・セットを割り振ります。

```
ALLOCATE FILE(UPDTRANS) DA('IGY.V6R5M0.SIGYSAMP(IGYTRANB)') SHR REUSE
```

7. 次のコマンドを使用して、ISPPLIB を割り振ります。

```
ALLOCATE FILE(ISPPLIB) DATASET(DSN3, DSN4) SHR REUSE
```

ここで、*DSN3* は、変更されたパネルが含まれているライブラリーです。*DSN4* は、ISPF パネル・ライブラリーです。

8. 変更されたパネルを使用して、IGYTCARB を呼び出します。

タスクの結果

関連参照

対話式システム生産性向上機能 (ISPF) ダイアログ開発者 ガイドとリファレンス

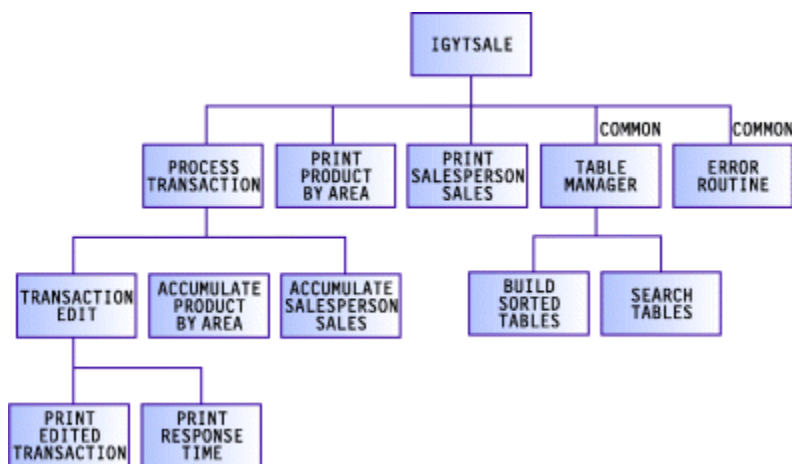
IGYTSALE: ネストされたプログラム・アプリケーション

アプリケーション IGYTSALE は、スポーツ用品販売店の商品売上と売上手数料を追跡します。

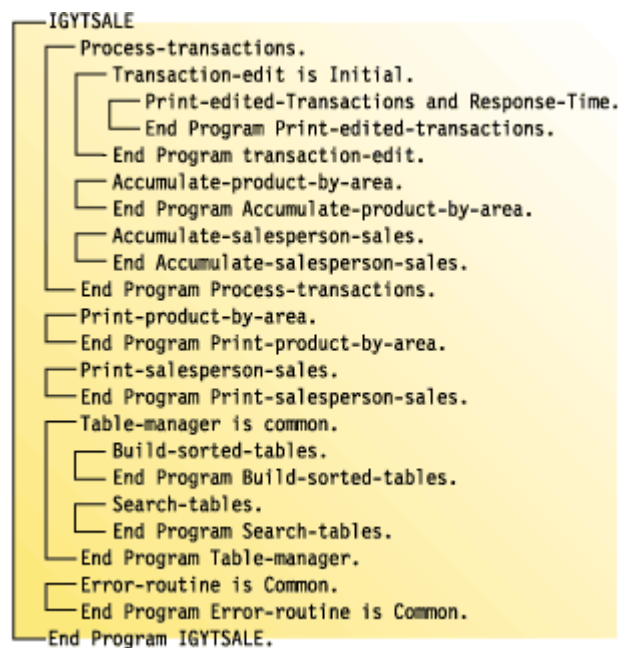
このネストされたプログラム・アプリケーションでは、次のタスクが実行されます。

1. 製品種目、顧客、および販売員の人数を記録します。このデータは、IGYTABLE と呼ばれるファイルに保管されます。
2. 有効なトランザクションとトランザクション・エラーを記録するファイルを保守します。無効なトランザクションにはすべてフラグが立てられ、結果は報告書に印刷されます。処理されるトランザクションは、IGYTRANA と呼ばれるファイルの中にあります。
3. トランザクションを処理して、地域ごとの売上高を報告します。
4. 各販売員の販売実績と手数料を記録して、その結果を報告書に印刷します。
5. 販売日と出荷日を地方時と UTC (世界協定時) で報告し、応答時間を計算します。

次の図は、アプリケーションの各部分を階層として示しています。



次の図は、各部分がどのようにネストされているかを示します。



関連タスク

[1019 ページの『IGYTSALE の実行準備』](#)

関連参照

[1014 ページの『IGYTSALE の入力データ』](#)

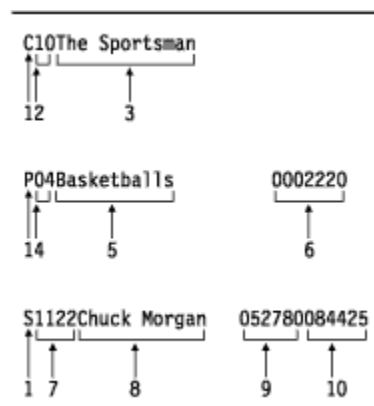
[1015 ページの『IGYTSALE によって作成される報告書』](#)

[1020 ページの『示されている言語エレメントおよび概念』](#)

IGYTSALE の入力データ

プログラムへの入力として、この流通業者は、その顧客、販売員、および製品についての情報を集め、その情報をコーディングし、入力ファイルを作成しました。

IGYTABLE と呼ばれるこの入力ファイルは、トランザクション処理時に使用できるように 3 つの異なるテーブルにロードされます。以下に、このファイルの形式を示し、その後に各項目を説明しています。



1. レコード・タイプ
2. 顧客コード
3. 顧客名
4. 製品コード
5. 製品説明
6. 製品の単価
7. 販売員の番号
8. 販売員名
9. 雇用日付
10. 手数料のレート

フィールド 1 の値 (C、P、または S) によって、入力レコードの形式が決まります。IGYTABLE のセクションの例を、以下に示します。

```
S1111Edyth Phillips 062484042327
S1122Chuck Morgan 052780084425
S1133Art Tung 022882061728
S1144Billy Jim Bob 010272121150
S1155Chris Preston 122083053377
S1166Al Willie Roz 111276100000
P01Footballs 0000620
P02Football Equipment 0032080
P03Football Uniform 0004910
P04Basketballs 0002220
P05Basketball Rim/Board0008830
P06Basketball Uniform 0004220
C01L. A. Sports
C02Gear Up
C03Play Outdoors
C04Sports 4 You
C05Sports R US
```

C06Stay Active
C07Sport Shop
C08Stay Sporty
C09Hot Sports
C10The Sportsman
C11Playing Ball
C12Sports Play
. . .

さらに、この流通業者では販売トランザクションに関する情報を集めました。各トランザクションは、特定の顧客に対する個々の販売員の売上高を示しています。顧客は、各トランザクション時に1から5個の品目を購入することができます。トランザクション情報がコーディングされ、IGYTRANA と呼ばれる入力ファイルに入られます。以下に、このファイルの形式を示し、その後に各項目を説明しています。

```
B11123919901110123314SAN DIEGO 11660919901114235505260200270500110522250100140010
  ↑   ↑      ↑      ↑      ↑      ↑   ↑   ↑      ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑   ↑
  1   2      3      4      5   6   7      8   9   8   9   8   9   8   9   8   9
```

1. 販売注文番号
2. 送り状を送った品目 (発注された各品目の数)
3. 販売の日付 (年月日時分秒)
4. 販売区域
5. 販売員の番号
6. 顧客コード
7. 出荷の日付 (年月日時分秒)
8. 製品コード
9. 売上数量

フィールド 8 および 9 は、発注された各品目の数 (フィールド 2) に応じて 1 から 8 回発生します。IGYTRANA のセクションの例を、以下に示します。

```
A00001119900227010101CNTRL VALLEY11442019900228259999
A00004119900310100530CNTRL VALLEY11441019900403150099
A00005119900418222409CNTRL VALLEY11441219900419059900
A00006119900523151010CNTRL VALLEY11442019900623250004
  419990324591515SAN DIEGO 11615 60200132200110522045100
B11114419901111003301SAN DIEGO 11661519901114260200132200110522041100
A00007119901115003205CNTRL VALLEY11332019901117120023
C00125419900118101527SF BAY AREA 11331519900120160200112200250522145111
B11116419901201132013SF BAY AREA 11331519901203060200102200110522045102
B11117319901201070833SAN Diego 11656619901203330200132200120522041100
B11118419901221191544SAN DIEGO 11661419901223160200142200130522040300
B11119419901210211544SAN DIEGO 11221219901214060200152200160522050500
B11120419901212000816SAN DIEGO 11220419901213150200052200160522040100
B11121419901201131544SAN DIEGO 11330219901203120200112200140522250100
B11122419901112073312SAN DIEGO 11221019901113100200162200260522250100
B11123919901110123314SAN DIEGO 11660919901114260200270500110522250100140010
B11124219901313510000SAN DIEGO 116611 1 0200042200120a22141100
B11125419901215012510SAN DIEGO 11661519901216110200162200130522141111
B11126119901111000034SAN DIEGO 11331619901113260022
B11127119901110154100SAN DIEGO 11221219901113122000
B11128419901110175001SAN DIEGO 11661519901113260200132200160521041104
. . .
```

IGYTSALE によって作成される報告書

下記の図は、IGYTSALE 出力のサンプルです。

プログラムは以下のデータを報告書に記録します。

- トランザクション・エラー
- 製品および区域別の販売実績
- 個々の販売員の販売実績と歩合

- 販売日から、販売された製品が配送された日付までの応答時間
- 実際の出力は、システムによって形式が多少変わることがあります。

[1016 ページの『例: IGYTSALE のトランザクション・エラー』](#)
[1016 ページの『例: IGYTSALE の製品別および区域別の販売分析』](#)
[1018 ページの『例: IGYTSALE の販売と歩合』](#)
[1019 ページの『例: IGYTSALE の販売から配送までの応答時間』](#)

例: IGYTSALE のトランザクション・エラー

IGYTSALE 出力の次の例では、最後の欄にトランザクション・エラーがあることを示しています。

Day of Report: Tuesday		C O B O L		S P O R T S		11/24/2003		03:12	Page: 1
Invalid Edited Transactions		Sales Area		Sales Pers		Cust. Product And Quantity Sold		Ship Date Stamp	
Sales Order	Inv. Items	Sales Time Stamp	Sales Area	Sales Pers	Cust. Product	And Quantity	Sold	Ship Date Stamp	
-----	-----	-----	-----	-----	-----	-----	-----	-----	
	4	19990324591515	SAN DIEGO	116	15	60200132200110522045100			
invalid								Error Descriptions	
								-Sales order number is missing	
								-Date of sale time stamp is	
table								-Salesperson number not numeric	
								-Product code not in product-	
invalid B11117		3	19901201070833	SAN Diego	1165	66	330200132200120522041100	19901203	Error Descriptions
								-Sales area not in area-table	
								-Salesperson not in sales-per-	
table								-Customer code not in customer-	
table								-Product code not in product-	
table								-Quantity sold not numeric	
B11123		9	19901110123314	SAN DIEGO	1166	09	260200270500110522250100140010	19901114	Error Descriptions
checked								-Invoiced items is invalid	
								-Product and quantity not	
invalid B11124		2	19901313510000	SAN DIEGO	1166	11	1 0200042200120a22141100		-Date of ship time stamp is
invalid								Error Descriptions	
								-Date of sale time stamp is	
invalid 133		8	1119110000	LOS ANGELES	1166	10	040112110210160321251104		-Product code is invalid
								-Date of ship time stamp is	
invalid								Error Descriptions	
								-Sales order number is invalid	
								-Invoiced items is invalid	
								-Date of sale time stamp is	
checked								-Product and quantity not	
invalid C11133		4	1990111944		1166	10	040112110210160321251104		-Date of ship time stamp is
invalid								Error Descriptions	
								-Date of sale time stamp is	
								-Sales area is missing	
								-Date of ship time stamp is	
invalid C11138		4	19901117091530	LOS ANGELES	1155	11	3200102010260321250004	19901119	Error Descriptions
								-Customer code is invalid	
D00009		9	19901201222222	CNTRL COAST	115	19	141 1131221	19901202	Error Descriptions
								-Invoiced items is invalid	

例: IGYTSALE の製品別および区域別の販売分析

IGYTSALE 出力の次の例では、製品および区域別による販売成績を示しています。

Day of Report: Tuesday		C O B O L		S P O R T S		11/24/2003		03:12	Page: 1
Sales Analysis By Product By Area		Areas of Sale							
			CNTRL COAST		CNTRL VALLEY		LOS ANGELES		NORTH COAST
									SAN DIEGO
	Product								SF BAY AREA
	Codes								
Totals									Product
=====									
==									
	Product Number	04							
	Basketballs								
	Units Sold				433			2604	5102
8139	Unit Price				22.20			22.20	22.20
	Amount of Sale				\$9,612.60			\$57,808.80	\$113,264.40

\$180,685.80 |

```
--
|Product Number 05 |
|Basketball Rim/Board|
|Units Sold |
14731 |          9900 |          2120 |          11 |          2700 |
|Unit Price |
|          88.30 |          88.30 |          88.30 |          88.30 |
|Amount of Sale |
$1,300,747.30 |
```

```
--
|Product Number 06 |
|Basketball Uniform |
|Units Sold |
1390 |          990 |          200 |          200 |
|Unit Price |
|          42.20 |          42.20 |          42.20 |
|Amount of Sale |
$58,658.00 |
```

```
--
|Product Number 10 |
|Baseball Cage |
|Units Sold |
7031 |          45 |          3450 |          16 |          200 |          3320 |
|Unit Price |
|          890.00 |          890.00 |          890.00 |          890.00 |          890.00 |
|Amount of Sale |
$6,257,590.00 |
```

```
--
|Product Number 11 |
|Baseball Uniform |
|Units Sold |
19249 |        10003 |          3578 |          2922 |          2746 |
|Unit Price |
|          45.70 |          45.70 |          45.70 |          45.70 |          45.70 |
|Amount of Sale |
$879,679.30 |
```

```
--
|Product Number 12 |
|Softballs |
|Units Sold |
4946 |          10 |          137 |          2564 |          13 |          2200 |          22 |
|Unit Price |
|          1.40 |          1.40 |          1.40 |          1.40 |          1.40 |          1.40 |
|Amount of Sale |
$6,924.40 |
```

```
--
|Product Number 13 |
|Softball Bats |
|Units Sold |
14068 |        3227 |          3300 |          1998 |          5444 |          99 |
|Unit Price |
|          12.60 |          12.60 |          12.60 |          12.60 |          12.60 |          12.60 |
|Amount of Sale |
$177,256.80 |
```

```
--
|Product Number 14 |
|Softball Gloves |
|Units Sold |
13395 |        1155 |          136 |          3119 |          3833 |          5152 |
|Unit Price |
|          12.00 |          12.00 |          12.00 |          12.00 |          12.00 |          12.00 |
|Amount of Sale |
$160,740.00 |
```

```
--
|Product Number 15 |
|Softball Cage |
|Units Sold |
5496 |          997 |          99 |          2000 |          2400 |
|Unit Price |
|          890.00 |          890.00 |          890.00 |          890.00 |
|Amount of Sale |
$4,891,440.00 |
```

```
--
|Product Number 16 |
```

Softball Uniform						
Units Sold	44		465	16	6165	200
Unit Price	45.70		45.70	45.70	45.70	45.70
Amount of Sale	\$2,010.80		\$21,250.50	\$731.20	\$281,740.50	\$9,140.00
\$314,873.00						
Product Number 25						
RacketBalls						
Units Sold	1001	10003	1108	8989	200	522
Unit Price	0.60	0.60	0.60	0.60	0.60	0.60
Amount of Sale	\$600.60	\$6,001.80	\$664.80	\$5,393.40	\$120.00	\$313.20
\$13,093.80						
Product Number 26						
Racketball Rackets						
Units Sold	21		862	194	944	31
Unit Price	12.70		12.70	12.70	12.70	12.70
Amount of Sale	\$266.70		\$10,947.40	\$2,463.80	\$11,988.80	\$393.70
\$26,060.40						
Total Units Sold	16503	20139	20016	15346	29812	17394 *
Total Sales	\$1,441,929.40	\$968,473.60	\$5,290,487.50	\$128,198.70	\$3,163,713.90	\$3,274,945.70 *
\$14,267,748.80 *						

例: IGYTSALE の販売と歩合

IGYTSALE 出力の次の例では、販売員別の販売実績と歩合を示しています。

Day of Report: Tuesday	C O B O L	S P O R T S	11/24/2003	03:12	Page: 1	
Sales and Commission Report						
Salesperson: Billy Jim Bob						
Customers:	Number of Orders	Products Ordered	Total for Order	Discount (if any)	Discount Amount	Commission Earned
-----	-----	-----	-----	-----	-----	-----
Sports Stop	3	10117	\$6,161.40	2.25%	\$138.63	\$746.45
The Sportsman	1	99	\$88,110.00	5.06%	\$4,458.36	\$10,674.52
Sports Play	1	9900	\$874,170.00	7.59%	\$66,349.50	\$105,905.69
-----	-----	-----	-----	-----	-----	-----
Totals:	5	20116	\$968,441.40		\$70,946.49	\$117,326.66
Salesperson: Willie Al Roz						
Customers:	Number of Orders	Products Ordered	Total for Order	Discount (if any)	Discount Amount	Commission Earned
-----	-----	-----	-----	-----	-----	-----
Winners Club	4	13998	\$1,572,775.90	7.59%	\$119,373.69	\$157,277.59
Winning Sports	1	3222	\$48,777.20	3.38%	\$1,648.66	\$4,877.72
The Sportsman	1	1747	\$27,415.50	3.38%	\$926.64	\$2,741.55
Play Outdoors	1	2510	\$18,579.60	3.38%	\$627.99	\$1,857.96
-----	-----	-----	-----	-----	-----	-----
Totals:	7	21477	\$1,667,548.20		\$122,576.98	\$166,754.82
Salesperson: Art Tung						
Customers:	Number of Orders	Products Ordered	Total for Order	Discount (if any)	Discount Amount	Commission Earned
-----	-----	-----	-----	-----	-----	-----
Sports Stop	1	23	\$32.20	2.25%	\$72	\$1.98
Winners Club	2	16057	\$2,274,885.00	7.59%	\$172,663.77	\$140,424.10
Gear Up	1	3022	\$107,144.00	7.59%	\$8,132.22	\$6,613.78
Sports Club	1	22	\$279.40	2.25%	\$6.28	\$17.24
Sports Fans Shop	1	1044	\$20,447.30	3.38%	\$691.11	\$1,262.17
L. A. Sports	1	1163	\$979,198.10	7.59%	\$74,321.13	\$60,443.94
-----	-----	-----	-----	-----	-----	-----
Totals:	7	21331	\$3,381,986.00		\$255,815.23	\$208,763.21
Salesperson: Chuck Morgan						
Customers:	Number of Orders	Products Ordered	Total for Order	Discount (if any)	Discount Amount	Commission Earned
-----	-----	-----	-----	-----	-----	-----
Sports Play	3	7422	\$3,817,245.40	7.59%	\$289,728.92	\$322,270.94
Sports 4 You	1	3022	\$398,335.40	7.59%	\$30,233.65	\$33,629.46
The Sportsman	1	3022	\$285,229.40	7.59%	\$21,648.91	\$24,080.49
Sports 4 Winners	1	1100	\$68,509.40	5.06%	\$3,466.57	\$5,783.90
Sports Club	1	12027	\$1,324,256.10	7.59%	\$100,511.03	\$111,800.32
-----	-----	-----	-----	-----	-----	-----
Totals:	7	26593	\$5,893,575.70		\$445,589.08	\$497,565.11
Salesperson: Chris Preston						
Customers:	Number of	Products	Total for	Discount	Discount	Commission

	Orders	Ordered	Order	(if any)	Amount	Earned
Playing Ball	1	5535	\$1,939,219.10	7.59%	\$147,186.72	\$103,509.69
Play Sports	1	5675	\$225,130.80	7.59%	\$17,087.42	\$12,016.80
Winners Club	1	631	\$14,069.70	2.25%	\$316.56	\$750.99
The Jock Shop	1	2332	\$28,716.60	3.38%	\$970.62	\$1,532.80
Totals:	4	14173	\$2,207,136.20		\$165,561.32	\$117,810.28
Salesperson: Edyth Phillips						
Customers:	Number of Orders	Products Ordered	Total for Order	Discount (if any)	Discount Amount	Commission Earned
Sports Play	2	3575	\$92,409.90	5.06%	\$4,675.94	\$3,911.43
Winning Sports	1	11945	\$56,651.40	5.06%	\$2,866.56	\$2,397.88
Totals:	3	15520	\$149,061.30		\$7,542.50	\$6,309.31
Grand Totals:	33	119210	\$14,267,748.80		\$1,068,031.60	\$1,114,529.39

例: IGYTSALE の販売から配送までの応答時間

IGYTSALE 出力の次の例では、米国での販売日から、販売された製品がヨーロッパに向けて配送された日付までの応答時間を示しています。

Day of Report: Monday COBOL SPORTS 11/24/2003 03:12 Page: 1						
Response Time from USA Sale to European Ship						
Prod Code	Units Sold	Sale Date/Time(PST) YYYYMMDD HHMMSS	Ship Date YYYYMMDD	Ship Day	Response Time Days	
25	9999	19900226 010101	19900228	WED	.95	
15	99	19900310 100530	19900403	TUE	23.57	
05	9900	19900418 222409	19900419	THU	.06	
25	4	19900523 151010	19900623	SAT	30.36	
04	1100	19901110 003301	19901114	WED	2.97	
12	23	19901114 003205	19901117	SAT	1.97	
14	5111	19900118 101527	19900120	SAT	1.57	
04	5102	19901201 132013	19901203	MON	1.44	
04	300	19901221 191544	19901223	SUN	1.19	
05	500	19901210 211544	19901214	FRI	3.11	
04	100	19901211 000816	19901213	THU	.99	
25	100	19901201 131544	19901203	MON	1.44	
25	100	19901112 073312	19901113	TUE	.68	
14	1111	19901214 012510	19901216	SUN	.94	
26	22	19901110 000034	19901113	TUE	1.99	
12	2000	19901110 154100	19901113	TUE	2.34	
04	1104	19901110 175001	19901113	TUE	2.25	
12	114	19901229 115522	19901230	SUN	.50	
15	2000	19901110 190113	19901114	WED	3.20	
10	1440	19901112 001500	19901115	THU	1.98	
25	1104	19901118 120101	19901119	MON	.49	
25	4	19901118 110030	19901119	MON	.54	
12	144	19901114 010510	19901119	MON	3.95	
14	112	19901119 010101	19901122	THU	1.95	
26	321	19901117 173945	19901119	MON	1.26	
13	1221	19901101 135133	19901102	FRI	.42	
10	22	19901029 210000	19901030	TUE	.12	
14	35	19901130 160500	19901201	SAT	.32	
11	9005	19901211 050505	19901212	WED	.78	
06	990	19900511 214409	19900515	TUE	3.09	
13	1998	19900712 150100	19900716	MON	3.37	
26	31	19901010 185559	19901011	THU	.21	
14	30	19901210 195500	19901212	WED	1.17	

IGYTSALE の実行準備

IGYTSALE プログラム (IGYTSALE、IGYTCRC、IGYTPRC、IGYTSRC、IGYTABLE、および IGYTRANA) によって必要とされるすべてのファイルは、IGY.V6R5M0.SIGYSAMP データ・セットのプロダクト・インストール・テープにあります。

このタスクについて

データ・セット名およびプロシージャ名は、インストール時に変更できます。これらの名前の妥当性について、システム・プログラマーにお問い合わせください。

IGYTSALE のソース・ファイルの中の、CBL ステートメントの以下のオプションは変更しないでください。

- NONUMBER
- SEQUENCE
- NONUMBER
- QUOTE

これらのオプションを適用すると、プログラムは診断メッセージを発行しなくなる可能性があります。ソース・ファイル内のシーケンス番号ストリングを使用すれば、使用されている言語エレメントを見つけることができます。

IGYTSALE を実行すると、次のメッセージが SYSOUT データ・セットに印刷されます。

```
Program IGYTSALE Begins
There were 00041 records processed in this program
Program IGYTSALE Normal End
```

関連概念

[1013 ページの『IGYTSALE: ネストされたプログラム・アプリケーション』](#)

関連タスク

[1020 ページの『IGYTSALE の実行』](#)

関連参照

[1014 ページの『IGYTSALE の入力データ』](#)

[1015 ページの『IGYTSALE によって作成される報告書』](#)

[1020 ページの『示されている言語エレメントおよび概念』](#)

IGYTSALE の実行

次の JCL を使用して、IGYTSALE プログラムをコンパイル、リンク・エディット、および実行します。プログラムのコンパイルだけ、またはコンパイルとリンク・エディットだけを行う場合は、IGYWCLG カタログ式プロシージャを変更しなければなりません。

このタスクについて

小文字で示されているフィールドには、システムまたはインストール先の会計情報を挿入します。

```
//IGYTSALE JOB (acct-info), 'IGYTSALE', MSGLEVEL=(1,1), TIME=(0,29)
//TEST EXEC IGYWCLG
//COBOL.SYSLIB DD DSN=IGY.V6R5M0.SIGYSAMP, DISP=SHR
//COBOL.SYSIN DD DSN=IGY.V6R5M0.SIGYSAMP(IGYTSALE), DISP=SHR
//GO.SYSOUT DD SYSOUT=A
//GO.IGYTABLE DD DSN=IGY.V6R5M0.SIGYSAMP(IGYTABLE), DISP=SHR
//GO.IGYTRANS DD DSN=IGY.V6R5M0.SIGYSAMP(IGYTRANA), DISP=SHR
//GO.IGYPRINT DD SYSOUT=A, DCB=BLKSIZE=133
//GO.IGYPRT2 DD SYSOUT=A, DCB=BLKSIZE=133
//
```

示されている言語エレメントおよび概念

サンプル・プログラムでは、いくつかの COBOL 言語エレメントおよび概念を示しています。

サンプル・プログラムに適正な言語エレメントを見つけるためには、シーケンス・ストリングの欄でプログラムの省略形を探します。

サンプル・プログラム	省略形
IGYTCARA	IA
IGYTCARB	IB
IGYTSALE	IS

次の表に、サンプル・プログラムで例示されている言語エレメントとプログラミングの概念を示します。言語エレメントまたは概念を説明し、シーケンス・ストリングを示しています。シーケンス・ストリングとは、ソース・ファイルのシーケンス・フィールドに現れる特殊な文字ストリングのことを言います。このストリングを検索索引数として使用して、リスト内のエレメントを見つけることができます。

言語エレメントまたは概念	シーケンス・ストリング
ACCEPT . . . FROM DAY-OF-WEEK	IS0900
ACCEPT . . . FROM DATE	IS0901
ACCEPT . . . FROM TIME	IS0902
ADD . . . TO	IS4550
AFTER ADVANCING	IS2700
AFTER PAGE	IS2600
ALL	IS4200
ASSIGN	IS1101
AUTHOR	IA0040
CALL	IS0800
呼び出し可能サービス (言語環境): 1. CEEDATM: 日付および時刻出力の形式設定 2. CEEDCOD: フィードバック・コード検査 3. CEEGMTO: 地方時からの UTC オフセット 4. CEELOCT: ローカル日付および時刻 5. CEESECS: タイム・スタンプから秒への変換	1. IS0875, IS2575 2. IS0905 3. IS0904 4. IS0850 5. IS2350, IS2550
CLOSE ファイル	IS1900
交換可能なコンマ、セミコロン、およびスペース	IS3500, IS3600
ネストされたプログラムの COMMON ステートメント	IS4600
複合 OCCURS DEPENDING ON	IS0700, IS3700
COMPUTE	IS4501
COMPUTE ROUNDED	IS4500
CONFIGURATION SECTION	IA0970
CONFIGURATION SECTION (オプション)	IS0200
CONTINUE ステートメント	IA5310, IA5380
COPY ステートメント	IS0500
DATA DIVISION (オプション)	IS5100
データ妥当性検査	IA5130-6190
実行まで (PERFORM . . . TEST AFTER)	IA4900-5010, IA7690-7770
一方を行います (PERFORM . . . TEST BEFORE)	IS1660
END-ADD	IS2900
END-COMPUTE	IS4510

言語エレメントまたは概念	シーケンス・ストリング
END-EVALUATE	IA6590, IS2450
END-IF	IS1680
END-MULTIPLY	IS3100
END-PERFORM	IS1700
END PROGRAM	IA9990
END-READ	IS1800
END-SEARCH	IS3400
ENVIRONMENT DIVISION (オプション)	IS0200
エラー処理、プログラムの終了	IA4620, IA5080, IA7800-7980
EVALUATE ステートメント	IA6270-6590
EVALUATE . . . ALSO	IS2400
EXIT PROGRAM は段落内で唯一のステートメントである必要はない	IS2000
指数	IS4500
EXTERNAL 節	IS1200
順次ファイルの FILE-CONTROL 記入項目	IA1190-1300
VSAM 索引付きファイルの FILE-CONTROL 記入項目	IA1070-1180
FILE SECTION (オプション)	IS0200
FILE STATUS コード検査	IA4600-4630, IA4760-4790
FILLER (オプション)	IS0400
フラグ、レベル 88、定義	IA1730-1800, IA2440-2480, IA2710
フラグ、レベル 88、テスト	IA4430, IA5200-5250
FLOATING POINT	IS4400
GLOBAL ステートメント	IS0300
ネストされたプログラムの INITIAL ステートメント	IS2300
INITIALIZE	IS2500
Initializing a table in the DATA DIVISION でのテーブルの初期化	IA2920-4260
インライン PERFORM ステートメント	IA4410-4520
I-O-CONTROL 段落 (オプション)	IS0200
INPUT-OUTPUT SECTION (オプション)	IS0200

言語エレメントまたは概念	シーケンス・ストリング
組み込み関数	
1. CURRENT-DATE	1. IA9005
2. MAX	2. IA9235
3. MEAN	3. IA9215
4. MEDIAN	4. IA9220
5. MIN	5. IA9240
6. STANDARD-DEVIATION	6. IA9230
7. UPPER-CASE	7. IA9015
8. VARIANCE	8. IA9225
9. WHEN-COMPILED	9. IA9000
IS (すべての節でオプション)	IS0700
LABEL RECORDS (オプション)	IS1150
LINKAGE SECTION	IS4900
指標と添え字の混合	IS3500
簡略名	IA1000
MOVE	IS0903
MOVE CORRESPONDING ステートメント	IA4810, IA4830
MULTIPLY . . . GIVING	IS3000
END-IF を使用した、ネストされた IF ステートメント	IA5460-5830
ネストされたプログラム	IS1000
NEXT SENTENCE	IS4300
NOT AT END	IS1600
NULL	IS4800
OBJECT-COMPUTER (オプション)	IS0200
OCCURS DEPENDING ON	IS0710
ODO は項目を受け取るのに最大長を使用する	IS1550
OPEN EXTEND	IB2210
OPEN INPUT	IS1400
OPEN OUTPUT	IS1500
ORGANIZATION (オプション)	IS1100
ページ替え	IA7180-7210
簡略化された条件の中の括弧	IS4850
PERFORM . . . WITH TEST AFTER (実行まで)	IA4900-5010, IA7690-7770
PERFORM . . . WITH TEST BEFORE (一方を行います)	IS1660
PERFORM . . . UNTIL	IS5000

言語エレメントまたは概念	シーケンス・ストリング
PERFORM . . . VARYING ステートメント	IA7690-7770
POINTER 関数	IS4700
印刷ファイル FD 記入項目	IA1570-1620
印刷報告書	IA7100-7360
PROCEDURE DIVISION . . . USING	IB1320-IB1650
PROGRAM-ID (30 文字まで許可される)	IS0120
READ . . . INTO . . . AT END	IS1550
REDEFINES ステートメント	IA1940, IA2060, IA2890, IA3320
参照変更	IS2425
関係演算子 <= (より小さい、または等しい)	IS4400
関係演算子 >= (より大きいまたは等しい)	IS2425
相対添え字付け	IS4000
REPLACE	IS4100
SEARCH ステートメント	IS3300
SELECT	IS1100
順序番号には任意の文字を入れることができる	IA, IB, IS
順次ファイル処理	IA4480-4510, IA4840-4870
PERFORM を使用した順次テーブル検索	IA7690-7770
SEARCH を使用した順次テーブル検索	IA5270-5320, IA5340-5390
SET INDEX	IS3200
SET . . . TO TRUE ステートメント	IA4390, IA4500, IA4860, IA4980
SOURCE-COMPUTER (オプション)	IS0200
SPECIAL-NAMES 段落 (オプション)	IS0200
STRING ステートメント	IA6950, IA7050
小文字のサポート	IS0100
TALLY	IS1650
ネストされたプログラムの TITLE ステートメント	IS0100
通勤者のレコードの更新	IA6200-6610
トランザクション作業値スペースの更新	IB0790-IB1000
USAGE BINARY	IS1300
USAGE PACKED-DECIMAL	IS1301
妥当性検査エレメント	IB0810, IB0860, IB1000
OCCURS が指定された VALUE	IS0600
VALUE SPACE (S)	IS0601

言語エレメントまたは概念	シーケンス・ストリング
VALUE ZERO (S) (ES)	IS0600
可変長テーブルの制御変数	IA5100
可変長テーブルの定義	IA2090-2210
可変長テーブルのロード	IA4840-4990
VSAM 索引付きファイルのキー定義	IA1170
VSAM 戻りコードの表示	IA7800-7900
WORKING-STORAGE SECTION	IS0250

JCL パラメーターへのアクセス

JCL から COBOL プログラムにパラメータ文字列を渡すには、PARM=キーワードの EXEC 声明。このパラメーターには、標準 COBOL コーディングによって、または CEE3PR2 言語環境 呼び出し可能サービス を呼び出すことによって、アクセスできます。

COBOL コーディングの使用

定義する必要があります LINKAGE SECTION 記録 (level-01) を受け取ることですユーザーパラメータ渡されたデータ PARM システムによって文字列の前に挿入されるハーフワード パラメータ リストの長さフィールドを考慮した文字列。

プログラムは、このフィールド長がゼロではないことをテストし、PARM ストリング・データが実際に渡されていることを検証できます。例:

```
LINKAGE SECTION.
01 PARMDATA.
   05 STRINGLEN PIC 9(4) USAGE COMP.
   05 STRINGPARM PIC X(80).
PROCEDURE DIVISION USING PARMDATA.
   IF STRINGLEN > 0 . . .
```

詳しくは、[627 ページの『LINKAGE SECTION のコーディング』](#)を参照してください。

CEE3PR2 の使用

CEE3PR2 呼び出し可能サービスへのパラメーターを定義する必要があります。その際、パラメーターを PROCEDURE DIVISION USING ステートメントに追加する必要はありません。

```
WORKING-STORAGE SECTION.
01 PARMLN PIC S9(9) BINARY.
01 PARMSTR.
   02 STR1-LENGTH PIC S9(4) BINARY.
   02 STR1-STRING.
       03 STR1-CHAR PIC X
           OCCURS 0 TO 256 TIMES
           DEPENDING ON STR1-LENGTH.
. . .
CALL "CEE3PR2" USING PARMLN,PARMSTR, FC.
```

詳細については、CEE3PR2 呼び出し可能なサービスについては、CEE3PR2 の中に[言語環境プログラミング リファレンス](#)。

関連タスク

[637 ページの『z/OS 下でのメインプログラム・パラメーターへのアクセス』](#)

Enterprise COBOL for z/OS のアクセシビリティ機能

アクセシビリティ機能は、運動や視覚などに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支援します。z/OS のアクセストピックスは、Enterprise COBOL for z/OS のアクセシビリティ機能を提供します。

アクセシビリティ機能

z/OS には、以下の主なアクセシビリティ機能があります。

- スクリーン・リーダーおよび画面拡大機能ソフトウェアで一般的に使用されるインターフェース
- キーボードのみによるナビゲーション
- 色、コントラスト、フォント・サイズなど表示属性のカスタマイズ機能

z/OS では、最新の W3C 標準 [WAI-ARIA 1.0](http://www.w3.org/TR/wai-aria/) (<http://www.w3.org/TR/wai-aria/>) が、US Section 508 (<https://www.access-board.gov/ict/>) および [Web Content Accessibility Guidelines \(WCAG\) 2.0](http://www.w3.org/TR/WCAG20/) (<http://www.w3.org/TR/WCAG20/>) に準拠するように使用されています。アクセシビリティ機能を利用するには、最新リリースのスクリーン・リーダーを、この製品でサポートされる最新の Web ブラウザーと併用してください。

IBM Knowledge Center 内の Enterprise COBOL for z/OS オンライン製品資料には、アクセシビリティ対応のものがありません。IBM Knowledge Center のアクセシビリティ機能については、<http://www.ibm.com/support/knowledgecenter/en/about/releasenotes.html> に説明があります。

キーボード・ナビゲーション

ユーザーは、TSO/E または ISPF を使用して z/OS ユーザー・インターフェースにアクセスできます。

ユーザーは、IBM 開発者 z/OS を使用して、z/OS サービスにアクセスすることもできます。

これらのインターフェースへのアクセスに関する情報については、以下の資料を参照してください。

- [z/OS TSO/E Primer](http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4p120) (<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4p120>)
- [z/OS TSO/E User's Guide](http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4c240/APPENDIX1.3) (<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4c240/APPENDIX1.3>)
- [z/OS ISPF User's Guide Volume I](http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ispzug70) (<http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ispzug70>)

上記の資料には、キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を含む TSO/E および ISPF の使用方法が記載されています。それぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

インターフェースに関する情報

Enterprise COBOL for z/OS のオンライン製品資料は、標準の Web ブラウザーから表示可能な IBM Knowledge Center で入手できます。

PDF ファイルでのアクセシビリティ・サポートは限定的です。PDF 資料では、オプションのフォント拡大機能およびハイコントラスト表示設定を使用でき、キーボードのみでナビゲートできます。

スクリーン・リーダーで、ピリオドやコンマなどの PICTURE 記号を含む構文図、ソース・コード例、およびテキストを正確に読み上げるには、すべての句読点を読み上げるようにスクリーン・リーダーを設定する必要があります。

支援テクノロジー製品は、z/OS 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、z/OS インターフェースへのアクセスに使用する支援テクノロジー製品の資料を参照してください。

関連アクセシビリティ情報

IBM では、標準の IBM ヘルプ・デスクおよびサポート Web サイトに加えて、耳が不自由なお客様に販売およびサポート・サービスをご利用いただけるように TTY 電話サービスを提供します。

TTY サービス
800-IBM-3383 (800-426-3383)
(北米内)

IBM のアクセシビリティへの取り組み

IBM のアクセシビリティへの取り組みについては、[IBM Accessibility \(www.ibm.com/able\)](http://www.ibm.com/able) を参照してください。

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。

IBM は、本書に記載の製品、サービス、または機能を日本においては提供し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。

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
U.S.A.

2 バイト (DBCS) 情報に関するライセンス照会については、国内の IBM 知的財産部門に連絡するか、書面でお問い合わせをお送りください。

〒 103-8510
東京都中央区日本橋箱崎町 19 番 21 号
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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。

知的財産部門合理的なソフトウェア
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
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。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 はこれらの製品のテストを行っておりません。したがって IBM 以外の製品に関するパフォーマンス、互換性、またはその他のクレームの正確性は確認できません。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にお願いします。

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。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

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。これらのサンプル・プログラムは"現状のまま"提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

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。© Copyright IBM Corp. 1991, 2020。

プライバシー・ポリシーに関する考慮事項:

Software as a Service ソリューションを含む IBM ソフトウェア製品 ("ソフトウェア・オファリング") は、製品の使用に関する情報の収集、エンド・ユーザー・エクスペリエンスの改善、エンド・ユーザーとの対話の調整、その他の目的のために Cookie または他のテクノロジーを使用できます。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

これらを目的とした Cookies を含むさまざまなテクノロジーの使用について詳しくは、IBM のプライバシー・ポリシー (<https://www.ibm.com/privacy>) および IBM のオンライン・プライバシー・ステートメント (<http://www.ibm.com/privacy/details>) のセクション「"Cookies、Web Beacons、その他のテクノロジー"」、および "IBM ソフトウェア製品と Software-as-a-Service のプライバシー・ステートメント" (<http://www.ibm.com/software/info/product-privacy>) を参照してください。

商標

IBM、IBM ロゴ、および [ibm.com](http://www.ibm.com)® は、世界の多くの国で登録された International Business Machines Corporation の商標です。世界中の多くの国で登録されています。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtml をご覧ください。

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 Oracle やその関連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。

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。

他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

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)
- 「ISO 1989:1985, *Programming languages - COBOL*」は「ISO/IEC 1989/AMD1:1992, *Programming languages - COBOL: Intrinsic function module*」および「ISO/IEC 1989/AMD2:1994, *Programming languages - Correction and clarification amendment for COBOL*」に改訂されました。
- ANSI X3.172-2002, *American National Standard Dictionary for Information Systems*
- INCITS/ISO/IEC 1989-2002, *Information technology - Programming languages - COBOL*
- INCITS/ISO/IEC 1989:2014, *Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

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。

A

簡略複合比較条件 (* abbreviated combined relation condition)

連続する一連の比較条件の中で、共通のサブジェクトまたは共通のサブジェクトと共通の比較演算子を明示的に省略したことにより生ずる複合条件。

アベンド

プログラムの異常終了。

2 GB 境界より上

いわゆる 2 GB 境界より上にあるストレージ。このストレージのアドレス指定は、AMODE 64 プログラムでのみ可能です。

16 MB 境界より上 (above the 16 MB line)

いわゆる 16 MB 境界より上かつ 2 GB 境界より下にあるストレージ。このストレージのアドレス指定は、AMODE 24 プログラムでは不可能です。1980 年代に IBM が MVS/XA アーキテクチャーを導入するまで、プログラムの仮想ストレージは 16MB に制限されていました。AMODE 24 としてリンク・エディットされたプログラムは、仮想ストレージ・ラインの下に保持されているかのように、16 MB のスペースしかアドレス指定できません。VS COBOL II 以降、プログラムに AMODE 31 を指定して、16 MB 境界より上にロードできるようになりました。

アクセス・モード (* access mode)

ファイル内のレコードを操作するに当たって使用する方式。

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 (,) によるデータ項目における小数点位置の物理表現。

実文書エンコーディング

XML 文書のエンコード・カテゴリーで、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

英字名 (* alphabet-name)

ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内のユーザー定義語で、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。

英字 (* alphabetic character)

英字またはスペース文字。

英数字の位置

「文字位置 (character position)」を参照。

英字データ項目

記号 A のみを含む PICTURE 文字ストリングを使用して記述されるデータ項目。英字データ項目には USAGE DISPLAY がある。

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。

英数字データ項目

USAGE DISPLAY として暗黙的または明示的に記述され、英数字、英数字編集、または数字編集のカテゴリを持つデータ項目への一般参照。

英数字編集データ項目

記号 A または X の少なくとも 1 つのインスタンスと、単純挿入記号 B、0、または / の少なくとも 1 つを含む PICTURE 文字ストリングによって記述されるデータ項目。英数字編集データ項目には USAGE DISPLAY がある。

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

GROUP-USAGE NATIONAL 節なしで定義されるグループ項目。INSPECT、STRING、および UNSTRING などの操作の場合、英数字グループ項目は、グループの実際の内容に関係なく、そのすべての内容が USAGE DISPLAY として記述されているかのように処理される。グループ内の基本項目 (MOVE CORRESPONDING、ADD CORRESPONDING、INITIALIZE など) の処理を必要とする操作の場合、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る。

英数字リテラル

'、"、X'、X"、Z'、Z" の各セットからの開始区切り文字を持つリテラル。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。

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、その内容が索引付きファイルの中のレコードを識別するもの。

ANSI (米国規格協会)

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。

引数

(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) 呼び出されたプログラムまたは呼び出されたメソッドに値を渡すために使用される、CALL または INVOKE ステートメントの USING 句のオペランド。

算術式 (* arithmetic expression)

数値リテラル、数値基本項目 (算術演算子で区切られた ID とリテラルなど) を表す ID、1 つの算術演算子で区切られた 2 つの算術式、または括弧で囲まれた算術式。

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、そこで指定された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)

単一のキャラクター、または以下の Secure Electronic Transaction に属する固定 2 文字の結合。

文字	意味
+	追加
-	減算

文字	意味
*	乗算
/	除算
**	指数

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は、ADD、COMPUTE、DIVIDE、MULTIPLY、SUBTRACT である。

アレイ

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。

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。

ASCII

情報交換用米国標準コード。基準コードは、7 ビットの符号化キャラクター (パリティ検査を含む 8 ビット) に基づく符号化キャラクターセットを使用します。この基準は、データ処理システム、データコミュニケーションシステム、および関連付けられている装置間の情報交換に使用されます。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 DBCS

「2 バイト ASCII (*double-byte ASCII*)」を参照。

割り当て名

COBOL ファイルの編成を識別する名前で、システムがこれを認識する際に使用する。

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。

AT END 条件

特定の条件下で、READ、RETURN、または SEARCH ステートメントの実行中に発生する条件。

- READ ステートメントは、ファイル内に次の論理レコードが存在しない場合、相対レコード番号の有効数字の数が相対キー・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できない場合に、順次アクセス・ファイルに対して実行される。
- RETURN ステートメントは、関連付けられたソート・ファイルまたはマージ・ファイルに次の論理レコードが存在しない場合に実行される。
- SEARCH ステートメントは、関連する WHEN 句のいずれかで指定された条件を満たさずに検索操作が終了すると実行される。

B

基本文字セット

言語のワード、文字ストリング、および区切り文字の作成時に使用される基本的な文字セット。基本的な文字セットはシングルバイトの EBCDIC で実装されている。拡張文字セットには DBCS 文字が含まれ、コメント、リテラル、ユーザー定義語で利用できる。

85 COBOL 標準における COBOL 文字セットと同義。

バッチ・コンパイル

プログラム・シーケンスと同義。

ビッグ・エンディアン

メインフレームおよび AIX ワークステーションがバイナリー・データおよび UTF-16 文字を保存するとき使用するデフォルト形式。この形式では、バイナリー・データ項目の最下位バイトが最高位アド

レスにあり UTF-16 文字の最下位バイトが最高位アドレスにあります。 リトル・エンディアン (*little-endian*) と比較。

バイナリー項目

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。 項目の左端のビットは、演算符号。

二分探索

二分探索の各段階では、データ・エレメント集合が 2 つに分割される。 数が奇数の場合は適切なアクションが取られる。

* ブロック

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。

ブール条件

ブール条件は、ブール・リテラルが true であるか false であるかを決定する。 ブール条件は定数条件式でのみ使用できる。

ブール・リテラル

true 値を示す B'1'、または false 値を示す B'0' のどちらか。 ブール・リテラルは定数条件式でのみ使用できる。

ブレークポイント

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。

バッファ

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。

組み込み関数

組み込み関数 (*intrinsic function*) を参照。

ビジネス・メソッド

ビジネス・ロジックまたはアプリケーションのルールをインプリメントする Enterprise Bean のメソッド。 (Oracle)

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。

バイトオーダーマーク (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。

バイトコード

Java コンパイラーによって生成され、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。 (Oracle)

C

呼び出し可能サービス

言語環境では、従来の言語環境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。 言語環境の規約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。

呼び出し先プログラム

CALL ステートメントのオブジェクトになるプログラム。 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。

呼び出し側プログラム (* calling program)

別のプログラムに対して CALL を実行するプログラム。

標準分解

複数の Unicode 文字を使用して単一の合成済み Unicode 文字を表す方法。正準分解は通常、ラテン文字と発音区別記号を分離するために使用され、ラテン文字と発音区別記号が個別に表現される。合成済み Unicode 文字とその標準分解を表す例については、合成済み文字 (*precomposed character*) を参照。

ケース構造

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。

カタログ式プロシージャ

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。

CCSID

コード化文字セット ID (*coded character set identifier*) を参照。

世紀ウィンドウ

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。

- ウィンドウ操作組み込み関数 DATE-TO-YYYYMMDD、DAY-TO-YYYYDDD、および YEAR-TO-YYYY の場合は、*argument-2* を使用して世紀ウィンドウを指定する。
- 言語環境呼び出し可能サービスについては、CEEScen で世紀ウィンドウを指定する。

* 性格

言語のそれ以上分割できない基本単位。

文字エンコード・ユニット

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 とも呼ばれる。

USAGE NATIONAL の場合、文字エンコード・ユニットは UTF-16 の 1 つの 2 バイト・コード・ポイントに対応する。

USAGE DISPLAY の場合、文字エンコード・ユニットは 1 バイトに対応する。

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・ポイントに対応する。

文字位置

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。

- 英数字の文字位置 (USAGE DISPLAY で表される文字の場合)
- DBCS 文字位置 (USAGE DISPLAY-1 で表される DBCS 文字の場合)
- 国別文字位置 (USAGE NATIONAL で表される文字の場合)。文字エンコード・ユニット (UTF-16 の場合) と同義。

文字セット

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ません。コード化文字セット (*coded character set*) も参照。

文字ストリング

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連続した文字のシーケンス。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。

チェックポイント

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場所。

* クラス

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 (オブジェクト指向)

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。

クラス階層

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 段落の項目。この項目には、クラス名を指定し、選択した属性をクラス定義に割り当てる文節が含まれる。

クラス名

オブジェクト指向 COBOL クラス定義の名前。

* クラス名

ENVIRONMENT DIVISION の SPECIAL-NAMES 段落で定義されるユーザー定義語。この語は、データ項目の内容がクラス名の定義にリストされている文字のみで構成されている命題 (真の値を定義することができる) に名前を割り当てる。

クラス・オブジェクト

クラスを表す実行時オブジェクト。

文節 (* 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。

クライアント

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。

COBOL 文字セット

COBOL 構文を作成する際に使用される文字セット。完了した COBOL キャラクター Secure Electronic Transaction は、次の文字で構成されています。

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
\$	通貨符号
,	コンマ

文字	意味
;	セミコロ
.	ピリオド (小数点、終止符)
"	引用符
'	アポストロフィ
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロ
-	下線

COBOL ワード (* COBOL word)

ワード (word) を参照。

コード・ページ

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。例えば、作業場上の英語の IBM 符号化ページの 1 つは IBM-1252 であり、ホスト上には IBM-1047 というコード・ページがあります。コード化キャラクター *Secure Electronic Transaction*。

コード・ポイント

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

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 (CCSID) (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。

照合シーケンス (* collating sequence)

コンピューターに受け入れ可能な文字のシーケンスで、ソート、マージ、比較、および索引付きファイルの順次処理を目的として順序付けしたもの。

* 列

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。

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である条件。条件 (condition) および 複合否定条件 (negated combined condition) も参照。

文字組み

他の前後の Unicode 文字を変更するために使用される Unicode 文字。通常、結合文字は、ラテン語文字を変更するために使用される Unicode 発音区別符号。合成済み文字 (precomposed character) を参照して、ラテン語文字 U+0061 (a) とともに使用される結合文字 U+0308 (¨) の例を確認すること。

コメント記入項目 (* comment-entry)

文書化のために使用され、実行には影響しない IDENTIFICATION DIVISION 内の項目。

コメント行

行の標識域内のアスタリスク (*) または、プログラム・テキスト域 (領域 A と領域 B) の最初の文字ストリングとしての、アスタリスクの後の大なり記号 (>) で表されるソース・プログラム行と、その行の領域 A と領域 B にあるコンピューターの文字セット 後続の の任意の文字。コメント行は、文書化にのみ

役立つ。行の標識域に斜線 (/) で表される特殊な形式のコメント行と、その行の領域 A および領域 B にあるコンピューターの文字セットからの文字があると、コメントの出力前にページが排出される。

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。

コンパイル・グループ

プログラム・シーケンスと同義。

コンパイル単位

個別にコンパイルできる COBOL ソース コードの単位: プログラム、クラス、ユーザー定義関数、またはプロトタイプ定義。ソースユニットとも呼ばれます。

コンパイル変数

特定のリテラル値のシンボル名、または DEFINE ディレクティブまたは DEFINE コンパイラー・オプションによって指定されたコンパイル時の算術式の値。

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 (* compile time)

COBOL コンパイラーによって、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。

コンパイル単位 (compile unit)

最初の実行命令から始まり、最後の実行命令まで続く実行プログラムユニット。「コンパイルユニット」には、C プログラムや AMODE64 COBOL プログラムのように、最初の実行命令の前に存在する可能性のある非実行命令 (DC) は含まれない。プログラム単位 (program unit) と同義。

コンパイルユニットアドレス

コンパイルユニットにおける最初の実行可能命令のアドレス。CEEDUMP トレースバックで見つかったプログラム・ユニット・アドレス (PU Addr) と同義。

コンパイル単位オフセット

コンパイルユニットの最初の実行命令のアドレスからアベンドのアドレスまでのオフセット。

CEEDUMP トレースバックで見つかったプログラム・ユニット・オフセット (PU Offset) と同義です。

コンパイル時の算術式

DEFINE ディレクティブおよび EVALUATE ディレクティブ、または定数条件式で指定される算術式のサブセット。コンパイル時演算式における、正規演算式との違い:

- ・ 指数演算子を指定することはできません。
- ・ オペランドはすべて、整数リテラルか、すべてのオペランドが整数リテラルである演算式でなければなりません。
- ・ 式はゼロによる除算が発生しないように指定する必要があります。

コンパイラー

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。

コンパイラー指示ステートメント (compiler-directing statement)

コンパイル時にコンパイラーに特定の処置を行わせるステートメント。標準のコンパイラー指示ステートメントは、COPY、REPLACE、USE である。

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 ODO

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るデータ項目には、**DEPENDING ON** オプションを指定した **OCCURS** 節によって記述される従属データ項目が含まれる。
- 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。

コンポーネント

(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、コンポーネントを作成するための、Oracle が提供するアーキテクチャーである。

合成形式 (composed form)

標準分解による合成済み Unicode 文字の表記。詳しくは、合成済み文字 (*precomposed character*) を参照。

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。

じょうけん

言語環境によって使用可能にされる、あるいは認識される例外。したがって、ユーザー条件処理ルーチンと言語条件処理ルーチンの活動化に適している。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

ある真の値が決定される実行時のデータの状況。本書において、"条件" (*condition-1*、*condition-2*) に関連して使用される場合。) 次のいずれかである。オプションとして括弧で囲まれた単純条件からなる条件式、あるいは、単純条件、論理演算子、および括弧の構文的に正しい組み合わせ (真理値を判別できる) からなる複合条件。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。

条件式 (* conditional expression)

EVALUATE、IF、PERFORM、または SEARCH ステートメントで指定された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。

条件付き句 (* 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 つまたは複数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。

* CONFIGURATION SECTION

ソース・プログラム、オブジェクト・プログラム、およびクラス定義の全体的な仕様を記述する ENVIRONMENT DIVISION のセクション。

コンソール

オペレーター・コンソールに関連する COBOL 環境名。

定数条件式

IF ディレクティブまたは EVALUATE ディレクティブの WHEN 句の中で使用できる条件式のサブセット。

定数条件式は、以下の項目のいずれかでなければなりません。

- 両方のオペランドがリテラルであるか、リテラル項のみを含む算術式である比較条件。条件は比較条件の規則に従う必要があり、以下の追加事項があります。
 - オペランドは同じカテゴリーでなければなりません。算術式は数値カテゴリーです。
 - リテラルが指定され、それらが数値リテラルでない場合、関係演算子は “IS EQUAL TO”、“IS NOT EQUAL TO”、“IS =”、“IS NOT =”、または “IS <>” になる。

比較条件 (*relation condition*) も参照。

- 定義済み条件。定義済み条件 (*defined condition*) も参照。
- ブール条件。ブール条件 (*boolean condition*) も参照。
- 前述の単純条件の形式を、AND、OR、および NOT を使用して複合条件に結合することによって形成した複合条件。簡略複合比較条件を指定することはできません。複合条件 (*complex condition*) も参照。

コンテインド・プログラム

別の COBOL プログラムにネストされている COBOL プログラム。

連続項目 (* contiguous items)

DATA DIVISION 内の連続した項目によって記述され、相互に一定の階層関係を持つ項目。

コピーブック

COPY ステートメントを使用してコンパイル時にソース・プログラムに組み込まれる一連のコードが入っているファイルまたはライブラリー・メンバー。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。

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。

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。

通貨記号の値

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。通貨符号値は、CURRENCY コンパイラー・オプション、または ENVIRONMENT DIVISION の SPECIAL-NAMES 段落の CURRENCY SIGN 節のいずれかによって定義できる。CURRENCY SIGN 節が指定されておらず、NOCURRENCY コンパイラー・オプションが有効な場合は、ドル記号 (\$) がデフォルトの通貨符号値として使用される。通貨記号 (*currency symbol*) も参照。

通貨記号

数字編集項目内の通貨符号値の位置を示すために PICTURE 節で使用する文字。通貨記号は、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落で CURRENCY コンパイラー・オプションまたは CURRENCY SIGN 節のいずれかによって定義できる。CURRENCY SIGN 節が指定されず、NOCURRENCY コンパイラー・オプションが有効な場合、デフォルトの通貨符号値および通貨記号としてドル記号 (\$) が使用される。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。

現行レコード (* current record)

ファイル処理において、ファイルに関連するレコード域の中で使用可能なレコード。

現行ボリューム・ポインター (* current volume pointer)

順次ファイルの現行のボリュームを指している概念上のエンティティ。

D

データ節 (* data clause)

COBOL プログラムの DATA DIVISION 内のデータ記述記入項目に現れる文節で、データ項目の特定の属性を記述する情報を提供する。

データ記述記入項目 (* data description entry)

COBOL プログラムの DATA DIVISION 内の記入項目。必要に応じて、レベル番号の後にデータ名を付け、その後に一連のデータ文節を続けることによって構成される。

DATA DIVISION

プログラムまたはメソッドによって処理されるデータを記述する COBOL プログラムまたはメソッドの部。使用されるファイルとその中に含まれるレコード、必要とされる内部 WORKING-STORAGE レコード、COBOL 実行単位内の複数のプログラムで使用可能になるデータ。

データ項目 (* data item)

COBOL プログラムにより、または関数評価の規則により、定義されたデータの単位 (リテラルを除く)。

データ・セット

「ファイル (*file*)」の同義語。

データ名 (* data-name)

データ記述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。

DBCS

2 バイト文字セット (*double-byte character set (DBCS)*) を参照

DBCS 文字

IBM 2 バイト文字セット (DBCS) に定義された任意の文字。

DBCS 文字位置

「文字位置 (*character position*)」を参照。

DBCS データ項目

少なくとも 1 つの記号 G、または NSYMBOL (DBCS) コンパイラー・オプションが有効な場合は少なくとも 1 つの記号 N を含む PICTURE 文字ストリングによって記述されるデータ項目。DBCS データ項目には USAGE DISPLAY-1 がある。

デバッグ行 (* debugging line)

行の標識区域に文字 D がある行のこと。

デバッグ・セクション (* debugging section)

USE FOR DEBUGGING ステートメントを含むセクション。

宣言文 (* declarative sentence)

分離文字ピリオドで終了する単一の USE ステートメントで構成されるコンパイラー指示文。

宣言部分 (* declaratives)

PROCEDURE DIVISION の先頭に書き込まれる特殊な目的を有する 1 つ以上のセクションの集合。最初のセクションの前にはキーワード DECLARATIVE が付き、最後のセクションの後にはキーワード END DECLARATIVES が続く。1 つの宣言は、セクション・ヘッダーと次に続く USE コンパイラー指示文、その後に続く関連する段落から構成される。この段落は、なくてもよいし、1 個でも複数個でもよい。

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。

定義条件

コンパイル変数が定義されているかどうかをテストするコンパイル時条件。定義条件は、IF ディレクティブまたは EVALUATE ディレクティブの WHEN 句で指定される。

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。

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。

従属領域

IMSにおいて、メッセージ・ドリブン・プログラム、バッチ・プログラム、またはオンライン電力/ガスを含む多重仮想ストレージ 仮想ストレージ地域。

降順キー (* descending key)

データ項目を比較する際の規則に一致するように、最高のキー値から始めて最低のキー値へとデータを順序付けている値に付けられるキー。

数字

0 から 9 までの数値のいずれか。COBOL では、用語はその他のシンボルを参照するために使用されることはありません。

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項目に指定された用途によって異なる。

直接アクセス (* direct access)

前回アクセスしたデータへの参照には依存せず、プロセスがデータの位置にのみ依存する方法で、データを記憶装置から取得したり、データを記憶装置に入れる機能。

表示浮動小数点データ項目

USAGE DISPLAY として暗黙的または明示的に記述され、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つデータ項目。

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。

部のヘッダー (* 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.
```

DLL

ダイナミック・リンク・ライブラリー (DLL) (*dynamic link library (DLL)*) を参照。

DLL アプリケーション

インポートしたプログラム、関数、または変数を参照するアプリケーション。

DLL リンク

DLL および NODYNAM オプションを指定してコンパイルされたプログラム内の CALL。CALL は、別のモジュール内のエクスポートされた名前、または別のモジュール内で定義されたメソッドの INVOKE に解決される。

DO 構文

構造化プログラミングでは、DO ステートメントを使用して、プロシージャ内いくつかのステートメントをグループ化する。COBOL では、インライン PERFORM ステートメントが同じように機能する。

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、PERFORM ステートメントで使用される TEST AFTER 句が同じように機能する。

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、PERFORM ステートメントで使用される TEST BEFORE 句が同じように機能する。

文書タイプ宣言

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。

文書タイプ定義 (DTD)

XML 文書のクラスの文法。文書タイプ宣言を参照。

全角アスキー

DBCS 文字および 1 バイト ASCII 文字を含む IBM の文字セット (「ASCII DBCS」 と呼ばれる)。

全角 EBCDIC

DBCS 文字および 1 バイト EBCDIC 文字を含む IBM の文字セット (「EBCDIC DBCS」 と呼ばれる)。

2 バイト文字セット (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。

DWARF

DWARF は UNIX International Programming Languages Special Interest Group (SIG) で開発された。言語に依存しないデバッグ情報を提供することにより、さまざまな言語の、統一された方法でのシンボリックなソース・レベル・デバッグのニーズを満たすように設計されている。DWARF ファイルには、さまざまなエレメントに編成されたデバッグ・データが含まれている。詳細については、[DWARF/ELF エクステンション ライブラリー・リファレンス](#) の [DWARF プログラム情報を参照](#)。

動的アクセス (* dynamic access)

特定の論理レコードを大容量記憶ファイルから順序に関係なく取得したり、ファイル内に配置したりすることができるアクセス・モードで、同じ OPEN ステートメントの範囲内でファイルから論理レコードを順次取得できる。

動的呼び出し

DYNAM オプションを指定してコンパイルされているプログラム内の CALL *literal* ステートメント および NODLL オプション、またはプログラム内の CALL *identifier* ステートメント NODLL オプションを指定してコンパイルされたもの。

動的長

実行時に変化する可能性がある論理長を持つ項目を記述する形容詞。

動的長基本項目

データ宣言項目に DYNAMIC LENGTH 節が含まれる基本データ項目。

ダイナミック・レングス・グループ

従属する動的長基本項目を含むグループ項目。

ダイナミック・リンク・ライブラリー (dynamic link library) (DLL)

リンク時ではなく、ロード時または実行時にプログラムにバインドされる実行可能コードおよびデータが入ったファイル。複数のアプリケーションが DLL 内のコードおよびデータを同時に共用することができる。DLL はプログラムの実行ファイルの一部ではないが、実行ファイルを正しく実行するために必要となることがある。

動的ストレージ域 (DSA) (dynamic storage area (DSA))

動的に獲得されるストレージであり、レジスタ保管域、および動的ストレージ割り振りに使用可能な区域 (プログラム変数など) から構成される。DSA は、プログラムまたは関数が呼び出されるときに割り振られ、呼び出しインスタンスの継続時間の間持続します。DSA は通常、言語環境によって管理されるスタック・セグメント内に割り振られる。

E

EBCDIC (拡張 2 進化 10 進交換コード) (* 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 (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。

EBCDIC DBCS

「2 バイト *EBCDIC (double-byte EBCDIC)*」を参照。

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
	スペース
0	0
+	正符号
-	負符号
CR	貸方
DB	借方
Z	ゼロの抑止
*	小切手変造防止
\$	通貨符号
,	コンマ (小数点)
.	ピリオド (小数点)
/	斜線 (スラッシュ)

EGCS

拡張図形文字セット (*extended graphic character set*) (EGCS) を参照。

EJB

Enterprise JavaBeans を参照。

EJB コンテナ

J2EE アーキテクチャーの EJB コンポーネント契約を実装するコンテナ。この契約は、セキュリティ、並行性、ライフ・サイクル管理、トランザクション、デプロイメント、およびその他のサービスを含んでいるエンタープライズ Bean 用のランタイム環境を指定します。EJB コンテナは、EJB サーバーまたは J2EE サーバーによって提供される。(Oracle)

EJB サーバー

EJB コンテナにサービスを提供するソフトウェア。EJB サーバーは、1 つ以上の EJB コンテナをホストできる。(Oracle)

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。

基本項目 (* elementary item)

論理的にそれ以上細分できないものとして記述されているデータ項目。

カプセル化

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。

エンクレーブ

言語環境のもとで実行される場合、エンクレーブは実行単位に類似している。エンクレーブは、LINK、C 言語の `system()` 関数を使うことで、他のエンクレーブを作ることができる。

エンコード・ユニット

文字エンコード・ユニット (*character encoding unit*) を参照。

END CLASS マーカー

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。クラス終了マーカーは次のとおり。

```
END CLASS class-name.
```

END METHOD マーカー

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ーは次のとおり。

```
END METHOD method-name.
```

* 終了 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ジャが存在しない物理的な位置。

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.
```

エンタープライズ Bean

ビジネス・タスクを実装し、EJB コンテナに存在するコンポーネント。(Oracle)

エンタープライズ JavaBeans

オブジェクト指向の分散エンタープライズ・レベル・アプリケーションの開発とデプロイメントのために、Oracle によって定義されたコンポーネント・アーキテクチャー。

* エントリー

分離文字ピリオドで終了す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。

エントリオフセット

プログラム起動時に制御が渡された最初の実行命令のアドレスから、アベンドのアドレスまでのオフセット。

入り口点

あるルーチンを実行するためにそのルーチンに入ったときに実行される最初の命令のアドレスまたはラベル。ロード・モジュール／プログラム・オブジェクトの中では、ロード・モジュールが呼び出されたときに制御が渡される場所である。

エントリー・ポイントは、プログラムのメイン・エントリー・ポイント（最初に実行可能な命令）であることもあれば、プログラムが呼び出されたときに制御が渡される、プログラム内で定義された代替エントリー・ポイントであることもある。

環境文節 (* environment clause)

ENVIRONMENT DIVISION 項目の一部として表示される文節。

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面を提供する。

環境名

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、そのニーモニック名は、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。

環境変数

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。

エスケープ・シーケンス

ストリング・文字列およびキャラクターリテラル内の特定の特殊文字を表すために使用される文字の一連。

エスケープするシーケンスは 2 つ以上のキャラクターで構成され、最初の文字は「エスケープ文字」と呼ばれるバックスラッシュ (\) 文字です。残りの文字は、エスケープ・シーケンスの変換処理を決定します。例えば、\n は、改行文字を意味するエスケープ・シーケンスである。

エスケープ・シーケンスは、C、C++、Java、Python などのプログラミング言語で使用する。COBOL には、「エスケープ・シーケンス」または「エスケープ文字」というコンセプトがありません。COBOL リテラル内で特殊文字を扱うには、Enterprise COBOL for z/OS 言語解説書の基本英数字リテラルおよび DBCS リテラルを参照。

実行時

実行時 (run time) を参照。

ランタイム環境 (execution-time environment)

ランタイム環境 (runtime environment) を参照。

明示範囲終了符号 (* 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。

指数

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。

* 表現

算術式または条件式。

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 句または 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。

拡張グラフィック文字セット (EGCS)

それぞれの図形文字を表すために 2 バイトを必要とする図形文字セット (漢字文字セットなど)。現在は、2 バイト文字セット (DBCS) と呼ばれるようになった。

拡張可能マークアップ言語

XML を参照。

拡張 (extensions)

85 COBOL 標準で説明されているものに加えて、IBM コンパイラーによってサポートされる COBOL 構文およびセマンティクス。

外部コード・ページ

XML 文書で、CODEPAGE コンパイラー・オプションで指定された値。

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 10 進データ項目

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

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。

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、名前が付けられたハードウェア装置またはソフトウェアで、2つの選択的な状態のうち一方が存在することを示すために使用される。

F

ファクトリー・データ

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 で宣言される。Java private 静的データと同等。

ファクトリー・メソッド

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 静的メソッドと同等。これらは通常、オブジェクトの作成をカスタマイズするために使用されます。

形象定数 (* figurative constant)

ある予約語を使用することによって参照されるコンパイラ生成の値。

* ファイル

論理レコードの集合。

ファイル属性対立条件 (* file attribute conflict condition)

あるファイルで入出力操作を実行する試みが失敗し、プログラムの中でそのファイルに対して指定したファイル属性が、そのファイルの固定属性と一致していないこと。

ファイル文節 (* file clause)

ファイル記述項目 (FD 項目) およびソート・マージ・ファイル記述項目 (SD 項目) のいずれかの一部として現れる DATA DIVISION 文節。

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 (File-Control)

ソース・プログラムで用いられるデータ・ファイルが宣言されている環境部の段落の名前。

ファイル制御ブロック (FCB) (file control block)

I/O ルーチンのアドレス、それらがどのようにオープンおよびクローズされたかに関する情報、およびファイル情報ブロック (FIB) へのポインターを含むブロック。

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。

FILE-CONTROL 段落

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。

ファイル記述記入項目 (* file description entry)

DATA DIVISION の FILE SECTION 内の中にある記入項目項目。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一連のファイル文節から構成される。

ファイル名 (* file-name)

DATA DIVISION の FILE SECTION 内のファイル記述項目またはソート・マージ・ファイル記述項目に記述されているファイル結合子に名前を付けるユーザー定義語。

ファイル編成 (* file organization)

ファイルの作成時に確立される永続的な論理ファイル構造。

ファイル位置標識

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。

* 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。

ファイル・システム

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。

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、ファイルの編成 (順次、相対、指標付き)、基本レコード・キー、代替レコード・キー、コード・セット、最小および最大のレコード・サイズ、レコード・タイプ (固定長、可変長)、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、レコード区切り文字がある。

* 固定長レコード

ファイル記述項目またはソート・マージ記述項目が、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。

定点項目

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。

フローティングコメント標識 (*>)

浮動コメント標識がプログラムのテキスト域 (領域 A プラス領域 B) 内の最初の文字ストリングである場合は、この行がコメント行であることを示します。また、浮動コメント標識がプログラムのテキスト領域内の 1 つ以上の文字ストリングの後にある場合は、インライン・コメントを示します。

浮動小数点

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

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。

* フォーマット

データの集合の特定の配列。

* 機能

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。

関数 ID (* function-identifier)

関数を参照する文字ストリングと分離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。

関数名

必要な引数を伴って関数の値を決定する呼び出しを行うメカニズムに付けられる名前を表すワード。

関数ポインター・データ項目

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。

G

ガーベッジ・コレクション

参照されなくなったオブジェクト用のメモリーが Java ランタイム・システムによって自動的に解放されること。

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。

グローバル参照

メソッドの有効範囲外にあるオブジェクトの参照。

グループ項目

(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。

グループ分離子

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。デフォルトの文字はコンマです。

H

ヘッダー・ラベル

(1) 記録メディア・ユニットのデータ・レコードの前にある、データ・セットのラベル。(2) 「ファイル開始ラベル (*beginning-of-file label*)」の同義語。

非表示にする(方法)

親クラスで同じメソッド名により定義されたファクトリーまたは静的メソッドを (サブクラスで) 再定義すること。したがって、サブクラスのメソッドは親クラスのメソッドを隠蔽する。

* 高位端

文字ストリングの左端の文字。

ハイパースペース

z/OS 環境で、プログラムがバッファーとして使用できる最大 2 GB までの連続する仮想記憶アドレス範囲。

I

IBM COBOL エクステンション

85 COBOL 標準で説明されているものに加えて、IBM コンパイラーによってサポートされる COBOL 構文およびセマンティクス。

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、メソッドを識別する。IDENTIFICATION DIVISION には、作成者名、インストール、日付を含めることができる。

* 識別子

データ項目に名前を付けるための文字ストリングと分離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。

IGZCBSN

COBOL/370 1.1 のブートストラップ・ルーチン。これは、COBOL/370 1.1 プログラムを含むすべてのモジュールとリンク・エディットする必要がある。

IGZCBSO

COBOL for MVS & VM 1.2、COBOL for OS/390 & VM、Enterprise COBOL のブートストラップ・ルーチン。COBOL for MVS & VM 1.2、COBOL for OS/390 & VM、Enterprise COBOL プログラムを含むすべてのモジュールとリンク・エディットする必要がある。

IGZEBST

VS COBOL II のブートストラップ・ルーチン。これは、VS COBOL II リリース 1 プログラムを含んで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。

ILC

言語間通信。言語間通信は、他の高水準言語を呼び出すかまたは他の高水準言語によって呼び出されるプログラムとして定義されています。アセンブラーは高水準言語と見なされません。このため、アセンブラー言語プログラムへの呼び出しおよびアセンブラー言語プログラムからの呼び出しは ILC と見なされません。

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。

暗黙範囲終了符号 (* 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。

IMS

情報管理システム (Information Management System)。IBM のライセンス製品。IMS は、階層データベース、データ通信 (DC)、変換処理、およびデータベースのバックアウトとリカバリーをサポートする。

* インデックス

コンピューターのストレージ域またはレジスター。ここにはテーブル内の個々のエレメントを識別するものを表現する内容が入る。

索引データ項目 (* index data item)

指標名に関連する値を、インプリメントする人が指定した形式で収めることができるデータ項目。

索引付きデータ名 (indexed data-name)

データ名とそれに続く括弧で囲まれた 1 つまたは複数の指標名から構成される ID。

索引付きファイル (* indexed file)

指標付き編成のファイル。

索引編成 (* indexed organization)

各レコードがそのレコードの中にある 1 つまたは複数のキー値によって識別される永続的な論理ファイル構造。

指標付け

指標名を使用した添え字付けと同義。

索引名 (* index-name)

特定のテーブルに関連付けられた指標に付ける名前を表すユーザー定義語。

継承

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。Enterprise COBOL 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。

継承の階層

クラス階層 (*class hierarchy*) を参照。

初期プログラム (* initial program)

実行単位内にプログラムが呼び出されるたびに初期状態に置かれるプログラム。

初期状態 (* initial state)

プログラムが実行単位の中に呼び出された最初期のそのプログラムの状態。

インライン

ルーチン、サブルーチン、または他のプログラムに分岐せずに、連続的に実行されるプログラム内の命令。

インライン・コメント

インライン・コメントは、プログラムのテキスト域にある、前に1つ以上の文字ストリングが付いた浮動コメント標識 (*>) で示され、コンパイル・グループの任意の行に書き込むことができます。浮動コメント標識に続く、領域 B の終わりまでの文字はすべて、コメント・テキストです。

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 句または 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。

* 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。

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 句または TO OFF 句付き)、START、WRITE がある。

入力プロシージャ (* input procedure)

ソート対象として指定されたレコードの解放を制御するために、形式 1 SORT ステートメントの実行中に制御が渡されるステートメントのセット。

インスタンス・データ

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。

* 整数

(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

カテゴリーが数字で、その定義が小数点の右側に桁位置を持たない関数。

Interactive System Productivity Facility (ISPF)

TSO または VM ユーザーに対してメニュー方式のインターフェースを提供する IBM ソフトウェア・プロダクト。ISPF には、ライブラリー・ユーティリティ、強力なエディター、およびダイアログ管理が組み込まれています。

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。

中間結果

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。

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 10 進データ項目

USAGE PACKED-DECIMAL または USAGE COMP-3 として記述され、項目を数値として定義する PICTURE 文字ストリング (9、S、P、または V の有効な記号の組み合わせ) 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

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する。COMP-2 は、倍精度浮動小数点データ項目を定義する。内部浮動小数点データ項目に関連した PICTURE 節はない。

* レコード内データ構造

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。

組み込み関数

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。

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。

* 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 エントリー

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。

* I-O モード

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 句または 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項目で FILE STATUS 節を使用することにより、プログラムで利用可能になる。

イザ

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。

ISPF

対話式システム生産性向上機能 (ISPF) (*Interactive System Productivity Facility (ISPF)*) を参照。

繰り返し構造

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。

J

J2EE

Java 2 Platform, Enterprise Edition (J2EE) を参照。

Java 2 Platform, Enterprise Edition (J2EE)

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に、Oracle によって定義された環境。J2EE プラットフォームは、多層の 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ている。(Oracle)

のための Java バッチランチャーとツールキット z/OS (JZOS)

従来のバッチ環境で実行されて z/OS システム・サービスにアクセスする z/OS Java アプリケーションの開発を支援するツールのセット。

Java バッチ処理プログラム (JBP)

オンライン・データベースおよび出力メッセージ・キューにアクセスできる IMS バッチ処理プログラム。JBP はオンラインで実行されますが、バッチ環境のプログラムと同様に、JCL を使用して、または TSO セッション内で開始されます。

Java バッチ処理領域

Java バッチ処理プログラムだけがスケジュールされる IMS 従属領域。

Java データベース接続 (JDBC)

Java プログラムのデータベースへのアクセスを可能にする API を定義する、Oracle の仕様。

Java メッセージ処理プログラム (JMP)

トランザクションによって駆動され、オンライン IMS データベースとメッセージ・キューにアクセスできる、Java アプリケーション・プログラム。

Java メッセージ処理領域

Java メッセージ処理プログラムだけがスケジュールされる IMS 従属領域。

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)

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。

JavaBeans

移植可能で、プラットフォームに依存しない、再使用可能なコンポーネント・モデル。(Oracle)

ジェービーピー

Java バッチ処理プログラム (JBP) (Java batch-processing program (JBP)) を参照。

JDBC

Java Database Connectivity (JDBC) を参照。

jmp

Java メッセージ処理プログラム (JMP) (Java message-processing program (JMP)) を参照。

ジョブ制御言語 (JCL) (job control language (JCL))

ジョブをオペレーティング・システムに識別させ、ジョブの要件を記述するために使われる制御言語。

JSON

JSON (JavaScript Object Notation) とは、単純なデータ交換フォーマットである。

Java VM

Java 仮想マシン (JVM) (Java virtual machine (JVM)) を参照。

JZOS

Java Batch Launcher と Toolkit for z/OS を参照。

K

K

記憶容量に関連して使用されるときは、2 の 10 乗。10 進表記では 1024。

* キー

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。

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。

* キーワード

コンテキスト・センシティブ語または予約語。その語の表示フォーマットがソース単位で使用されるときは、その語は必須である。

K バイト (kilobyte (KB))

1 キロバイトは 1024 バイトに相当する。

L

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。

言語環境プログラム

z/OS 言語環境の省略名。C、C++、COBOL、FORTRAN、および PL/I アプリケーションに共通のランタイム環境およびランタイム・サービスを提供する一連のアーキテクチャー構造およびインターフェース。これは、言語環境準拠のコンパイラーおよびジャワ アプリケーション〜別にコンパイルされたプログラムに必要です。

言語環境に準拠

言語環境 規則に準拠するオブジェクト・コードを生成するコンパイラー製品 (Enterprise COBOL、COBOL for OS/390 & VM、MVS および VM 用 COBOL、C/C++ for MVS & VM、PL/I for MVS & VM など) の特性。

最後に使われた状態

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。

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

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 のレベル標識には、CD、FD、SD がある。

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 までの範囲のレベル番号は、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 桁の数字として書き込むことも、0 の後に有効数字を書き込むこともできる。レベル番号 66、77、および 88 は、データ記述項目の特性を識別する。

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。

ライブラリー・テキスト (* library text)

COBOL ライブラリーの中にある一連のテキスト・ワード、コメント行、インライン・コメント、区切り文字のスペース、または区切り文字の疑似テキスト区切り文字。

リリアン日

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。

* 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。

リンク

(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) 1 つまたは複数のコンピュータ・プログラムのデータ項目または部分を相互に接続すること。例えば、実行可能ファイルを生成するために、リンケージ・エディタによってオブジェクト・プログラムをリンクすること。

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。

リンカー

z/OS バインダー (リンク・エディター) のいずれかを参照する用語。

リテラル

ストリング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、その値が決められる文字ストリング。

リトル・エンディアン

Intel プロセッサが 2 進データおよび UTF-16 文字を保管するために使用するデフォルト形式。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。

ローカル参照

メソッドの有効範囲内にあるオブジェクトの参照。

ロケール

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。

* 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。

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。

* 下位端

文字ストリングの右端の文字。

M

メインプログラム

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。

Make ファイル

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。

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。

* メガバイト

1 メガバイトは 1,048,576 バイトに相当する。

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合。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。

メッセージ処理プログラム (MPP)

トランザクションによって駆動され、オンライン IMS データベースとメッセージ・キューにアクセスできる、IMS アプリケーション・プログラム。

メッセージ・キュー

メッセージがアプリケーション・プログラムによって処理されたり端末に送信されたりする前に、キューに入れられるデータ・セット。

メソッド

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。

メソッド識別記入項目 (* 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文節が入っている。

メソッド起動

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。

メソッド名

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る。

メソッド隠蔽

「隠蔽 (*hide*)」を参照。

メソッド多重定義

「多重定義 (*overload*)」を参照。

メソッドのオーバーライド

「オーバーライド (*override*)」を参照。

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。

モジュール定義ファイル

プログラム・オブジェクト内のコード・セグメントを記述したファイル。

MPP

メッセージ処理プログラム (MPP) (*message-processing program (MPP)*) を参照。

マルチタスキング

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。

マルチスレッド化

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。

N

名前

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

XML 名前空間 (*XML namespace*) を参照。

国別文字

(1) USAGE NATIONAL データ項目または国別リテラル内の UTF-16 文字。(2) UTF-16 で表される任意の文字。

国民性データ

UTF-16 で表されるデータの一般参照。

ナショナル・キャラクター・ポジション

「文字位置 (*character position*)」を参照。

国別データ

「国別文字データ (*national character data*)」を参照。

ナショナルデータ項目

USAGE NATIONAL のカテゴリ国別、国別編集、数字編集のデータ項目。

こくすうじデータこうもく

暗黙的または明示的に USAGE NATIONAL として記述され、PICTURE の記号 9、S、P、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。

全国編集データ項目

記号 N の少なくとも 1 つのインスタンスと、単純挿入記号 B、0、または / の少なくとも 1 つを含む PICTURE 文字ストリングによって記述されているデータ項目。国別編集データ項目には USAGE NATIONAL がある。

国家浮動小数点データ項目

暗黙的または明示的に USAGE NATIONAL として記述され、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つ外部浮動小数点データ項目。

国別グループ項目

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、UNSTRING などの操作で、国別カテゴリ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る。英数字グループ項目内の USAGE NATIONAL データ項目の定義とは対照的に、この処理により、国別文字の埋め込みと切り捨てが確実に正しく行われる。MOVE CORRESPONDING、ADD CORRESPONDING、INITIALIZE など、グループ内の基本項目の処理を必要とする操作の場合、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る。

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。

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。

ネイティブ・メソッド

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。

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。

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。

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。

次の実行可能文 (* 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。

次の実行可能ステートメント (* 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。

次のレコード (* next record)

ファイルの現行レコードに論理的に続くレコード。

独立項目 (* noncontiguous items)

他のデータ項目と階層上の関係を持たない、WORKING-STORAGE SECTION および LINKAGE SECTION 内の基本データ項目。

独立項目 (* noncontiguous items)

別のデータ項目への階層関係がない、WORKING-STORAGE および LINKAGE SECTION の基本データ項目。

非数字項目 (* nonnumeric item)

その内容を、コンピューターの文字セットからの文字の任意の組み合わせで構成して記述することができるデータ項目。特定のカテゴリの非数字項目は、さらに制限された文字セットから形成することができる。

ヌル

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定数。NULLS は、NULL を使用できる場所であればどこでも使用できる。

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。

数値データ項目

(1) 詳細がその中身を、0 から 9 の数字～別に選択された文字で表される量に制限するデータ・アイテム。符号付きの場合、項目には演算符号の +、-、または他の表記の演算符号も含むことができる。(2) カテゴリ数値、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、COMP-5 を持つことができる。

数値編集データ項目

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。数値編集項目は、USAGE DISPLAY または USAGE NATIONAL のいずれかで表すことができる。

数字関数 (* numeric function)

クラスとカテゴリは数字だが、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。

数値項目 (* numeric item)

その内容の記述が、「0」から「9」までの数字から選択された文字で表される値に制限されるデータ項目。符号付きの場合、その項目には +、-、または他の演算符号の表記を入れることもできる。

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。

0**オブジェクト**

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・コード

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の処理に適する。

*** OBJECT-COMPUTER**

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ている ENVIRONMENT DIVISION にある段落の名前。

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・デック

リンケージ・エディターへの入力として適切なオブジェクト・プログラムの部分。「オブジェクト・モジュール (object module)」および「テキスト・デック (text deck)」と同義。

オブジェクト

単一の、場合によっては多数のオブジェクト。COBOL クラス定義の Object 段落における指定に基づいてインスタンス生成される。オブジェクト・インスタンスは、そのクラス定義に記述されたデータおよびすべての継承データのコピーを保持する。オブジェクト・インスタンスに関連付けられたメソッドには、そのクラス定義で定義されたメソッドおよびすべての継承メソッドが含まれる。

オブジェクト・インスタンスは Java クラスのインスタンスにすることができる。

オブジェクト・モジュール (object module)

オブジェクト・デック (object deck) または テキスト・デック (text deck) と同義。

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。

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。

オブジェクト・プログラム

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 コンパイラーがソース・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・リファレンス

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。

古くなったエレメント (* obsolete element)

85 COBOL 標準の COBOL 言語エレメントのうち、2002 COBOL 標準から削除されたもの。

ODO オブジェクト

以下の例では、X は OCCURS DEPENDING ON 節のオブジェクト (ODO オブジェクト) である。

```
WORKING-STORAGE SECTION.  
01 TABLE-1.  
   05 X PIC S9.  
   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

上記の例では、Y は OCCURS DEPENDING ON 節の対象 (ODO 対象) である。テーブル内の ODO 対象の数である Y の値は、X の値によって異なる。

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 句または 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。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、EXTEND のいずれかとして指定する。

* オペランド

(1) オペランドの一般的な定義は、"操作対象のコンポーネント" 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。

操作

オブジェクトに関して要求できるサービス。

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値リテラルに付けられる代数符号。

オプション・ファイル

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められる予約語。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。

* 出力ファイル

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。

出力モード (* output mode)

OUTPUT 句または 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 句または 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。

出力プロシージャ (* output procedure)

形式 1 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。

オーバーフロー条件

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ときに起こる条件。

多重定義

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。

オーバーライド

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。

P

パッケージ

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。

パック 10 進数データ項目

内部 10 進数データ項目 (internal decimal data item) を参照。

埋め込み文字

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。

ページ

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。

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 で使用できる段落ヘッダーは、以下のとおり。

```
PROGRAM-ID. (Program IDENTIFICATION
DIVISION)
```

```
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION
DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

ENVIRONMENT DIVISION で使用できる段落ヘッダーは、以下のとおり。

```
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。

パラメーター

(1) 呼び出し側プログラムと呼び出し先プログラムの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。

永続的な再使用可能 JVM

トランザクション間で JVM をリセットすることによりトランザクション処理用にシリアルに再利用できる JVM。リセット・フェーズでは、JVM が既知の初期状態に復元される。

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。

物理レコード (* physical record)

ブロック (*block*) を参照。

ポインター・データ項目

アドレス値を保管できるデータ項目。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・データ項目と等しいかどうかを比較したり、他のポインター・データ項目に内容を移動することができる。

ポート

(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

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。

合成済み文字

標準分解により複数の Unicode 文字を使用して表すことができる単一の Unicode 文字。合成済み文字は合成文字形式と同じ物理表記を持たない。例えば、Unicode 文字 U+00E4 (ä) は、Unicode 文字 U+0061 + U+0308 (ä) (ラテン語小文字 a + 結合発音区別符号) の組み合わせとして表すことができる合成済み文字。通常、合成済み文字は、発音区別符号を持つラテン語文字や他の結合文字を表すために使用される。

事前初期設定

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。

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。

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。部門番号は 0 から 9 の文字だけを含めるものとします。部門数値は、1 桁または 2 桁のいずれかで表すことができます。

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。

プロシージャ (* procedure)

PROCEDURE DIVISION 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・ブランチ・ステートメントには、ALTER、CALL、EXIT、EXIT PROGRAM、GO TO、MERGE (OUTPUT PROCEDURE 句付き)、PERFORM、SORT (INPUT PROCEDURE 句または OUTPUT PROCEDURE 句付き)、XML PARSE がある。

PROCEDURE DIVISION

問題を解決するための指示を含む COBOL の割り算。

プロシージャ統合

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 の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。

プロシージャ・ポインター

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 節を付けて定義したデータ項目が、プロシージャへの入り口点のアドレスを収める。

プロシージャ・ポインター・データ項目

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。通常、COBOL および 言語環境プログラムとの通信に使用されます

プロセス

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。

プログラム

(1) コンピューターによる処理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ます。

プログラム名

IDENTIFICATION DIVISION と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。

プログラム名

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。

プログラム・ユニット

コンパイルユニットの同義語。

プログラム単位アドレス

コンパイル・ユニット・アドレスの同義語。CEEDUMP のトレースバックでは「PU Addr」と略される。

プログラム・ユニット・オフセット

コンパイル・ユニット・オフセットの同義語。CEEDUMP のトレースバックでは「PU Offset」と略記される。

プロジェクト

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。

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、インライン・コメント、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隣接した 2 つの等号文字 (==)。

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

Q

QSAM (待機順次アクセス方式) (QSAM (Queued Sequential Access Method))

基本順次アクセス方式 (BSAM) の拡張版。この方式を使用する場合、キューは、処理を待機する入力データ・ブロック、または処理が終了して補助ストレージまたは出力装置への転送を待機する出力データ・ブロックで形成される。

修飾データ名 (* qualified data-name)

データ名と、その後に連結語の OF および 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。

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。

R

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って、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、またはそこに入れたりする論理レコードを識別するアクセス・モード。

* 記録

論理レコード (*logical record*) を参照。

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、レコード域内の現行の文字位置の数は、明示的または暗黙的な 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。

レコード記述記入項目 (* record description entry)

特定のレコードに関連したデータ記述項目全体。「レコード記述 (*record description*)」と同義。

記録モード (recording mode)

ファイル内の論理レコードの形式。レコード・モードは、F (固定長)、V (可変長)、S (スパン)、または U (不定フォーマット) とすることができる。

レコード・キー

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。

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。

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。

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)

RECURSIVE 属性が PROGRAM-ID ステートメントで指定されていれば、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)。

リール

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。

再入可能

プログラムまたはルーチンの属性で、複数のユーザーがプログラム・オブジェクトの単一のコピーを共有できるようにすること。

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。

参照変更

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、新しいカテゴリー英数字、カテゴリー DBCS、またはカテゴリー国別データ項目を定義する方式。

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分離文字の構文的に正しい組み合わせ。区切り用の左括弧区切り文字、左端の文字位置、区切り文字のコロン、任意指定の長さ、および区切り用の右括弧区切り文字を含む。

関係 (* relation)

関係演算子 (*relational operator*) または 比較条件 (*relation condition*) を参照。

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

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。

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
IS NOT >	より大きくない
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
IS NOT <	より小さくない
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
IS >=	より大きいか等しい
IS LESS THAN OR EQUAL TO	より小さいか等しい
IS <=	より小さいか等しい

相対ファイル (* relative file)

相対編成のファイル。

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。この数値は、整数の数値リテラルとして扱われる。

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。

* リソース

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。

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。

再使用可能環境 (reusable environment)

再利用可能な環境は、事前初期化のための古い COBOL インタフェース (RTEREUS ランタイム・オプション)、または 言語環境 インタフェース、CEEPIPI のいずれかを使用して、メイン・プログラムとしてアセンブラ・プログラムを確立するときに作成される。

ルーチン

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。言語環境では、処置、ファンクション、またはサブルーチンのいずれかを参照します。

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ーを識別するユーザー定義語。

ランタイム (* 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。

ランタイム環境

COBOL プログラムが実行される環境。

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 CALL または INVOKE ステートメントによって相互に作用し、実行時に 1 つのエンティティとして機能する複数のオブジェクト・プログラム。

実行単位は、言語環境の用語では別プログラムとも呼ばれます。

S

SBCS

1 バイト文字セット (SBCS) (single-byte character set (SBCS)) を参照。

範囲終了符号

COBOL の予約語で、特定の PROCEDURE DIVISION 文の終わりを示す。明示的 (END-ADD など) にも暗黙的 (区切りピリオド) にもできる。

* セクション

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付けられた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、その後に続く予約語 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 の中にあるセクションに名前を付けるユーザー定義語。

セグメント化

85 COBOL 標準 セグメンテーション・モジュールに基づく Enterprise COBOL の機能。セグメンテーション機能は、セクション・ヘッダーの優先順位番号を使用して、セクションを固定セグメントまたは独立セグメントに割り当てる。セグメント種別は、セグメントに含まれるプロシージャが初期状態の制御を受け取るか、最後に使われた状態の制御を受け取るかに影響を及ぼす。

選択構造

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。

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。

別々にコンパイルされるプログラム (* separately compiled program)

あるプログラムをそこに含まれたプログラムと共に、他のすべてのプログラムとは別個にコンパイルしたときのそのプログラム。

区切り文字 (* 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 文字以上。

区切りコンマ (* 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 (,)。

区切りピリオド (* separator period)

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 つのピリオド (.)。

区切りセミコロン (* 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 (;)。

プログラムの順序

コンパイラに入力できる単一のソース・ファイルに含まれている一連の個別 COBOL プログラム。

プログラム・シーケンスは、バッチ・コンパイルまたはコンパイル・グループとも呼ばれます。

シーケンス構造

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。

順次アクセス (* sequential access)

ファイル内のレコードの並び方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。

逐次探索

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索方法。

セッション Bean

EJB において、クライアントによって作成され、通常は 1 つのクライアント/サーバー・セッションの期間だけ存在する Enterprise Bean。(Oracle)

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。

符号条件 (* sign condition)

データ項目や算術式の代数値が、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。

シグニチャー

(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。

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
- スイッチ状況条件
- 符号条件

条件 (condition) および 単純否定条件 (negated simple condition) も参照。

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。

スラックバイト (レコード内)

複数の基本データ項目を正しく位置合わせするために、コンパイラーによってデータ項目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。正しい位置合わせを行うために遊びバイトが必要なときは、SYNCHRONIZED 節によって、コンパイラーに遊びバイトを挿入させる。

スラックバイト (レコード間)

複数の基本データ項目を正しく位置合わせするために、プログラマーによってファイルのブロック化論理レコードの間に挿入されるバイト。場合によっては、レコード間に遊びバイトを挿入することによってバッファ内で処理されるレコードのパフォーマンスが改善される。

ソート・ファイル (* sort file)

形式 1 SORT ステートメントによってソートされるレコードの集合。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 内の中にある記入項目項目。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一連のファイル文節から構成される。

* 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。

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。

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・プログラム

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 ステートメントで始まり、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。

ソースユニット

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。

特殊文字

以下の Secure Electronic Transaction に属するキャラクター。

文字	意味
+	正符号
-	負符号 (-) (ハイフン)

文字	意味
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
\$	通貨符号
,	コンマ
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符
'	アポストロフィ
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロ
_	下線

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。

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。

特殊レジスター (* special registers)

コンパイラの生成する特定の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。

標準データ・フォーマット (* standard data format)

COBOL データ部でデータの特性を記述するために使用される概念。この概念のもとでは、データの特性は、データが内部的にコンピューターに、または特定の外部メディアに保管される方法に適した形式ではなく、印刷ページ上での無限の長さを持つデータ 外観に適した形式で表現される。

* 声明

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。

構造化プログラミング

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ブランチは行われない。

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。

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。

サブプログラム (* subprogram)

呼び出し先プログラム (*called program*) を参照。

添え字 (* subscript)

整数、(オブションで演算子 + または - 付きの整数が後ろにある) データ名、あるいは(オブションで演算子 + 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。

置換文字

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。

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。

サロゲート・ペア

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・コード化文字セットに適合する文字を 65,536 文字を超えて提供する。

スイッチ状況条件

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。

記号文字 (* 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。

構文

(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。

* システム名

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。

T

* テーブル

OCCURS 節を使用して DATA DIVISION で定義される、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繰り返し項目の集合に属するデータ項目。

テキスト・デック

オブジェクト・デック (*object deck*) またはオブジェクト・モジュール (*object module*) と同義。

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定義語。

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。
- ・リテラル。英数字リテラルの場合には、そのリテラルの境界となる開始の引用符と終了の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。

スレッド

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)。

トークン

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。

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。

トレーラー・ラベル

(1) 記録メディア・ユニットのデータ・レコードの後にある、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。

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。

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。

型付きオブジェクト参照

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。

U

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。

無制限テーブル (unbounded table)

上限として *integer-2* を指定する代わりに、OCCURS *integer-1* to UNBOUNDED を持つテーブル。

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。

Enterprise COBOL は、国別データ型の表現としてビッグエンディアン形式のを使用するユニコードをサポートしています。UTF-16

不変リソース ID (URI)

リソースを一意に指す文字のシーケンスのことで、Enterprise COBOL では、名前空間の ID。URI の構文は、[Uniform Resource Identifier \(URI\): Generic Syntax](#) の資料に定義されている。

ユニット

直接アクセスのモジュールであり、その大きさは IBM によって決められている。

汎用オブジェクト・リファレンス

どのクラスのオブジェクトでも参照できるデータ名。

無制限ストレージ

AMODE 31 では、無制限ストレージは、2 GB 境界より下にあり、16 MB 境界より上または下に配置可能。

AMODE 64 では、無制限ストレージには、プログラムで使用可能なすべてのストレージ (2 GB 境界より上と下の両方) が含まれる。

不成功の実行 (* 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。

UPSI スイッチ

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

URI を参照。

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。

V

* 変数

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。

可変長項目

OCCURS 節の DEPENDING 句で記述されたテーブル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。

* かへんはっせいデータこうもく

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、そのデータ記述項目内に OCCURS DEPENDING ON 節を含んでいるか、そのような項目に従属している必要がある。

* 可変位置グループ

同じレコード内の可変長テーブルに続くグループ項目 (可変長テーブルに従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

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)。

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。

ボリューム

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。

ボリューム切り替え処理手順 (volume switch procedures)

ファイルの終わりに達する前にユニットまたはリールの終わりに達したとき、自動的に実行されるシステム固有の処理手順。

VSAM ファイルシステム

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

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。

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。

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。

* WORKING-STORAGE SECTION

WORKING-STORAGE データ項目を記述する DATA DIVISION のセクション。独立項目または WORKING-STORAGE レコード、あるいはその両方で構成される。

ワークステーション

コンピュータの総称 (パーソナル・コンピュータ、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。

ラッパー

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。

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・システム間での構造化情報の伝送および共有が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 データ

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 Element ・タイプ宣言で定義される。

XML 宣言

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 文書

W3C XML 規格で定義されているとおり正しい形式のデータ・オブジェクト。

XML ネーム・スペース

W3C XML ネーム・スペース仕様によって定義されたメカニズムで、Element 名および属性名の集まりの有効範囲を制限する。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 スキーマ

W3C によって定義されたメカニズムで、XML 文書の構造と内容を記述し、制約する。XML スキーマは、それ自体が XML で表され、特定タイプ (購入注文など) の XML 文書のクラスを効率的に定義する。

Y

Z

z/OS UNIX ファイルシステム

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 ユニックス を使用してアクセスできる。

ゾーン化された 10 進データ項目

暗黙的または明示的に USAGE DISPLAY として記述され、PICTURE の記号 9、S、P、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 進数データ品目の中身は、0 から 9 の文字で表され、オプションで符号付きで表されます。PICTURE スtring が符号を指定していて、SIGN IS SEPARATE 節が指定されている場合、符号は文字 + または - として表される。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 進数字である (先行または末尾)。

#

85 COBOL 標準

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*」および「ANSI INCITS 23b-1993, *Programming Languages - Correction Amendment 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*」および「ISO/IEC 1989/AMD2:1994, *Programming languages - Correction and clarification amendment for COBOL*」に改訂されました。

2002 COBOL 標準

以下の標準によって定義された COBOL 言語。

- INCITS/ISO/IEC 1989-2002, *Information technology - Programming languages - COBOL*

2014 COBOL 標準

以下の標準によって定義された COBOL 言語。

- INCITS/ISO/IEC 1989:2014, *Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

リソース・リスト

Enterprise COBOL for z/OS の資料

COBOL for z/OS の資料

以下の資料が「[Enterprise COBOL for z/OS ライブラリ](#)」にあります。

- 新機能, SC31-5708-00
- カスタマイズ・ガイド, SC27-8712-03
- 言語リファレンス, SC27-8713-03
- プログラミング・ガイド, SC27-8714-03
- マイグレーション・ガイド, GC27-8715-03
- パフォーマンス・チューニング・ガイド, SC27-9202-02
- メッセージとコード, SC27-4648-02
- プログラム・ディレクトリー, GI13-4526-03
- ライセンス・プログラム仕様, GI13-4532-03

ソフトコピー資料

次のコレクション・キットには、Enterprise COBOL およびその他の製品資料が含まれます。それらは <https://www.ibm.com/resources/publications> にあります。

- *z/OS Software Products Collection*
- *z/OS and Software Products DVD Collection*

サポート

Enterprise COBOL for z/OS のご使用の際に問題がある場合は、サイト: <https://www.ibm.com/support/pages/node/6560933> 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

z/OS ライブラリー資料

以下の資料が「[z/OS 図書館](#)」にあります。

ランタイム・ライブラリー拡張機能

- 一般的なデバッグ方式 ライブラリー参照
- 一般的なデバッグ方式 ユーザーズ・ガイド
- DWARF/ELF 拡張ライブラリー参照

z/Architecture

- *z/Architecture* 解説書

z/OS DFSMS

- カタログのためのアクセス方式サービス・プログラム
- *Checkpoint/Restart*
- *Macro Instructions for Data Sets*
- データ・セットの使用法

- *Utilities*

z/OS DFSORT

- アプリケーション・プログラミング・ガイド
- インストールおよびカスタマイズ

z/OS ISPF

- ダイアログ開発者 ガイドとリファレンス
- ユーザーズ・ガイド 第1巻
- ユーザーズ・ガイド 第2巻

z/OS 言語環境プログラム

- 概念
- カスタマイズ
- デバッグのガイド
- *Language Environment Vendor Interfaces*
- プログラミング・ガイド
- プログラミング・リファレンス
- ランタイム・メッセージ
- ランタイム マイグレーション・ガイド
- ILC (言語間通信) アプリケーションの作成

z/OS MVS

- JCL 解説書
- JCL ユーザーズ・ガイド
- プログラミング: 高水準言語向け呼び出し可能サービス
- プログラム管理: ユーザーズ・ガイドおよび解説書
- システム・コマンド
- *z/OS Unicode Services* ユーザーズ・ガイドおよび解説書
- *z/OS XML System Services* ユーザーズ・ガイドおよび解説書

z/OS TSO/E

- コマンド解説書
- 入門
- ユーザーズ・ガイド

z/OS UNIX システム・サービス

- コマンド解説書
- プログラミング: アセンブラー呼び出し可能サービス 解説書
- ユーザーズ・ガイド

z/OS XL C/C++

- プログラミング・ガイド
- ランタイム・ライブラリー・リファレンス

CICS Transaction Server for z/OS

以下の資料が「[CICS ライブラリー](#)」にあります。

- CICS アプリケーションの開発

- *API (EXEC CICS) リファレンス*
- *CICS システム・プログラムの開発*
- *グローバル・ユーザー出口リファレンス*
- *XPI Reference*
- *CICS での EXCI の使用*

COBOL 報告書作成プログラム・プリコンパイラー

- *Programmer's Manual*, SC26-4301
- *Installation and Operation*, SC26-4302

Db2 for z/OS

以下の資料が「[Db2 図書館](#)」にあります。

- *アプリケーション・プログラミングおよび SQL ガイド*
- *コマンド解説書*
- *SQL 解説書*

IBM z/OS デバッガ (以前の IBM z システムズ およびデバッグ・ツールのデバッグ)

IBM z/OS デバッガ については、[IBM z/OS デバッガ 図書館](#)を参照してください。

IBM 開発者 z/OS (以前の IBM Developer for z Systems)

IBM Developer for z/OS に関する情報は、[IBM 開発者 z/OS 図書館](#)にあります。

注 : IBM Developer for z Systems および Rational Developer for z Systems は IBM 開発者 z/OS に置き換えられています。

以下の資料が「[IBM Publications Center](#)」にあり、資料番号で検索できます。

IMS

- *Application Programming API Reference*, SC18-9699
- *Application Programming Guide*, SC18-9698

WebSphere Application Server for z/OS

- *Applications*, SA22-7959

Softcopy publications for z/OS

以下のコレクション・キットには、z/OS および関連製品資料が含まれます。

- *z/OS CD Collection Kit*, SK3T-4269

JAVA

- *IBM ジャワ - ツール資料用の SDK*, publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp
- *The Java 2 Enterprise Edition Developer's Guide*, download.oracle.com/javaee/1.2.1/devguide/html/DevGuideTOC.html
- *Java 2 on z/OS*, www.ibm.com/servers/eserver/zseries/software/java/
- *The Java EE 5 Tutorial*, download.oracle.com/javaee/5/tutorial/doc/
- *Java Language Specification, Third Edition* (Gosling ほか), java.sun.com/docs/books/jls/
- *The Java Native Interface*, download.oracle.com/javase/1.5.0/docs/guide/jni/

- *JDK 5.0 Documentation*, download.oracle.com/javase/1.5.0/docs/

JSON

- JavaScript Object Notation (JSON), www.json.org

Unicode および文字表現

- *Unicode*, www.unicode.org/
- *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

Enterprise COBOL for z/OS の [1026](#)

z/OS の使用 [1026](#)

アクセス方式サービス

代替索引の事前作成 [225](#)

VSAM データ・セットのロード [213](#)

z/OS に対する VSAM データ・セットの定義 [219](#)

アセンブラ

PROCEDURE DIVISION の拡張 [512](#)

programs

からのコンパイル [289](#)

からの呼び出し (CICS での) [547](#)

マルチスレッド化 [657](#)

リスト [414](#), [843](#)

値の割り当て

アドレス

入り口点アドレスを渡す [617](#)

増分 [630](#)

プログラム間での受け渡し [629](#)

NULL 値 [630](#)

アドレスを増分する [630](#)

アドレッシング・モード、定義 [40](#)

アプリケーションの移植

分離符号の影響 [50](#)

暗黙の範囲終了符号 [21](#)

イグズオプト

ランタイム・オプション

JCL サンプル [476](#)

移行の考慮事項

Java および COBOL [330](#)

異常終了、コンパイル時 [398](#)

一意の COBOL ランタイム・メッセージ (IGZUQMSG) [471](#)

一括表示表現 [838](#)

移動、制御権の

ネストされたプログラム [614](#)

メインプログラムとサブプログラム [603](#)

呼び出し側プログラム [603](#)

呼び出し先プログラム [603](#)

COBOL プログラム相互間の [605](#), [613](#)

COBOL プログラムと非 COBOL プログラムの間での [603](#)

入り口点

入り口アドレスの受け渡し [617](#)

代替 [620](#)

代替、ENTRY ステートメントの [617](#)

プロシーチャー・ポインター・データ項目 [617](#)

ENTRY ラベル [620](#)

印刷されるページのサイズ、制御 [189](#)

インスタンス

削除 [769](#)

作成 [767](#)

定義 [743](#)

インスタンス・データ

初期化 [767](#)

インスタンス・データ (続き)

それをアクセス可能にする [756](#)

定義 [743](#), [750](#), [772](#)

private [750](#)

インスタンス・メソッド

多重定義 [755](#)

定義 [743](#), [751](#), [772](#)

呼び出し、オーバーライドした [766](#)

overriding [755](#)

インスタンス・メソッドの多重定義 [755](#)

インライン PERFORM

概要 [107](#)

例 [108](#)

インライン・コメント [1053](#)

受け渡し、プログラム間でのデータの

アドレス [629](#)

オプションの考慮事項 [42](#)

呼び出し側プログラムの引数 [625](#)

呼び出し先プログラムのパラメーター [626](#)

BY CONTENT [623](#)

BY REFERENCE [623](#)

BY VALUE

概要 [623](#)

制約事項 [625](#)

EXTERNAL データ [633](#)

Java との [789](#)

JNI サービス [786](#)

OMITTED 引数 [627](#)

RETURN-CODE 特殊レジスター [632](#)

埋められていないトラック [186](#)

英字データ

国別との比較 [150](#)

これを伴う MOVE ステートメント [32](#)

UTF-8 との比較 [161](#)

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 [32](#)

比較する

国別と [150](#)

UTF-8 へ [161](#)

ZWB の影響 [468](#)

変換

IGZCA2D による DBCS への [868](#)

MOVE による UTF-8 への [159](#)

MOVE による国別への [142](#)

NATIONAL-OF による国別への [143](#)

2 バイト文字を含む [867](#)

英数字比較 [103](#)

英数字編集データ

これを伴う MOVE ステートメント [32](#)

初期化

例 [29](#)

INITIALIZE の使用 [80](#)

英数字リテラル

混合 DBCS/EBCDIC の変換 [867](#)

英数字リテラル (続き)

- [説明 25](#)
- [2 バイト文字を含む 867](#)
- [DBCS の内容での 169](#)

エラー

- [コンパイラー・オプションの競合 376](#)
- [算術演算 266](#)
- [処理 265](#)
- [処理のルーチン 276](#)
- [入出力の処理 176](#)
- [プロセッシング](#)
 - [行順次ファイル 241](#)
 - [QSAM ファイル 190](#)
 - [VSAM ファイル 217](#)
 - [XML GENERATE 732](#)
 - [XML PARSE 711](#)
- [メッセージ・テーブル](#)
 - [指標付けを使用した例 83](#)
 - [添え字付けを使用した例 82](#)
- [リスト 301](#)

エラー・メッセージ

- [コンパイラー](#)
 - [カスタマイズ 895](#)
 - [形式 308](#)
 - [作成する重大度レベルの判別 402](#)
 - [重大度レベル 308, 896](#)
 - [ソースの訂正 306](#)
 - [ソース・リストへの組み込み 492](#)
 - [端末への送信 296](#)
 - [出口モジュールからの 903](#)
 - [フラグを立てる重大度の選択 492](#)
 - [リストにおける位置 308](#)
 - [リストの生成 307](#)
- [コンパイラーの指示 307](#)

エンクレーブ 603

エンコード

- [英数字 XML 文書の場合の指定 707](#)
- [言語文字 132](#)
- [構文解析対象の XML 文書 690](#)
- [生成された XML 出力での制御 732](#)
- [説明 148](#)
- [CODEPAGE オプションでの指定 385](#)
- [XML 文書での矛盾 712](#)
- [XML 文書の 704, 705](#)

エンコード宣言

- [指定 707](#)
- [省略を推奨 707](#)

オーバーフロー条件

- [ストリングの結合および分割 266](#)
- [CALL 275](#)
- [UNSTRING 113](#)

大文字への変換 123

オブジェクト

- [削除 769](#)
- [作成 767](#)
- [定義 743](#)

オブジェクト・インスタンス、定義 743

オブジェクト・インスタンスの解放 769

オブジェクト・コード

- [コンパイルおよびリスト 301](#)
- [作成 297](#)
- [生成 388](#)
- [80 桁レコードでの作成 393](#)

オブジェクト参照

オブジェクト参照 (続き)

- [型式化 761](#)
- [設定 762](#)
- [汎用 761](#)

比較する 762

引き渡しの例 765

ローカルからグローバルへの変換 767

オブジェクト指向 (OO) COBOL フレームワークの外部での

COBOL/Java 相互運用性

AMODE の考慮事項 819

COBOL/Java 相互運用性

オブジェクト指向 COBOL の外部 815

Java 相互運用可能 COBOL アプリケーションでのエラー処理 817

オブジェクト指向 COBOL

アプリケーションの準備

JCL または TSO/E の使用 327

z/OS ユニックスのもとでの 322

オブジェクト指向プログラムの書き込み 743

互換 330

コンパイル

JCL または TSO/E の使用 326

z/OS ユニックスのもとでの 321

実行

JCL または TSO/E の使用 327

XPLINK リンケージ 329

z/OS ユニックスのもとでの 324

制約事項

ソートとマージ 243

CICS 743

CICS のもとでは実行できない 545

EXEC CICS ステートメント 743

EXEC SQL ステートメント 743

SQL コンパイラー・オプション 743

SQL ステートメント 558

バインディング(binding)

概要 327

例 328

プログラムは再入可能でなければならない 620

リンク

概要 322

例 323

DLL 648

IMS

データベースへのアクセス 576

COBOL からの Java メソッドの呼び出し 575

Java からの COBOL メソッドの呼び出し 574

Java との通信 790

OO プログラムとの間の呼び出し 617

オブジェクト指向 COBOL の外部での COBOL/Java 相互相互運用性

グループ・パラメーター 813

バイト配列 813

配列セマンティクス 814

配列の制限 814

COBOL での Java 互換配列型の使用 813

オプション

ランタイム 468

オプション・ファイル

QSAM 188

VSAM 212

[カ行]

ガーベッジ・コレクション [769](#)

開発

AMODE 64 プログラム [587](#)

外部 10 進数データ

ゾーン [53](#)

national [53](#)

外部クラス名 [749](#), [760](#)

外部シンボル [530](#)

外部データ

共用 [633](#)

ストレージの獲得 [43](#)

保管場所 [43](#)

外部ファイル [634](#)

外部浮動小数点データ

表示 [53](#)

national [54](#)

カウント

生成される XML 文字 [728](#)

文字 (INSPECT) [120](#)

書き込み、レコードのブロックの [184](#)

拡張モード [49](#), [859](#)

カスタマー・サポート xxxiii, [1077](#)

型式化オブジェクト参照 [761](#)

カタログ式プロシージャ

コンパイル (IGYWC) [281](#)

コンパイル、リンク・エディット、実行 (IGYWCLG) [283](#)

コンパイルおよびリンク・エディット (IGYWCL) [282](#)

コンパイル用 JCL [280](#)

各国語サポート (NLS)

データ処理 [131](#)

DBCS [168](#)

LANGUAGE コンパイラー・オプション [413](#)

可変位置グループ [88](#)

可変位置データ項目 [88](#)

可変長テーブル

値の割り当て [87](#)

オーバーレイを防ぐ [90](#)

作成 [84](#)

例 [85](#)

ロードの例 [86](#)

可変長レコード

ソート [250](#)

OCCURS DEPENDING ON (ODO) 節 [841](#)

QSAM

要求 [180](#)

レイアウト [181](#)

VSAM

定義 [208](#)

RRDS [202](#)

環境変数

行順次ファイルの割り振り [238](#)

コピーブック [481](#)

コンパイラー [311](#)

設定およびアクセス [580](#)

設定およびアクセスの例 [582](#)

テキスト名 [312](#), [481](#)

ファイルの定義、例 [9](#)

ファイルを割り振るための使用 [175](#)

ライブラリー名 [312](#), [481](#)

ランタイム [581](#)

CLASSPATH

設定の例 [327](#)

環境変数 (続き)

CLASSPATH (続き)

説明 [581](#)

Java クラスの場所の指定 [324](#)

COBJVMINITOPTIONS

説明 [581](#)

JVM オプションの指定 [325](#)

COBOL_INSTALL_DIR [312](#)

COBOPT [311](#)

LIBPATH

設定の例 [327](#)

説明 [581](#)

COBOL クラスの場所の指定 [324](#)

PATH

設定の例 [327](#)

説明 [581](#)

QSAM ファイルの定義 [191](#)

STEPLIB

説明 [581](#)

例 [314](#)

SYSLIB

説明 [312](#)

JNI.cpy の場所の指定 [322](#)

_BPX_SHAREAS [583](#)

_CEE_ENVFILE

説明 [581](#)

Java 設定値を示す [327](#)

_CEE_RUNOPTS

説明 [581](#)

ランタイム・オプションの指定 [579](#)

XPLINK の設定 [329](#)

_IGZ_SYSOUT [581](#)

環境名 [5](#)

漢字データのテスト [169](#)

漢字比較 [104](#)

関数プロトタイプ

外部名 [665](#)

言語標準の考慮 [667](#)

使用 [665](#)

ベスト・プラクティス [666](#)

関数プロトタイプの使用 [665](#)

関数ポインター・データ項目

言語環境プログラム・サービスの呼び出し [619](#)

定義 [617](#)

呼び出し DLL プログラム

例 [646](#)

CALL ステートメント [619](#)

COBOL の呼び出し [619](#)

DLL での [645](#)

JNI サービスのアドレッシング [906](#)

SET 関数ポインター [617](#)

簡略名

SPECIAL-NAMES 段落 [6](#)

完了コード

ソート [254](#)

マージ [254](#)

関連データ・ファイルの作成 [297](#)

キー

基本、KSDS ファイルの [204](#)

許容データ型

MERGE ステートメントでの [252](#)

OCCURS 節の [74](#)

SORT ステートメントでの [252](#)

相対レコード [206](#)

キー (続き)
 ソート用の
 概要 [244](#)
 定義 [252](#)
 代替、KSDS ファイルの [205](#)
 テーブル・エレメントの順序を指定するための [74](#)
 二分探索用の [93](#)
 マージ用の
 概要 [244](#)
 定義 [252](#)
 キーボード・ナビゲーション [1026](#)
 キーワード [1056](#)
 記述、コンピューター [5](#)
 基本クラスター名 [221](#)
 基本ライブラリー [296](#)
 逆順、テープ・ファイルの [187](#)
 行、テーブルの [74](#)
 行順次ファイル
 オープン [240](#)
 書き込み [239](#)
 国別データはサポートされない [241](#)
 クローズ [241](#)
 再オープン防止のためのクローズ [240](#)
 制御文字 [238](#)
 入出力エラー処理 [241](#)
 入出力ステートメント [239](#)
 プロセッシング [237](#)
 ブロック化 [12](#)
 編成 [237](#)
 読み取り [239](#)
 レコードの追加 [241](#)
 レコードの読み取り [240](#)
 DATA DIVISION 記入項目 [238](#)
 ENVIRONMENT DIVISION 記入項目 [237](#)
 z/OS のもとでの
 作成 [238](#)
 ジョブ制御言語 (JCL) (job control language (JCL))
 [238](#)
 割り振り [238](#)
 行番号 [505](#), [537](#)
 共用
 データ
 概要 [623](#)
 再帰的またはマルチスレッド化されたプログラムの
 [17](#)
 パラメーター受け渡しの仕組み [623](#)
 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 [633](#)
 別々にコンパイルされたプログラムの [17](#)
 メソッドから値を戻す [766](#)
 メソッドへの引数の引き渡し [764](#)
 有効範囲、名前の [616](#)
 Java との [789](#)
 LINKAGE SECTION のコーディング [627](#)
 PROCEDURE DIVISION ヘッダー [628](#)
 RETURN-CODE 特殊レジスター [632](#)
 ファイル
 有効範囲、名前の [616](#)
 EXTERNAL 節の使用 [13](#), [634](#)
 GLOBAL 節の使用 [13](#)
 句、定義 [19](#)
 国別 10 進数データ (USAGE NATIONAL)
 形式 [53](#)
 初期化の例 [30](#)
 国別 10 進数データ (USAGE NATIONAL) (続き)
 定義 [141](#)
 例 [49](#)
 国別グループ項目
 英数字グループと比べた場合の利点 [141](#)
 英数字リテラルを伴う VALUE 節の例 [82](#)
 概要 [141](#)
 基本項目として扱われる
 ほとんどの場合 [24](#), [141](#)
 MOVE の例 [33](#)
 国別データのみを含むことができる [24](#), [146](#)
 グループ項目として扱われる
 サマリー [147](#)
 INITIALIZE で [32](#)
 INITIALIZE の例 [147](#)
 MOVE CORRESPONDING で [33](#)
 これを伴う MOVE ステートメント [33](#)
 使用
 概要 [145](#)
 基本項目として [146](#)
 初期化
 INITIALIZE の使用 [31](#), [80](#)
 VALUE 節の使用 [82](#)
 生成された XML 文書の [728](#)
 定義 [145](#)
 テーブルの定義 [74](#)
 引数として渡す [628](#)
 例 [24](#)
 BYTE-LENGTH 組み込み関数および [129](#)
 Java との通信 [790](#)
 LENGTH 組み込み関数および [129](#)
 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 [128](#)
 外部 10 進数 [53](#)
 外部浮動小数点 [54](#)
 キーの
 MERGE ステートメントでの [252](#)
 OCCURS 節の [74](#)
 SORT ステートメントでの [252](#)
 組み込み関数を使用した評価 [126](#)
 形象定数 [140](#)
 検査 (INSPECT) [120](#)
 これを伴う MOVE ステートメント [32](#), [142](#)
 最小項目または最大項目の検出 [127](#)
 指定 [133](#)
 出力での表示 [36](#)
 条件式の [148](#), [149](#)
 初期化の例 [29](#)
 生成された XML 文書の [728](#)
 その参照変更 [118](#)
 定義 [134](#)
 比較する
 英字、英数字、または DBCS と [150](#)
 英数字グループと [150](#)
 概要 [148](#)
 数値との [149](#)
 2 つのオペランド [149](#)
 分割 (UNSTRING) [114](#)
 変換
 大文字または小文字への [123](#)
 概要 [142](#)
 ギリシャ語の英数字との間、例 [144](#)
 中国語 GB 18030 との間の [167](#)

国別データ (続き)

変換 (続き)

例外 [144](#)

DISPLAY-OF による UTF-8 への [160](#)

DISPLAY-OF による英数字への [143](#)

INSPECT による [120](#)

MOVE による英数字、DBCS、または整数からの
[142](#)

NATIONAL-OF による UTF-8 からの [160](#)

NATIONAL-OF による英数字または DBCS からの
[143](#)

NUMVAL、NUMVAL-C による数値への [123](#)

UTF-8 との間の [162](#)

文字の逆順 [123](#)

リテラル

使用 [134](#)

連結 (STRING) [111](#)

ACCEPT による入力 [35](#)

BYTE-LENGTH 組み込み関数および [129](#)

Java との通信 [790](#)

LENGTH OF 特殊レジスター [129](#)

LENGTH 組み込み関数および [129](#)

USAGE 節がない場合の NSYMBOL コンパイラー・オプション [134](#)

XML 文書のエンコード [705](#)

国別比較 [104](#)

国別浮動小数点データ (USAGE NATIONAL)

定義 [54](#), [141](#)

国別編集データ

これを伴う MOVE ステートメント [32](#)

初期化

例 [29](#)

INITIALIZE の使用 [80](#)

定義 [134](#)

編集記号 [134](#)

PICTURE 節 [134](#)

国別リテラル

使用 [134](#)

説明 [26](#)

組み込みエラー・メッセージ [493](#)

組み込み関数

英数字データ項目の変換 [122](#)

国別データ項目の変換 [122](#)

コンパイルの日付の検索 [130](#)

最大または最小項目の検出 [127](#)

参照修飾子としての [120](#)

紹介 [38](#)

数字関数

言語環境プログラム 呼び出し可能サービスとの違い
[65](#)

整数、浮動小数点、混合 [63](#)

同等の言語環境プログラム 呼び出し可能サービス
[64](#)

ネストされた [64](#)

引数としてのテーブル・エレメント [64](#)

引数としての特殊レジスター [64](#)

用途 [63](#)

例 [63](#)

中間結果 [864](#), [866](#)

データ項目の長さの検出 [129](#)

データ項目の評価 [126](#)

テーブル・エレメントの処理 [94](#)

ネスト [39](#)

例

組み込み関数 (続き)

例 (続き)

最小 [120](#)

整数 [120](#)

長さ [67](#), [128](#), [129](#)

ログ [68](#)

ANNUITY [68](#)

BIT-OF [125](#)

BIT-TO-CHAR [126](#)

CHAR [127](#)

CURRENT-DATE [67](#)

DISPLAY-OF [144](#)

HEX-OF [125](#)

HEX-TO-CHAR [126](#)

INTEGER-OF-DATE [67](#)

LOWER-CASE [123](#)

MAX [67](#), [95](#), [127](#), [128](#)

MEAN [68](#)

MEDIAN [68](#), [95](#)

NATIONAL-OF [144](#)

NUMVAL [123](#)

NUMVAL-C [67](#), [123](#)

ORD [127](#)

ORD-MAX [95](#), [127](#)

PRESENT-VALUE [67](#)

RANGE [68](#), [95](#)

REM [68](#)

REVERSE [123](#)

SQRT [68](#)

SUM [95](#)

UPPER-CASE [123](#)

WHEN-COMPILED [130](#)

Unicode 関数の例 [164](#)

UTF-8 [163](#)

組み込み相互参照

説明 [500](#)

例 [535](#)

組み込みの CICS 変換プログラム

概要 [551](#)

コンパイラー・オプション [549](#)

利点 [551](#)

組み込みマップ要約 [495](#), [508](#)

クライアント

定義 [759](#)

クラス

インスタンス化

COBOL の [768](#)

Java [768](#)

インスタンス・データ [750](#)

オブジェクト、JNI による参照の取得 [786](#)

定義 [743](#), [746](#)

名前

外部 [749](#), [760](#)

プログラムにおいて [748](#)

ファクトリー・データ [774](#)

ユーザー定義 [8](#)

クラス条件

データの妥当性検査 [487](#)

テスト

概要 [103](#)

漢字の [169](#)

数値の [60](#)

DBCS の [169](#)

クラスター、VSAM [219](#)

グループ移動と基本移動の対比 [33](#), [146](#)

グループ項目

可変位置 [88](#)

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 [146](#)

国別データと比較 [150](#)

グループ移動と基本移動の対比 [33](#), [146](#)

グループ項目として扱われる

INITIALIZE で [32](#)

INITIALIZE の例 [80](#)

これを伴う MOVE ステートメント [33](#)

初期化

INITIALIZE の使用 [31](#), [79](#)

VALUE 節の使用 [82](#)

定義 [24](#)

テーブルの定義 [73](#)

引数として渡す [628](#)

UTF-8 データとの比較 [161](#)

グローバル名 [616](#)

計算

算術データ項目 [839](#)

指標の [78](#)

添え字の [842](#)

形式 V VSAM ファイルの READ INTO [209](#)

形式、レコードの

可変長

QSAM のレイアウト [181](#)

QSAM 用の要求 [180](#)

VSAM 用定義 [208](#)

形式 D

要求 [180](#)

レイアウト [181](#)

形式 F

要求 [179](#)

レイアウト [179](#)

形式 S

概要 [182](#)

要求 [181](#)

レイアウト [183](#)

形式 U

要求 [183](#)

レイアウト [184](#)

形式 V

要求 [180](#)

レイアウト [181](#)

固定長

QSAM のレイアウト [179](#)

QSAM 用の要求 [179](#)

VSAM 用定義 [208](#)

スパン

概要 [182](#)

要求 [181](#)

レイアウト [183](#)

未定義

要求 [183](#)

レイアウト [184](#)

QSAM ASCII テープの場合 [199](#)

形象定数

国別文字 [140](#)

定義 [26](#)

HIGH-VALUE の制約事項 [140](#), [156](#)

UTF-8-character [156](#)

継承の階層の定義 [745](#)

継続

継続 (続き)

エントリー [260](#)

構文検査 [388](#)

プログラム [267](#)

ケース構造、EVALUATE ステートメント [101](#)

言語環境プログラム 呼び出し可能サービス

概要 [853](#)

各国語のサポート [854](#)

組み込み関数との違い [65](#)

サンプル・リスト [855](#)

種類 [853](#)

条件処理 [854](#)

省略されたフィードバック・コード [856](#)

使用例 [856](#)

数学 [854](#)

数学用 [64](#)

対応する数学組み込み関数 [64](#)

動的ストレージ・サービス [854](#)

日時の計算 [854](#)

日時用 [66](#)

フィードバック・コード [855](#)

メッセージ処理 [854](#)

戻りコード [856](#)

CALL での呼び出し [855](#)

RETURN-CODE 特殊レジスター [856](#)

言語間通信

サブプログラム [603](#)

マルチスレッド化 [657](#)

CICS のもとで [548](#)

COBOL および Java 間の [785](#)

IMS アプリケーション [575](#)

PL/I タスク [657](#)

検査、有効データの

条件式 [103](#)

更新、VSAM レコードの [215](#)

構造、INITIALIZE による初期化 [31](#)

構造化プログラミング [838](#)

構文エラー

NOCOMPILE コンパイラー・オプションによる検出 [490](#)

構文解析データ項目、定義 [690](#)

構文図の読み方 [xxxi](#)

効率、コーディングの [837](#)

考慮事項

AMODE 64 [590](#), [593-596](#)

考慮事項と制約事項

AMODE 31/64 の場合 [598](#)

AMODE COBOL/JAVA の場合 [596](#)

コーディング

インスタンス・メソッド [751](#), [772](#)

エラー、回避 [837](#)

オブジェクト指向プログラム

概要 [743](#)

再入可能でなければならない [620](#)

技法 [12](#), [837](#)

クライアント [759](#)

クラス定義 [746](#)

決定 [99](#)

効率的 [837](#)

コンストラクター・メソッド [775](#)

サブクラス

概要 [770](#)

例 [773](#)

条件テスト [104](#)

単純化 [851](#)

コーディング (続き)

- テーブル [73](#)
- テスト条件 [104](#)
- 入出力ステートメント
 - 行順次ファイル用 [239](#)
 - QSAM ファイル用 [186](#)
 - VSAM ファイル用 [209](#)
- 入出力の概要 [173](#)
- ファイル入出力の概要 [171](#)
- ファクトリー定義 [774](#)
- ファクトリー・メソッド [775](#)
- ループ [107](#)
- CICS のもとで実行されるプログラム
 - 概要 [545](#)
 - 再入可能でなければならない [620](#)
 - システム日付の取得 [547](#)
 - 制約事項 [545](#)
 - 入出力 [546](#)
 - 呼び出し [547](#)
 - DISPLAY ステートメント [546](#)
 - SORT ステートメント [553](#)
- DATA DIVISION [12](#)
- Db2 のもとで実行されるプログラム
 - 概要 [557](#)
 - ストアド・プロシージャは再入可能でなければならない [620](#)
 - ストリング・データの CCSID [563](#)
- ENVIRONMENT DIVISION [5](#)
- EVALUATE ステートメント [101](#)
- IDENTIFICATION DIVISION [3](#)
- IF ステートメント [99](#)
- IMS のもとで実行されるプログラム
 - 概要 [569](#)
 - 再入可能でなければならない [620](#)
 - 制約事項 [569](#)
- Java との相互運用が可能なデータ型 [790](#)
- PROCEDURE DIVISION [18](#)
- SQL ステートメント
 - 概要 [558](#)
 - 制約事項 [558](#)
- SQLIMS ステートメント
 - 概要 [570](#)

コード

- コピー [852](#)
- 最適化 [843](#)
- コード化文字セット
 - 定義 [132](#)
 - XML 文書における [704](#)

コード・ページ

- 英数字 XML 文書の場合の指定 [707](#)
- 指定 [385](#)
- 定義 [132](#)
- 特殊文字の 16 進値 [708](#)
- ユーロ通貨サポート [71](#)
- Db2 ストリング・データ [563](#)
- DBCS [387](#)
- overriding [144](#)
- XML 文書での矛盾 [712](#)

コード・ポイント、定義 [132](#)

互換

- オブジェクト指向構文 [330](#)
- Java および COBOL [330](#)

互換モード [49, 859](#)

固定小数点演算

固定小数点演算 (続き)

- 指数 [862](#)
- 比較 [69](#)
- 評価 [68](#)
- 例の評価 [70](#)
- 固定小数点データ
 - 外部 10 進数 [53](#)
 - 固定小数点と浮動小数点との間の変換 [58](#)
 - 使用計画 [839](#)
 - 中間結果 [861](#)
 - バイナリー [54](#)
 - パック 10 進数 [55](#)
 - 変換と精度 [58](#)

固定情報

- 形象、定義 [26](#)
- 定義 [26](#)
- データ項目 [838](#)

固定長レコード

- QSAM
 - 要求 [179](#)
 - レイアウト [179](#)
- VSAM
 - 定義 [208](#)
 - RRDS [202](#)

コピーブック

- 使用 [852](#)
- 説明 [481](#)
- 相互参照 [533](#)
- 探索 [317, 482](#)
- ユーザー提供モジュールからの入手 [400](#)

コピーブック相互参照、記述 [494](#)

コピー・ライブラリー

- 指定 [296](#)
- 探索順序 [482](#)
- データ・セット [291](#)
- 例 [853](#)
- COPY ステートメント [481](#)
- SYSLIB [296](#)
- z/OS ユニックス 探索順序 [312, 317](#)

コプロセッサ、Db2

- 概要 [557](#)
- 推奨コンパイラ・オプション SQLCCSID [564](#)
- ストリング・データの CCSID の決定 [563](#)
- プリコンパイラとの相違 [565](#)
- SQL INCLUDE の使用 [559](#)
- SQL コンパイラ・オプションで使用可能にする [561](#)

コプロセッサ、IMS

- 概要 [569](#)
- SQLIMS コンパイラ・オプションで使用可能にする [571](#)

コメント

- 送付 xxxiii
- コメント行 [1039](#)
- 小文字への変換 [123](#)
- 混合 AMODE [593](#)

混合 DBCS/EBCDIC リテラル

- 英数字から DBCS への変換 [868](#)
- DBCS から英数字への変換 [870](#)

コントロール

- 移動 [603](#)
- ネストされたプログラム内の [614](#)
- プログラムのフロー [99](#)

コンパイラ

- エラー・メッセージのリストの生成 [307](#)

コンパイラー (続き)

制限

DATA DIVISION [12](#)

中間結果の計算 [860](#)

メッセージ

カスタマイズ [895](#)

作成する重大度レベルの判別 [402](#)

重大度レベル [308](#), [896](#)

ソース・リストへの組み込み [492](#)

端末への送信 [296](#)

出口モジュールからの [903](#)

フラグを立てる重大度の選択 [492](#)

戻りコード

概要 [308](#)

メッセージ・カスタマイズの影響 [897](#)

最も高い重大度に依存 [309](#)

z/OS ユニックス シェルでの呼び出し

概要 [313](#)

例 [315](#)

z/OS ユニックス のもとでの環境変数 [311](#)

コンパイラー・オプション

外部参照 [463](#), [494](#)

組み込みの CICS 変換プログラム用 [549](#)

コンディション [388](#)

コンパイラー呼び出しでの [503](#)

サービス [440](#)

シグニチャー情報バイト [513](#)

指定

PROCESS (CBL) の使用 [300](#)

状況 [504](#)

省略形 [371](#)

その表 [371](#)

データ [391](#)

出口 [399](#)

デバッグ用

概要 [490](#)

TEST に関する制約事項 [488](#)

THREAD に関する制約事項 [488](#)

名前 [421](#)

パフォーマンスの考慮事項 [845](#)

番号 [422](#), [501](#)

分離型の CICS 変換プログラム用 [549](#), [552](#)

矛盾する [376](#)

有効な [513](#)

優先順位

バッチで [304](#)

例 [305](#)

SYSOPTF データ・セットで [295](#), [429](#)

z/OS のもとでの [299](#)

z/OS ユニックス のもとでの [313](#)

リスト例 [525](#)

85 COBOL 標準 準拠 [375](#)

ADATA [378](#)

ADV [378](#)

AFP

パフォーマンスの考慮事項 [845](#)

AMODE 64 [588](#)

APOST [379](#)

ARCH

パフォーマンスの考慮事項 [845](#)

ARITH

説明 [381](#)

パフォーマンスの考慮事項 [845](#)

AWO

コンパイラー・オプション (続き)

AWO (続き)

説明 [382](#)

パフォーマンスの考慮事項 [845](#)

BLOCK0

説明 [383](#)

パフォーマンスの考慮事項 [845](#)

BUFSIZE [384](#)

CICS [384](#)

CODEPAGE [385](#)

COMPILE [388](#)

COPYLOC [389](#)

COPYRIGHT [390](#)

CURRENCY [390](#)

DBCS [392](#)

DECK [393](#)

DEFINE [393](#)

DIAGTRUNC [394](#)

DISPSIGN [395](#)

DLL [396](#)

DUMP [398](#)

DYNAM [398](#), [846](#)

EXPORTALL [401](#)

FASTSRT

パフォーマンスの考慮事項 [846](#)

FLAG [402](#), [492](#)

FLAGSTD [403](#)

HGPR

パフォーマンスの考慮事項 [846](#)

IMS および CICS のもとでの [546](#)

IMS で推奨される [573](#)

INITCHECK

パフォーマンスの考慮事項 [846](#)

INITIAL [407](#)

INLINE

パフォーマンスの考慮事項 [846](#)

INTDATE [408](#)

INVDATA [408](#)

JAVAIOP [412](#)

LANGUAGE

説明 [413](#)

バッチ・コンパイルでの例 [305](#)

LINECOUNT [414](#)

LIST [414](#), [500](#)

LP [415](#)

LSACHECK [416](#)

MAP [418](#), [495](#), [500](#)

MAXPCF [419](#)

MAXPCF(nnn)

パフォーマンスの考慮事項 [846](#)

MDECK [419](#)

NOCOMPILE [490](#)

NOFASTSRT [258](#)

NSYMBOL [421](#)

NUMCHECK

パフォーマンスの考慮事項 [846](#)

NUMPROC [426](#)

NUMPROC(PFD)

パフォーマンスの考慮事項 [847](#)

NUMPROC(PFD|NOPFD) [59](#)

OBJECT [427](#)

OFFSET [427](#)

OPTFILE [428](#)

OPTIMIZE

コンパイラー・オプション (続き)

OPTIMIZE (続き)

説明 [429](#)

パフォーマンスの考慮事項 [843](#), [847](#)

OUTDD [430](#)

PARMCHECK

パフォーマンスの考慮事項 [847](#)

PGMNAME [432](#)

QUALIFY [434](#)

QUOTE [379](#)

RENT

説明 [435](#)

パフォーマンスの考慮事項 [847](#)

RMODE

説明 [436](#)

パフォーマンスの考慮事項 [847](#)

RULES [437](#)

SEQUENCE [439](#)

SMARTBIN

パフォーマンスの考慮事項 [847](#)

SOURCE [442](#), [500](#)

SPACE [443](#)

SQL

説明 [443](#)

Db2 での使用 [561](#)

SQLCCSID

ストリング・データの CCSID への影響 [563](#)

説明 [444](#)

パフォーマンスの考慮事項 [565](#)

Db2 コプロセッサで推奨する [564](#)

SQLIMS [445](#)

SSRANGE

パフォーマンスの考慮事項 [848](#)

STGOPT [447](#)

SUPPRESS [448](#)

SYSOPTF データ・セットでの指定 [295](#)

TERMINAL [448](#)

TEST

説明 [448](#)

デバッグに使用 [499](#)

パフォーマンスの考慮事項 [848](#)

THREAD

説明 [453](#)

デバッグに関する制約事項 [488](#)

パフォーマンスの考慮事項 [848](#)

TRUNC

説明 [454](#)

パフォーマンスの考慮事項 [848](#)

TSO での指定 [301](#)

TUNE

パフォーマンスの考慮事項 [848](#)

VBREF [458](#), [500](#)

VLR

説明 [458](#)

VSAMOPENFS [461](#)

WORD [461](#)

XMLPARSE [462](#)

z/OS のもとでの指定 [301](#)

z/OS ユニックス のもとでの指定 [312](#)

ZONECHECK [464](#)

ZONEDATA [465](#)

ZWB [468](#)

コンパイラー・オプションの階層

バッチで [304](#)

コンパイラー・オプションの階層 (続き)

SYSOPTF データ・セットで [429](#)

z/OS のもとでの [299](#)

z/OS ユニックス のもとでの [313](#)

コンパイラー限界値

AMODE 64 サポート付き [588](#)

コンパイラー指示ステートメント

概要 [20](#)

説明 [481](#)

コンパイラー出力 [500](#)

コンパイラー・ディレクティブ

データ [597](#)

コンパイラー・データ・セット

コンパイルに必要な [291](#)

入出力 [291](#)

cob2 を使用した [318](#)

SYSADATA (ADATA レコード) [297](#)

SYSDEBUG (デバッグ・レコード) [298](#)

SYSIN [294](#)

SYSJAVA [297](#)

SYSLIB (ライブラリー) [296](#)

SYSLIN (オブジェクト・コード) [297](#)

SYSMDECK (ライブラリー処理) [298](#)

SYSOPTF [295](#)

SYSOUT (リスト) [296](#)

SYS PUNCH (オブジェクト・コード) [297](#)

SYSTEM (メッセージ) [296](#)

z/OS UNIX ファイルシステムでの [280](#), [286](#)

コンパイラー・マニュアル・ページを表示するための -? cob2

オプション [318](#)

コンパイラー・マニュアル・ページを表示するための -help

cob2 オプション [317](#)

コンパイラー・リスト

入手 [500](#)

コンパイル

アセンブラー・プログラムからの [289](#)

オブジェクト指向アプリケーション

例 [323](#), [328](#)

cob2 コマンド [321](#)

JCL または TSO/E の使用 [326](#)

z/OS ユニックス のもとでの [321](#)

およびバインド [367](#)

カタログ式プロシージャによる

コンパイル [281](#)

コンパイル、リンク・エディット、実行 [283](#)

コンパイルおよびリンク・エディット [282](#)

結果 [301](#)

シェル・スクリプトの使用 [319](#)

制御 [299](#)

データ・セット [291](#)

バッチ [302](#)

85 COBOL 標準 への準拠 [375](#)

AMODE 64 プログラム [588](#), [590](#), [593-596](#)

cob2 コマンドの使用

概要 [313](#)

例 [315](#)

DLL [314](#)

JCL (ジョブ制御言語) による [279](#)

TSO のもとでの

概要 [287](#)

CLIST の例 [288](#)

z/OS のもとでの [279](#)

z/OS ユニックス のもとでの [311](#)

z/OS ユニックスファイル [282](#)

コンパイルおよびリンク、z/OS ユニックス シェルでの
オブジェクト指向アプリケーション
例 [323](#)
cob2 コマンド [322](#)
概要 [313](#)
例 [315](#)
DLL [314](#)
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 [402](#)
重大度レベル [308](#)
コンパイラーの指示エラー [307](#)
コンパイルおよびリンク・ステップの表示 [317](#)
ダンプ、生成 [398](#)
表示後のコンパイルおよびリンク・ステップの実行 [317](#)
コンパイルに使用されるデータ・セット [291](#)
コンパイルの統計 [504](#)
コンパイル用作業データ・セット [291](#)

[サ行]

再帰呼び出し
コーディング [617](#)
識別 [4](#)
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
[605](#)
INITIAL 属性なしのサブプログラム [606](#), [607](#)
再実行依頼、ジョブの [829](#)
再始動
概要 [823](#)
自動 [827](#)
自動または据え置き [823](#)
据え置き [827](#)
再始動、プログラムの [826](#)
最大または最小項目、検出 [127](#)
最適化
一括表示表現 [838](#)
一貫性のあるデータ [840](#)
構造化プログラミング [837](#)
コンパイラー・オプションの影響 [844](#)
指標計算 [842](#)
指標付け [841](#)
添え字計算 [842](#)
添え字付け [841](#)
定数データ項目 [838](#)
テーブル・エレメント [841](#)
到達不能コード [843](#)
トップダウン・プログラミング [838](#)
ネストされたプログラムの統合 [843](#)
パック 10 進数データ項目 [839](#)
パフォーマンスにおける意味 [841](#)
パフォーマンスへの影響 [837](#)
パラメーター引き渡しの影響 [626](#)
含まれているプログラムの統合 [843](#)
プロシーチャー統合 [844](#)
未使用データ項目 [429](#), [506](#)
ライン外の PERFORM [838](#)
ALTER ステートメントの回避 [838](#)
BINARY データ項目 [839](#)
OCCURS DEPENDING ON [841](#)
最適化プログラム
概要 [843](#)

再入可能プログラム [620](#)
サイン表記 [59](#)
索引
値を割り当てる [78](#)
エレメントの変位の計算例 [76](#)
キーの誤りの検出 [275](#)
初期化 [78](#)
増分または減分 [78](#)
他のテーブルの参照 [78](#)
定義 [76](#)
範囲検査 [492](#)
OCCURS INDEXED BY 節による作成 [77](#)
索引付きファイル編成
指定 [204](#)
説明 [172](#)
削除、VSAM ファイルからのレコードの [217](#)
作成
オブジェクト [767](#)
オブジェクト・コード [297](#)
可変長テーブル [84](#)
関連データ・ファイル [297](#)
ライブラリー処理出力ファイル [298](#)
SYSJAVA ファイル [297](#)
z/OS での QSAM ファイル [191](#), [193](#)
z/OS での行順次ファイル [238](#)
サフィックス、cob2 での [318](#)
サブクラス
インスタンス・データ [772](#)
コーディング
概要 [770](#)
例 [773](#)
サブストリング
その参照変更 [117](#)
テーブル・エレメントの [118](#)
サブプログラム
終了
効果 [604](#)
説明 [603](#)
定義 [623](#)
メインプログラム [603](#)
リンケージ
共通データ項目 [626](#)
PROCEDURE DIVISION [628](#)
サポート xxxiii, [1077](#)
算術演算
エラー処理 [266](#)
組み込み関数を使用した [63](#)
コーディングが容易な COMPUTE ステートメント [62](#)
算術式
括弧で囲まれた [62](#)
参照修飾子としての [119](#)
説明 [62](#)
非算術ステートメントでの [867](#)
算術比較 [69](#)
算術評価
固定小数点と浮動小数点の対比 [68](#)
精度 [859](#)
中間結果 [859](#)
データ形式の変換 [58](#)
パフォーマンスに関するヒント [839](#)
変換と精度 [58](#)
優先順位 [63](#), [860](#)
例 [68](#), [70](#)
参照修飾子

参照修飾子 (続き)
 組み込み関数、例 [120](#)
 としての算術式 [119](#)
 としての変数 [118](#)
 参考文献 [1077](#)
 参照変更
 国別データ [118](#)
 組み込み関数 [117](#)
 生成された XML 文書 [728](#)
 テーブル [77](#), [118](#)
 範囲外の値 [118](#)
 例 [118](#)
 SSRANGE による検査式 [446](#)
 UTF-8 文書 [162](#)
 サンプル・プログラム [1006](#)
 支援テクノロジー [1026](#)
 シグニチャー
 固有でなければならない [752](#)
 定義 [752](#)
 シグニチャー情報バイト
 有効なコンパイラー・オプション [513](#)
 DATA DIVISION [513](#)
 ENVIRONMENT DIVISION [513](#)
 PROCEDURE DIVISION [513](#)
 指数
 固定小数点演算で評価される [862](#)
 パフォーマンスに関するヒント [840](#)
 浮動小数点演算で評価される [866](#)
 システム決定のブロック・サイズ
 コンパイラー・データ・セット [294](#)
 QSAM ファイル [185](#), [383](#)
 システム・ダンプ [265](#)
 システム日付
 CICS のもとで [547](#)
 システム名 [6](#)
 実行
 COBOL AMODE 64 アプリケーション [369](#)
 実行時
 パフォーマンス [837](#)
 ファイル名の変更 [10](#)
 マルチスレッド化制限 [657](#)
 z/OS での引数へのアクセス
 概要 [637](#)
 例 [637](#)
 z/OS ユニックス での引数へのアクセス
 概要 [584](#)
 例 [584](#)
 実行する、オブジェクト指向アプリケーションを
 JCL または TSO/E の使用 [327](#)
 XPLINK リンケージ [329](#)
 z/OS ユニックス のもとの
 概要 [324](#)
 XPLINK リンケージ [329](#)
 実行単位
 説明 [603](#)
 マルチスレッド化での役割 [652](#)
 指定
 COBOL 専用ランタイム・オプション [369](#)
 COBOL ランタイム オプション [470](#)
 自動再始動 [827](#)
 指標付け
 エレメントの変位の計算例 [76](#)
 添え字付けより望ましい [841](#)
 定義 [76](#)
 指標付け (続き)
 テーブル [77](#)
 例 [83](#)
 指標データ項目
 添え字や指標として使用できない [78](#)
 USAGE IS INDEX 節による作成 [78](#)
 終了 [604](#)
 主記憶域のバッファーへの割り振り [384](#)
 出力
 行順次ファイル用のコーディング [239](#)
 データ・セット [296](#)
 ファイルへ [171](#)
 CICS 用のコーディング [546](#)
 QSAM ファイル用のコーディング [186](#)
 VSAM ファイル用のコーディング [209](#)
 z/OS でのコンパイラーからの [293](#)
 出力ファイル、cob2 での [318](#)
 出力プロシージャ
 コーディング [248](#)
 制約事項 [249](#)
 有効でない FASTSORT オプション [256](#)
 例 [249](#), [253](#)
 RETURN または RETURN INTO が必要 [248](#)
 順次記憶装置 [172](#)
 順次ファイル編成 [171](#)
 順序、評価の
 コンパイラー・オプション [376](#)
 算術演算子 [63](#), [860](#)
 使用可能ファイル
 QSAM [188](#)
 VSAM [219](#)
 状況コード、VSAM ファイル
 説明 [273](#)
 例 [273](#)
 条件式
 EVALUATE ステートメント [99](#)
 IF ステートメント [99](#)
 PERFORM ステートメント [109](#)
 条件処理
 言語環境プログラム の使用 [854](#)
 入力または出力プロシージャでの [249](#)
 QSAM ファイルのクローズ [190](#)
 VSAM ファイルのクローズ [217](#)
 条件ステートメント
 オブジェクト参照 [762](#)
 概要 [19](#)
 NOT 句を指定した [20](#)
 条件付きコンパイル出力、例 [537](#)
 条件の検査 [104](#)
 照合シーケンス
 国別キーのバイナリー [252](#)
 指定 [7](#)
 シンボリック文字 [8](#)
 代替
 選択 [254](#)
 例 [8](#)
 非数値比較 [7](#)
 文字の序数位置 [127](#)
 ASCII [7](#)
 EBCDIC [7](#)
 HIGH-VALUE [7](#)
 ISO 7 ビット・コード [7](#)
 LOW-VALUE [7](#)
 MERGE [7](#), [254](#)

照合シーケンス (続き)

NATIVE [7](#)
SEARCH ALL [7](#)
SORT [7](#), [254](#)
STANDARD-1 [7](#)
STANDARD-2 [7](#)

常駐モード、定義 [40](#)

商標 [1031](#)

初期化

インスタンス・データ [767](#)
可変長グループ [87](#)
国別グループ項目
 INITIALIZE の使用 [31](#), [80](#)
 VALUE 節の使用 [82](#)
グループ項目
 INITIALIZE の使用 [31](#), [79](#)
 VALUE 節の使用 [82](#)
構造、INITIALIZE による [31](#)
テーブル
 エレメントのすべての出現 [82](#)
 グループ・レベルの [82](#)
 それぞれの項目の個別の [81](#)
 INITIALIZE の使用 [79](#)
 PERFORM VARYING の使用 [109](#)

例 [28](#)

初期ヒープ・ストレージ・マップ [528](#)

ジョブ再実行要求 [829](#)

ジョブ・ストリーム [603](#)

資料 [1077](#)

身体障がい [1026](#)

診断、プログラムの [504](#)

シンボリック定数 [838](#)

スイッチおよびフラグ

 スイッチをオフに設定する例 [107](#)

 スイッチをオンに設定する例 [106](#)

説明 [104](#)

 単一値のテストの例 [105](#)

定義 [105](#)

 複数値のテストの例 [105](#)

リセット [106](#)

スイッチ状況条件 [104](#)

数学

 組み込み関数 [63](#), [68](#)

 言語環境プログラム 呼び出し可能サービス [65](#), [854](#)

数字組み込み関数

 言語環境プログラム 呼び出し可能サービスとの違い [65](#)

 整数、浮動小数点、混合 [63](#)

 同等の言語環境プログラム 呼び出し可能サービス [64](#)

 ネストされた [64](#)

 引数としてのテーブル・エレメント [64](#)

 引数としての特殊レジスター [64](#)

 用途 [63](#)

 例

 最小 [120](#)

 整数 [120](#)

 長さ [67](#), [128](#)

 ログ [68](#)

 ANNUITY [68](#)

 CURRENT-DATE [67](#)

 INTEGER-OF-DATE [67](#)

 MAX [67](#), [95](#), [127](#), [128](#)

 MEAN [68](#)

 MEDIAN [68](#), [95](#)

 NUMVAL [123](#)

数字組み込み関数 (続き)

例 (続き)

 NUMVAL-C [67](#), [123](#)

 ORD [127](#)

 ORD-MAX [95](#)

 PRESENT-VALUE [67](#)

 RANGE [68](#), [95](#)

 REM [68](#)

 SQRT [68](#)

 SUM [95](#)

数字編集データ

初期化

 例 [30](#)

 INITIALIZE の使用 [80](#)

定義 [134](#)

編集記号 [51](#)

BLANK WHEN ZERO 節

 コーディング、数値データでの [134](#)

例 [51](#)

PICTURE 節 [51](#)

USAGE DISPLAY

 初期化の例 [30](#)

表示 [51](#)

USAGE NATIONAL

 初期化の例 [30](#)

表示 [51](#)

数値データ

外部 10 進数

 USAGE DISPLAY [53](#)

 USAGE NATIONAL [53](#)

外部浮動小数点

 USAGE DISPLAY [53](#)

 USAGE NATIONAL [54](#)

国別 10 進数 (USAGE NATIONAL) [53](#)

国別との比較 [149](#)

国別浮動小数点 (USAGE NATIONAL) [54](#)

ストレージ形式 [52](#)

ゾーン 10 進数 (USAGE DISPLAY)

形式 [53](#)

サイン表記 [59](#)

定義 [49](#)

内部浮動小数点

 USAGE COMPUTATIONAL-1 (COMP-1) [55](#)

 USAGE COMPUTATIONAL-2 (COMP-2) [55](#)

バイナリー

 USAGE BINARY [54](#)

 USAGE COMPUTATIONAL (COMP) [54](#)

 USAGE COMPUTATIONAL-4 (COMP-4) [54](#)

 USAGE COMPUTATIONAL-5 (COMP-5) [54](#)

パック 10 進数

 サイン表記 [59](#)

 USAGE COMPUTATIONAL-3 (COMP-3) [55](#)

 USAGE PACKED-DECIMAL [55](#)

表示浮動小数点 (USAGE DISPLAY) [53](#)

変換

 固定小数点と浮動小数点との間の [58](#)

精度 [58](#)

 MOVE による UTF-8 への [159](#)

 MOVE による国別への [142](#)

編集記号 [51](#)

PICTURE 節 [49](#), [51](#)

USAGE DISPLAY [49](#)

USAGE NATIONAL [49](#)

USAGE とは無関係に代数値の比較が可能 [150](#)

数値のクラス・テスト
 検査、有効データの [60](#)
 NUMPROC、NUMCLS の影響 [60](#)
 数値比較 [103](#)
 数値リテラルの説明 [26](#)
 据え置き再始動 [827](#)
 スキップ、レコードのブロックの [184](#)
 ステートメント
 暗黙の範囲終了符号 [21](#)
 コンパイラー指示 [20](#)
 条件 [19](#)
 定義 [19](#)
 ネスト・レベル [505, 537](#)
 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、プログラムで使用される [500](#)
 ステートメント相互参照リスト
 説明 [500](#)
 ストライド、テーブル [842](#)
 ストライブ拡張形式 QSAM ファイル [197](#)
 スtring
 処理 [111](#)
 ヌル終了 [629](#)
 Java
 宣言する [791](#)
 取り扱い [795](#)
 ストレージ
 管理、言語環境プログラム 呼び出し可能サービス [854](#)
 装置
 順次 [172](#)
 直接アクセス [172](#)
 ソート時の使用 [260](#)
 引数のための [625](#)
 マッピング [500](#)
 文字データ [148](#)
 スパン・ファイル [182](#)
 スパン・レコード形式
 説明 [181](#)
 要求 [181](#)
 レイアウト [183](#)
 スレッド化
 および事前初期設定 [653](#)
 制御転送 [653](#)
 プログラムの終了 [653](#)
 z/OS ユニックスの考慮事項 [579](#)
 制御インターバル・サイズ (CISZ) のパフォーマンスの考慮事項 [227, 850](#)
 制限事項、コンパイラーの
 ユーザー・データ [12](#)
 DATA DIVISION [12](#)
 静的データ、定義 [743](#)
 静的データ域、ストレージの割り振り [42](#)
 静的メソッド
 定義 [743](#)
 呼び出し [777](#)
 静的呼び出し
 行う [606](#)
 動的呼び出しでの [611](#)
 パフォーマンス [611](#)
 例 [612](#)
 製品サポート xxxiii, [1077](#)
 制約事項
 オブジェクト指向プログラム [743](#)
 制約事項 (続き)
 添え字付け [77](#)
 入出力プロシージャ [249](#)
 CICS
 コーディング [545](#)
 ソート [262](#)
 ファイル [6](#)
 分離型の変換プログラム [551](#)
 呼び出し [547](#)
 16 MB 境界 [546](#)
 FILE を使用した検証を伴う構文解析 [699](#)
 OUTDD コンパイラー・オプション [431](#)
 Db2 コプロセッサ [562](#)
 IMS
 コーディング [6, 569](#)
 16 MB 境界 [546](#)
 IMS SQL コプロセッサ [572](#)
 SQL コンパイラー・オプション [562](#)
 SQL ステートメント [558](#)
 SQLIMS コンパイラー・オプション [572](#)
 セクション
 グループ化 [110](#)
 宣言 [22](#)
 定義 [19](#)
 セグメント化 [844](#)
 設定
 指標 [78](#)
 指標データ項目 [78](#)
 スイッチおよびフラグ [106](#)
 ゼロ抑制
 BLANK WHEN ZERO 節の例 [51](#)
 PICTURE 記号 Z [51](#)
 宣言型プロシージャ
 EXCEPTION/ERROR
 マルチスレッド化 [270](#)
 USE FOR DEBUGGING [488](#)
 相互参照
 組み込み [500](#)
 コピーブック [500](#)
 ステートメント [500](#)
 ステートメント・リスト [458](#)
 データおよびプロシージャ名 [494](#)
 テキスト名およびデータ・セット [494](#)
 特殊定義記号 [535](#)
 プログラム名 [533](#)
 リスト [463](#)
 COPY/BASIS [533](#)
 COPY/BASIS ステートメント [500](#)
 相対ファイル編成 [172](#)
 装置
 クラス [291](#)
 の要件 [291](#)
 添え字
 定義 [76](#)
 範囲検査 [492](#)
 変数、例 [76](#)
 リテラル、例 [75](#)
 添え字付け
 参照変更 [77](#)
 制約事項 [77](#)
 相対 [77](#)
 定義 [76](#)
 データ名またはリテラルを使用 [77](#)
 変数、例 [76](#)

添え字付け (続き)

リテラル、例 [75](#)

例 [82](#)

ソース・コード

行番号 [505](#), [506](#), [511](#), [537](#)

コンパイラー・データ・セット [294](#)

プログラム・リスト [301](#)

リストの説明 [500](#)

ソート

オリジナル・シーケンスの保持 [254](#)

可変長レコード [250](#)

完了コード [254](#)

キー

概要 [244](#)

定義 [252](#)

基準 [252](#)

作業ファイル

説明 [245](#)

終了 [255](#)

出力プロシージャ

コーディング [248](#)

例 [249](#), [253](#)

使用ストレージ [260](#)

診断メッセージ [255](#)

制御ステートメントの受け渡し [259](#)

正常終了の判別 [254](#)

制約事項 [243](#)

説明 [243](#)

代替照合シーケンス [254](#)

チェックポイント・リスタート [261](#)

テーブル

概要 [94](#)

動作の制御 [258](#)

特殊レジスター [258](#)

入出力プロシージャに関する制約事項 [249](#)

入力プロシージャ

コーディング [247](#)

例 [253](#)

パフォーマンス

可変長ファイル [251](#)

FASTSRT [255](#)

ファイルの説明 [245](#)

プロセス [244](#)

ワークスペース [261](#)

CICS のもとで [262](#)

FASTSRT コンパイラー・オプション

同じ QSAM ファイルを入出力両用に使用 [257](#)

の要件 [256](#)

パフォーマンスの向上 [255](#)

NOFASTSRT コンパイラー・オプション [258](#)

z/OS データ・セット定義用の DD ステートメント [250](#)

z/OS のもとで必要なデータ・セット [250](#)

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 [468](#)

形式 [53](#)

サイン表記 [59](#)

例 [49](#)

属性メソッド [756](#)

ソフトコピー情報 [xxxiii](#)

[タ行]

代替入り口点の呼び出し [620](#)

代替索引

代替索引 (続き)

作成 [220](#)

使用 [205](#)

パス [221](#)

パスワード [218](#)

パフォーマンスの考慮事項 [226](#)

例 [221](#)

代替照合シーケンス

選択 [254](#)

例 [8](#)

代替予約語テーブル

指定 [461](#)

CICS [553](#)

ダイナミック・リンク・ライブラリー

オブジェクト指向のための作成 [322](#)

オブジェクト指向プログラムでの使用 [324](#)

コンパイル [640](#)

作成

概要 [639](#)

z/OS ユニックス シェルから [314](#)

説明 [639](#)

動的呼び出しでの使用 [645](#)

必要なコンパイラー・オプション [314](#)

リンク [641](#)

C/C++ プログラムでの使用 [648](#)

CALL identifier の使用 [643](#)

DLL サポートのあるプログラムは再入可能でなければなら
ない [620](#)

DLL 用のバインダー・オプション [641](#)

Java とのインターオペラビリティでの使用 [324](#)

Java とのインターオペラビリティのための [322](#)

OO COBOL アプリケーションで [648](#)

z/OS UNIX ファイルシステム における探索順序 [644](#)

対話式プログラムの例 [1010](#)

多重継承、許可されていない [746](#), [770](#)

探索

テーブル

概要 [91](#)

逐次探索 [91](#)

二分探索 [92](#)

パフォーマンス [91](#)

名前の宣言の [616](#)

探索順序

z/OS UNIX ファイルシステム における DLL [644](#)

短縮リストの例 [502](#)

ダンプ

要求 [265](#)

DUMP コンパイラー・オプションを指定した [301](#)

端末へのメッセージの送信 [448](#)

段落

グループ化 [110](#)

定義 [19](#)

チェーン・リスト処理

概要 [629](#)

例 [630](#)

チェックポイント

概要 [823](#)

再始動用の JCL の例 [829](#)

生成されるメッセージ [826](#)

設計 [824](#)

設定 [823](#)

ソート時の制限 [824](#)

単一

ディスク [826](#)

- チェックポイント (続き)
 - 単一 (続き)
 - テープ [825](#)
 - テスト [824](#)
 - 複数 [824, 826](#)
 - メソッド [823](#)
 - レコード・データ・セット [825](#)
 - 85 COBOL 標準 [824](#)
 - DFSORT 時の再始動 [261](#)
- 置換
 - データ項目 (INSPECT) [120](#)
 - テキスト、Db2 の考慮事項 [566](#)
 - QSAM ファイル内のレコード [189](#)
 - VSAM ファイル内のレコード [216](#)
- 置換文字 [140](#)
- 逐次探索
 - 説明 [91](#)
 - 例 [92](#)
- 中間結果 [859](#)
- 中国語 GB 18030 データ
 - プロセッシング [167](#)
- チューニング考慮事項、パフォーマンス [844, 845](#)
- 直接アクセス
 - 記憶装置 (DASD) [227](#)
 - 直接指標付け [77](#)
 - ファイル編成 [172](#)
- 追加、レコードの
 - 行順次ファイルへの [241](#)
 - QSAM ファイルへの [188](#)
 - VSAM ファイルへの [216](#)
- 通貨記号
 - 使用 [70](#)
 - 複数の文字 [71](#)
 - ユーロ [71](#)
 - 16 進数リテラル [71](#)
- 定義
 - デバッグ・データ・セット [298](#)
 - ファイルの概要 [9, 171](#)
 - ライブラリー [296](#)
 - QSAM ファイル
 - z/OS に対する [191, 193](#)
 - VSAM ファイル
 - z/OS に対する [219](#)
 - z/OS のもとでのソートまたはマージ・ファイル [250](#)
 - z/OS への行順次ファイル [238](#)
- 定数域 [529](#)
- 定様式ダンプ [265](#)
- ディレクトリー
 - パスの追加 [317](#)
- データ
 - 受け渡し [623](#)
 - 英数字と DBCS との間の変換 [867](#)
 - グループ化 [628](#)
 - 形式、数値タイプ [52](#)
 - 形式の変換 [58](#)
 - 効率的な実行 [837](#)
 - 数字 [49](#)
 - 妥当性検査 [60](#)
 - 非互換 [60](#)
 - 分割 (UNSTRING) [113](#)
 - 命名 [13](#)
 - レコード・サイズ [13](#)
 - 連結 (STRING) [111](#)
- データ圧縮 [831](#)
- データ域、動的 [398](#)
- データおよびプロシージャ名相互参照の記述 [494](#)
- データ記述項目 [12](#)
- データ項目
 - 可変位置 [88](#)
 - 基本、定義 [24](#)
 - 共通、サブプログラム・リンケージの [626](#)
 - 組み込み関数を使用した評価 [126](#)
 - 組み込み関数を使用した変換 [122](#)
 - グループ、定義 [24](#)
 - 最小項目または最大項目の検出 [127](#)
 - サブストリングの参照 [117](#)
 - 参照変更 [117](#)
 - 指標によるテーブル・エレメントの参照 [76](#)
 - 初期化の例 [28](#)
 - 数字 [49](#)
 - 分割 (UNSTRING) [113](#)
 - 変換、大文字または小文字への [123](#)
 - マップ [301](#)
 - 未使用 [429, 506](#)
 - 文字から数値への変換 [123](#)
 - 文字のカウント (INSPECT) [120](#)
 - 文字の逆順 [123](#)
 - 文字の置換 (INSPECT) [120](#)
 - 文字の変換 (INSPECT) [120](#)
 - 連結 (STRING) [111](#)
 - 16 進数字または 2 進数字から変換 [125](#)
 - 16 進数字または 2 進数字に変換 [125](#)
 - 2 バイト文字を含む英数字 [867](#)
 - DBCS [867](#)
 - Java の型のコーディング [789](#)
- データ項目の比較
 - オブジェクト参照 [762](#)
 - ゾーン 10 進数および英数字、ZWB の影響 [468](#)
- national
 - 英字、英数字、または DBCS と [150](#)
 - 英数字グループと [150](#)
 - 概要 [148](#)
 - 数値との [149](#)
 - 2 つのオペランド [149](#)
- UTF-8
 - 英字、英数字と [161](#)
 - 英数字グループと [161](#)
 - 概要 [160](#)
 - 2 つのオペランド [161](#)
- データ項目の分割 (UNSTRING) [113](#)
- データ項目の変換
 - 英数字へ
 - DISPLAY による [36](#)
 - DISPLAY-OF を使用した [143](#)
 - 大文字または小文字への
 - 組み込み関数を使用した [123](#)
 - INSPECT による [122](#)
 - 国別から UTF-8 への [162](#)
 - 国別から中国語 GB 18030 へ [167](#)
 - 国別データでの例外 [144](#)
 - 国別と
 - 中国語 GB 18030 から [167](#)
 - ACCEPT による [35](#)
 - MOVE を使用した [142](#)
 - NATIONAL-OF を使用した [143, 160](#)
 - UTF-8 から [162](#)
 - 組み込み関数を使用した [122](#)
 - コード・ページ間の [124](#)

データ項目の変換 (続き)

- 精度 [58](#)
- データ・フォーマット間の [58](#)
- 文字の逆順 [123](#)
- 16 進数字から [125](#)
- 16 進数字に [125](#)
- 16 進数字または 2 進数字から
組み込み関数を使用した [125](#)
- 16 進数字または 2 進数字に
組み込み関数を使用した [125](#)
- 2 進数字から [125](#)
- 2 進数字に [125](#)
- INSPECT による [120](#)
- INTEGER、INTEGER-PART による整数への [120](#)
- NUMVAL、NUMVAL-C による数値への [123](#)
- UTF-8 へ

MOVE を使用した [159](#)

データ項目の連結 (STRING) [111](#)

データ・セット

- 環境変数を使用した定義 [175](#)
- コンパイラー・オプション [295](#)
- 出力 [296](#)
- ソース・コード [294](#)
- 代替データ・セット名 [289](#)
- チェックポイント/再始動の例 [829](#)
- チェックポイント・レコード [825](#)
- 名前、代替 [290](#)
- ファイル、同じ意味 [6](#)
- JAVAERR [327](#)
- JAVAIN [327](#)
- JAVAOUT [327](#)
- SYSADATA [297](#)
- SYSDEBUG [298](#)
- SYSIN [294](#)
- SYSJAVA [297](#)
- SYSLIB [296](#)
- SYSLIN [297](#)
- SYSMDECK [298](#)
- SYSOPTF [295](#)
- SYSPRINT [296](#)
- SYSPUNCH [297](#)
- SYSTEM [296](#)

データ操作

- 文字データ [111](#)
- DBCS データ [867](#)

データ定義 [506](#)

データ定義属性コード [506](#)

データの検査 (INSPECT) [120](#)

データ名

- 相互参照 [531](#)
- 相互参照リスト [302](#)
- MAP リスト内の [506](#)
- OMITTED [14](#)
- VSAM ファイルのパスワード [218](#)

テープ・ファイル

- 逆順 [187](#)
- パフォーマンス [185](#)

テーブルの動的なロード [79](#)

適合要件

- 85 COBOL 標準 [375](#)
- INVOKE でのオブジェクト参照引き渡しの例 [765](#)
- INVOKE の RETURNING 句 [766](#)
- INVOKE の USING 句 [764](#)

テキスト名相互参照、記述 [494](#)

出口モジュール

- 生成されるエラー・メッセージ [903](#)
- メッセージ重大度のカスタマイズ [893](#)
- ライブラリー名の代わりに使用される [888](#)
- COBOL プログラムの呼び出し [886](#)
- SYSADATA データ・セットのために呼び出される [892](#)
- SYSLIB の代わりに使用される [888](#)
- SYSPRINT の代わりに使用される [891](#)

テスト

- 条件 [108](#)
- 数値オペランド [103](#)
- データ [103](#)
- UPSI スイッチ [104](#)

鉄道線路構文図の読み方 [xxxi](#)

デバッグ

- 概要 [485](#)
- コンパイラー・オプション
- 概要 [490](#)
- TEST に関する制約事項 [488](#)
- THREAD に関する制約事項 [488](#)
- データ・セットの定義 [298](#)
- デバッガーの使用 [499](#)
- パフォーマンスとの対比 [451](#)
- ランタイム・オプション [488](#)
- COBOL 言語機能の使用 [486](#)

デバッグ、言語機能の

- クラス・テスト [487](#)
- 宣言部分 [488](#)
- デバッグ行 [488](#)
- デバッグ・ステートメント [488](#)
- 範囲終了符号 [486](#)
- ファイル状況キー [487](#)
- DISPLAY ステートメント [486](#)
- INITIALIZE ステートメント [488](#)
- SET ステートメント [488](#)
- WITH DEBUGGING MODE 節 [488](#)

統計組み込み関数 [68](#)

到達不能コード [843](#)

動的データ域、ストレージの割り振り [42](#)

動的ファイル割り振り

- 環境変数を使用
- 行順次ファイル [238](#)
- QSAM ファイル [191](#)
- VSAM ファイル [222](#)

割り振りの順序 [175](#)

CBLQDA を使用 [188](#)

動的呼び出し

- 行う [607](#)
- 使用時期 [608](#)
- 静的呼び出しでの [611](#)
- 制約事項 [607](#)
- パフォーマンス [611](#)
- 例 [612](#)
- DLL リンケージでの使用 [645](#)

読者のコメント

送付 [xxxiii](#)

特殊機能指定 [6](#)

特殊レジスター

組み込み関数の引数 [64](#)

ADDRESS OF

CALL ステートメントでの使用 [624](#)

JNIEnvPtr

JNI 呼び出し可能サービスのための使用 [786](#)

LENGTH OF [129](#), [624](#)

特殊レジスター (続き)

RETURN-CODE [632](#)

SORT-RETURN

ソートまたはマージの終了 [255](#)

ソートまたはマージの成功の判断 [254](#)

WHEN-COMPILED [130](#)

XML 構文解析での使用 [691](#), [693](#)

XML-CODE [692](#), [694](#)

XML-EVENT [692](#), [693](#)

XML-INFORMATION [692](#)

XML-NAMESPACE [692](#), [697](#)

XML-NAMESPACE-PREFIX [692](#), [697](#)

XML-NNAMESPACE [692](#), [697](#)

XML-NNAMESPACE-PREFIX [692](#), [697](#)

XML-NTEXT [692](#), [696](#)

XML-TEXT [692](#), [696](#)

特殊レジスター・テーブル [530](#)

特記事項 [1029](#)

トップダウン・プログラミング

回避するための構成 [838](#)

トラブルシューティング

COBOL/Java 相互運用性 [360](#)

[ナ行]

内部浮動小数点データ (COMP-1、COMP-2) [55](#)

長さを検出する、データ項目の [129](#)

名前宣言

探索 [616](#)

日時操作

言語環境プログラム 呼び出し可能サービス [854](#)

二分探索

説明 [92](#)

例 [93](#)

入出力

エラー後のロジック・フロー [267](#)

エラーの検査 [271](#)

概要 [171](#)

コーディングの概要 [173](#)

処理エラー

行順次ファイル [241](#)

QSAM ファイル [190](#), [267](#)

VSAM ファイル [217](#), [267](#)

FASTSORT オプションによる制御 [402](#)

入出力エラー宣言でのデッドロック [270](#)

入出力コーディング

誤った索引キーの検出 [275](#)

エラー処理技法 [267](#)

正常操作の検査 [271](#)

AT END (ファイルの終わり) 句 [269](#)

EXCEPTION/ERROR 宣言 [270](#)

VSAM 状況コードの検査 [273](#)

入力

行順次ファイル用のコーディング [239](#)

ファイルから [171](#)

CICS 用のコーディング [546](#)

QSAM ファイル用のコーディング [186](#)

VSAM ファイル用のコーディング [209](#)

z/OS のもとでのコンパイラへの [292](#)

入力プロシージャ

コーディング [247](#)

制約事項 [249](#)

有効でない FASTSORT オプション [256](#)

例 [253](#)

入力プロシージャ (続き)

RELEASE または RELEASE FROM が必要 [247](#)

ヌル終了ストリング

処理 [629](#)

取り扱い [116](#)

例 [117](#)

ネストされた COPY ステートメント [852](#), [888](#)

ネストされた IF ステートメント

コーディング [100](#)

望ましい EVALUATE ステートメント [100](#)

CONTINUE ステートメント [100](#)

NULL ブランチを伴う [99](#)

ネストされた区切り範囲ステートメント [21](#)

ネストされた組み込み関数 [64](#)

ネストされたプログラム

移動、制御の [614](#)

指針 [614](#)

説明 [614](#)

マップ [500](#), [511](#)

有効範囲、名前の [616](#)

呼び出し [614](#)

INITIAL 節の影響 [5](#)

ネストされたプログラムの統合 [843](#)

ネストされたプログラム・マップ

説明 [500](#)

例 [511](#)

ネスト・レベル

ステートメント [505](#), [537](#)

プログラム [505](#), [511](#), [537](#)

[ハ行]

バイト・オーダー・マークが生成されない [732](#)

バイト・ストリーム・ファイル

QSAM による処理 [198](#)

配列

COBOL の [39](#)

Java

宣言する [791](#)

取り扱い [792](#)

バインダー

c89 コマンド [314](#)

DLL の場合に推奨 [641](#)

DLL の場合に必要なおプション [641](#)

バインドする、オブジェクト指向アプリケーションを

例 [328](#)

JCL または TSO/E の使用 [327](#)

パス名

コピーブックの検索に使用 [317](#), [481](#)

パスワード

代替索引 [218](#)

例 [219](#)

VSAM ファイル [218](#)

パック 10 進数データ [422](#)

パック 10 進数データ項目

効率的な使用 [55](#), [839](#)

サイン表記 [59](#)

説明 [55](#)

同義語 [52](#)

バッチ・コンパイル

オプションの優先順位

概要 [304](#)

例 [305](#)

説明 [302](#)

バッチ・コンパイル (続き)

LANGUAGE オプション

例 [305](#)

バッファー

最適な使用 [11](#)

QSAM 用に取得 [197](#)

パフォーマンス

一貫性のあるデータ型 [840](#)

検証を伴う XML 文書の構文解析 [699](#)

コーディング [837](#)

コンパイラ・オプション

AFP [845](#)

ARCH [845](#)

ARITH [845](#)

AWO [845](#)

BLOCK0 [845](#)

DYNAM [846](#)

FASTSRT [846](#)

HGPR [846](#)

INITCHECK [846](#)

INLINE [846](#)

MAXPCF [846](#)

NUMCHECK [846](#)

NUMPROC [59](#), [847](#)

OPTIMIZE [843](#), [847](#)

PARMCHECK [847](#)

RENT [847](#)

RMODE [847](#)

SMARTBIN [847](#)

SQLCCSID [565](#)

SSRANGE [848](#)

TEST [848](#)

THREAD [454](#), [848](#)

TRUNC [454](#), [848](#)

TUNE [848](#)

コンパイラ・オプションの影響 [844](#)

最適化プログラム

概要 [843](#)

算術式 [840](#)

算術評価 [839](#)

指数 [840](#)

ストライプ拡張形式 QSAM データ・セット [197](#)

チューニング [837](#)

データの使用 [839](#)

テープ、QSAM [185](#)

テーブル探索

逐次探索の改善 [92](#)

二分と逐次の比較 [91](#)

テーブルのコーディング [841](#)

テーブルの処理 [842](#)

デバッグとの対比 [451](#)

バッファー・サイズの影響 [384](#)

プログラミング・スタイル [837](#)

ブロック化、QSAM ファイルの [184](#), [383](#)

変数添え字のデータ形式 [77](#)

呼び出し [611](#)

ライン外 PERFORM とインラインの比較 [108](#)

ワークシート [849](#)

AIXBLD ランタイム・オプション [850](#)

APPLY WRITE-ONLY 節 [11](#)

CBLPSHPOP に関する考慮事項 [554](#)

CBLPSHPOP ランタイム・オプション [554](#)

CICS

概要 [837](#)

パフォーマンス (続き)

CICS コーディング [850](#)

EVALUATE 内の WHEN 句の順序 [102](#)

FASTSRT [255](#)

FASTSRT によるソート [255](#)

IMS 環境 [573](#), [850](#)

OCCURS DEPENDING ON [841](#)

VSAM ファイル [225](#), [850](#)

パラメーター

呼び出し先プログラムの中での記述 [626](#)

ADEXIT [892](#)

INEXIT [887](#)

LIBEXIT [889](#)

MSGEXIT [894](#)

PRTEXIT [891](#)

z/OS でのメインプログラムからのアクセス

概要 [637](#)

例 [637](#)

z/OS ユニックス でのメインプログラムからのアクセス

概要 [584](#)

例 [584](#)

範囲区切りステートメント

説明 [20](#)

ネストされた [21](#)

範囲終了符号

暗黙の [21](#)

デバッグでの補助 [486](#)

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 [761](#)

非 OO COBOL/Java 相互運用可能アプリケーションのコンパイル、リンク、および実行

非 OO アプリケーションの構築と実行 [331](#)

z/OS UNIX での非 OO アプリケーションの準備 [332](#)

非 OO アプリケーションの構築と実行

例: COBOL/Java 相互運用可能アプリケーションの構築 [332](#)

比較条件 [104](#)

引数 BY CONTENT をユーザー定義関数に渡します [662](#)

引数

メインプログラムからの

z/OS でのアクセス [637](#)

z/OS ユニックス でのアクセス [584](#)

呼び出し側プログラムでの記述 [625](#)

BY VALUE で渡す [625](#)

OMITTED の指定 [627](#)

OMITTED 引数に関するテスト [627](#)

引数として渡すデータのグループ化 [628](#)

引数を省略するための OMITTED 句 [627](#)

ビッグ・エンディアン、リトル・エンディアンへの変換 [133](#)

日付操作

コンパイルの日付の検索 [130](#)

表

値のロード [79](#)

値の割り当て [80](#)

エレメント [73](#)

エレメントのサブストリングの参照 [118](#)

エレメントの参照 [76](#)

可変長

オーバーレイを防ぐ [90](#)

作成 [84](#)

初期化 [87](#)

ロードの例 [86](#)

行 [74](#)

表 (続き)

組み込み関数による処理 [94](#)
効率のよいコーディング [841](#), [842](#)
参照変更 [77](#)
指標、定義 [76](#)
指標による参照の例 [76](#)
初期化
 エレメントのすべての出現 [82](#)
 グループ・レベルの [82](#)
 それぞれの項目の個別の [81](#)
 INITIALIZE の使用 [79](#)
 PERFORM VARYING の使用 [109](#)

ストライド計算 [842](#)

説明 [39](#)

添え字、定義 [76](#)

添え字による参照の例 [75](#)

ソート

 概要 [94](#)

多次元 [74](#)

探索

 概要 [91](#)

 順次 [91](#)

 逐次 [91](#)

 バイナリー [92](#)

 パフォーマンス [91](#)

定義 [73](#)

同一エレメント仕様 [841](#)

動的ロード [79](#)

配列との比較 [39](#)

深さ [75](#)

ループ [109](#)

レコードの再定義 [81](#)

列 [73](#)

1 次元 [73](#)

2 次元 [74](#)

3 次元 [75](#)

OCCURS 節による定義 [73](#)

評価、データ項目の内容の

 組み込み関数 [126](#)

 クラス・テスト

 概要 [103](#)

 数値の [60](#)

 INSPECT ステートメント [120](#)

表現

 データ [60](#)

 符号 [59](#)

表示浮動小数点データ (USAGE DISPLAY) [53](#)

ファイル

 オプション

 QSAM [188](#)

 VSAM [212](#)

 オペレーティング・システムに対する定義 [9](#)

 外部 [634](#)

 概要 [172](#)

 可用性

 QSAM [188](#)

 VSAM [219](#)

 行順次、割り振り [238](#)

 使用法の説明 [10](#)

 説明 [12](#)

 属性 [194](#)

 データ・セット、同じ意味 [6](#)

 名前の変更 [10](#)

 パフォーマンスのソート

ファイル (続き)

 パフォーマンスのソート (続き)

 可変長ファイル [251](#)

 FASTSORT [255](#)

 ファイル定義レコードのストレージ [655](#)

 プログラム・ファイルを外部ファイルに関連付ける [6](#)
 プロセッシング

 行順次 [237](#)

 マルチスレッド化 [654](#)

 QSAM [177](#)

 VSAM [201](#)

 VSAMDB [229](#)

 マルチスレッド化処理

 シリアライゼーション [654](#)

 推奨使用パターン [655](#)

 推奨編成 [655](#)

 例 [656](#)

 利用不能

 QSAM [188](#)

 VSAM [219](#)

 CICS での制限 [6](#)

 COBOL コーディング

 概要 [173](#)

 入出力ステートメント [186](#), [209](#), [239](#)

 DATA DIVISION 記入項目 [178](#), [207](#), [238](#)

 ENVIRONMENT DIVISION 記入項目 [177](#), [203](#), [237](#)

 z/OS に対する識別 [191](#), [193](#), [219](#)

 ファイル・アクセス・モード

 索引付きファイル (KSDS) の場合 [207](#)

 順次 [207](#)

 順次ファイル (ESDS) の場合 [207](#)

 選択 [173](#)

 相対ファイル (RRDS) の場合 [207](#)

 動的 [207](#)

 パフォーマンスの考慮事項 [226](#)

 要約テーブル [204](#)

 ランダム [207](#)

 例 [207](#), [476](#)

 ファイル位置標識 (CRP) [211](#), [214](#)

 ファイル記述 (FD) 項目 [13](#)

 ファイル状況キー

 エラー処理 [487](#)

 エラー処理の場合の設定 [176](#)

 正常 OPEN かどうかの検査 [271](#), [272](#)

 入出力エラーの検査 [271](#)

 05 [211](#)

 35 [211](#)

 39 [211](#)

 VSAM 状況コードと一緒に使用 [273](#)

 VSAM、の重要性 [217](#)

 ファイル状況コード

 02 [215](#)

 30 [213](#)

 37 [186](#)

 39 [187](#), [194](#), [198](#)

 49 [216](#)

 90 [185](#), [190](#), [217](#)

 92 [217](#), [582](#)

 ファイルのオープン

 行順次 [240](#)

 マルチスレッド化シリアライゼーション [654](#)

 QSAM [187](#)

 VSAM

 概要 [211](#)

- ファイルのオープン (続き)
 - VSAM (続き)
 - 空の [212](#)
- ファイルの終わり (AT END 句) [269](#)
- ファイルの可用性
 - z/OS のもとでの QSAM ファイル [188](#)
 - z/OS のもとでの VSAM ファイル [219](#)
- ファイルのクローズ
 - 行順次 [241](#)
 - マルチスレッド化シリアライゼーション [654](#)
- QSAM
 - 概要 [190](#)
 - マルチスレッド化 [190](#)
- VSAM
 - 概要 [217](#)
 - マルチスレッド化 [217](#)
- ファイルのクローズ、自動的な
 - 行順次 [241](#)
 - QSAM [190](#)
 - VSAM [217](#)
- ファイルの割り振り [175](#)
- ファイル編成
 - 概要 [171](#)
 - 行順次 [237](#)
 - 索引付き [172](#), [204](#)
 - 順次 [171](#), [204](#)
 - 選択 [173](#)
 - 相対 [172](#)
 - 相対レコード [206](#)
 - ESDS、KSDS、RRDS の比較 [202](#)
 - QSAM [177](#)
 - VSAM [202](#)
- ファイル割り振り
 - 行順次 [238](#)
 - 説明 [175](#)
 - QSAM [191](#)
 - TSO のもとでの [287](#)
 - VSAM [222](#)
- ファクトリー・セクション、定義 [774](#)
- ファクトリー定義、コーディング [774](#)
- ファクトリー・データ
 - それをアクセス可能にする [775](#)
 - 定義 [743](#), [774](#)
 - private [775](#)
- ファクトリー・メソッド
 - 隠蔽 [776](#)
 - 定義 [743](#), [775](#)
 - プロシージャ・プログラムのラップに使用 [782](#)
 - 呼び出し [777](#)
- ファクトリー・メソッドの隠蔽 [776](#)
- フィードバック
 - 送付 [xxxiii](#)
- 深さ、テーブルの [75](#)
- 複合 OCCURS DEPENDING ON
 - 可変位置グループ [88](#)
 - 可変位置データ項目 [88](#)
 - 基本形式 [87](#)
 - 複合 ODO 項目 [88](#)
- 複数の通貨記号
 - 使用 [71](#)
 - 例 [72](#)
- 含まれているプログラムの統合 [843](#)
- 符号条件
 - 数値オペランドの符号のテスト [103](#)
- 物理ブロック [171](#)
- 物理レコード [14](#), [171](#)
- 不定形式レコード形式
 - 要求 [183](#)
 - レイアウト [184](#)
 - QSAM [199](#)
- 浮動小数点演算
 - 指数 [866](#)
 - 比較 [69](#)
 - 評価 [68](#)
 - 例の評価 [70](#)
- 浮動小数点データ
 - 外部 [53](#)
 - 固定小数点と浮動小数点との間の変換 [58](#)
 - 使用計画 [839](#)
 - 中間結果 [866](#)
 - 内部
 - 形式 [55](#)
 - パフォーマンスに関するヒント [840](#)
 - 変換と精度 [58](#)
- フラグおよびスイッチ [104](#)
- ブランチ、暗黙の [107](#)
- フローティングコメント標識 (*>) [1050](#)
- プログラム
 - 決定
 - スイッチおよびフラグ [104](#)
 - ループ [108](#)
 - EVALUATE ステートメント [99](#)
 - IF ステートメント [99](#)
 - PERFORM ステートメント [108](#)
 - 構造化 [3](#)
 - 再始動 [826](#)
 - 再入可能 [620](#)
 - サブプログラム [603](#)
 - シグニチャー情報バイト [513](#)
 - 初期化コード [522](#)
 - 診断 [504](#)
 - 制約 [837](#)
 - 属性コード [511](#)
 - 統計 [504](#)
 - ネスト・レベル [505](#), [537](#)
 - メイン [603](#)
 - cob2 によるコンパイルおよびリンク
 - 概要 [313](#)
 - 例 [315](#)
 - DLL [314](#)
 - z/OS のもとでのコンパイル [279](#)
 - z/OS ユニックス の開発 [579](#)
 - z/OS ユニックス のもとでのコンパイル [311](#)
- プログラム終了
 - ステートメント [604](#)
 - メインおよびサブプログラムで取られるアクション [604](#)
- プログラム情報
 - リスト例 [525](#)
- プログラム処理テーブル [548](#)
- プログラムのコンパイルおよびデバッグ
 - 非 OO COBOL/Java 相互運用可能アプリケーションのコンパイル、リンク、および実行 [331](#)
- プログラムのドキュメンテーション [5](#)
- プログラム・プロローグ域
 - リスト例 [526](#)
- プログラム名
 - 指定 [3](#)
 - 相互参照 [533](#)

プログラム名 (続き)
大/小文字の処理 [432](#)
特定の接頭部の使用を回避 [3](#)
プロシージャおよびデータ名相互参照の記述 [494](#)
プロシージャ統合 [844](#)
プロシージャ・ポインター・データ項目
入り口点の入り口アドレス [617](#)
定義 [617](#)
呼び出し可能サービスへのパラメーターの受け渡し [617](#)
C/C++ の呼び出し [619](#)
DLL での [645](#)
JNI サービスの呼び出し [619](#)
SET プロシージャ・ポインター [617](#)
プロセス
定義 [652](#)
プロセッシング
チェーン・リスト
概要 [629](#)
例 [630](#)
テーブル
指標付けを使用した例 [83](#)
添え字付けを使用した例 [82](#)
ブロック化、QSAM ファイルの
BLOCK CONTAINS 節の使用 [184](#)
BLOCK0 の使用 [383](#)
ブロック化、レコードの [184](#)
ブロック化因数の定義 [179](#)
ブロック・サイズ
コンパイラー・データ・セット [294](#)
システム決定の
コンパイラー・データ・セット [294](#)
QSAM ファイル [185, 383](#)
ASCII ファイル [199](#)
QSAM ファイル
可変長 [180](#)
固定長 [179](#)
レコード・レイアウト [181](#)
DCB の使用 [192](#)
文、定義 [19](#)
分離型の CICS 変換プログラム
コンパイラー・オプション [549, 552](#)
使用 [551](#)
制約事項 [551](#)
分離符号
移植性 [50](#)
印刷 [50](#)
行順次ファイル用 [241](#)
符号付き国別 10 進数に必要 [50](#)
ページ
コントロール [189](#)
深さ [14](#)
ベース・ロケーター [506, 508](#)
ベース・ロケーター・テーブル [529](#)
変換、COBOL データから XML への
概要 [727](#)
例 [733](#)
変更する
ソース・リストのタイトル [5](#)
ファイル名 [10](#)
文字から数値への [123](#)
変数
参照修飾子としての [118](#)
定義 [23](#)
変数、環境

変数、環境 (続き)
設定およびアクセスの例 [582](#)
ライブラリー名 [481](#)
ランタイム [581](#)
ポインター・データ項目
アドレスの受け渡し [629](#)
アドレスの増分 [630](#)
説明 [39](#)
チェーン・リストの処理 [629](#)
チェーン・リストの処理に使用 [630](#)
NULL 値 [630](#)
保護、VSAM ファイルの [218](#)
保持、ソートでのオリジナル・シーケンスの [254](#)
本製品のアクセシビリティ機能 [1026](#)

[マ行]

マージ
完了コード [254](#)
キー
概要 [244](#)
定義 [252](#)
基準 [252](#)
作業ファイル
説明 [245](#)
終了 [255](#)
使用ストレージ [260](#)
診断メッセージ [255](#)
制御ステートメントの受け渡し [259](#)
正常終了の判別 [254](#)
制約事項 [243](#)
説明 [243](#)
代替照合シーケンス [254](#)
ファイルの説明 [245](#)
プロセス [244](#)
z/OS データ・セット定義用の DD ステートメント [250](#)
z/OS のもとで必要なデータ・セット [250](#)
マッピング、DATA DIVISION 項目の [500](#)
マルチスレッド化
概要 [651](#)
言語間通信 [657](#)
再帰 [653](#)
再帰的要件 [657](#)
再入可能性 [657](#)
再入可能性要件 [657](#)
事前初期設定 [653](#)
制御転送 [653](#)
制約 [656](#)
ソートおよびマージの制限 [243](#)
データ・セクションの選択
OO クライアントにおいて [761](#)
入出力エラー宣言 [270](#)
ネストされたプログラム [657](#)
非同期シグナル [657](#)
ファイル入出力のコーディング
シリアライゼーション [654](#)
推奨使用パターン [655](#)
推奨編成 [655](#)
例 [656](#)
古いコンパイラー [657](#)
プログラムの終了 [653](#)
用語 [652](#)
ランタイムの制限 [657](#)
リソースへのアクセスの同期化 [656](#)

マルチスレッド化 (続き)

- AMODE 設定 [657](#)
- COBOL プログラム [651](#)
- COBOL プログラムの準備 [651](#)
- EXIT PROGRAM ステートメント [604](#)
- GOBACK ステートメント [605](#)
- IGZEOPT [657](#)
- IGZETUN [657](#)
- PL/I タスクとの [657](#)
- QSAM ファイルのクローズ [190](#)
- STOP RUN ステートメント [604](#)
- THREAD コンパイラー・オプション
 - いつ選択するか [653](#)
 - 制限 [453](#)
- UPSI スイッチ [657](#)
- VSAM ファイルのクローズ [217](#)

- マルチスレッド化での UPSI スイッチ [657](#)
- マルチスレッド化での非同期シグナル [657](#)
- マルチスレッド環境での実行 [453](#)
- マルチスレッドでのファイルのシリアライゼーション [654](#)
- 矛盾するコンパイラー・オプション [376](#)
- 無制限グループ
 - プロセッシング [95](#)

明示範囲終了符号 [21](#)

命名

- ファイル [9](#)
- programs [3](#)

命令ステートメントのリスト [19](#)

メインプログラム

- サブプログラム [603](#)
- 動的呼び出し [607](#)
- z/OS でのパラメーター・リストへのアクセス
 - 概要 [637](#)
 - 例 [637](#)
- z/OS ユニックス でのパラメーター・リストへのアクセス
 - 概要 [584](#)
 - 例 [584](#)

メソッド

- インスタンス [751](#), [772](#)
- から値を戻す [754](#)
- コンストラクター [775](#)
- シグニチャー [752](#)
- スーパークラスの呼び出し [766](#)
- 多重定義 [755](#)
- ファクトリー [775](#)
- ファクトリーの隠蔽 [776](#)
- 呼び出し [763](#), [777](#)
- 渡された引数の取得 [754](#)
- Java アクセス制御 [789](#)
- overriding [755](#), [776](#)

メッセージ

- コンパイラー
 - カスタマイズ [895](#)
 - 作成する重大度レベルの判別 [402](#)
 - 重大度レベル [308](#), [896](#)
 - ソース・リストへの組み込み [492](#)
 - 端末への送信 [296](#)
 - フラグを立てる重大度の選択 [492](#)
 - リストの生成 [307](#)
- コンパイラーの指示 [307](#)
- 出口モジュールからの [903](#)
- SYSTEM への送信 [448](#)

メッセージ処理、言語環境プログラム 呼び出し可能サービス [854](#)

メモリー・マップ

- DSA [512](#)

文字セット、定義 [132](#)

文字の逆順 [123](#)

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とき [632](#)

- 言語環境プログラム・サービスからのフィードバック・コード [856](#)
- コンパイラー

- 概要 [308](#)

- メッセージ・カスタマイズの影響 [897](#)

- 最も高い重大度に依存 [309](#)

CICS ECI からの [549](#)

Db2 SQL ステートメントからの [561](#)

RETURN-CODE 特殊レジスター [632](#), [856](#)

VSAM ファイル

- 説明 [273](#)

- 例 [273](#)

- RLS モード [225](#)

XML 構文解析の [709](#), [872](#)

[ヤ行]

有効範囲、名前の

- グローバル [616](#)

- ローカル [616](#)

ユーザー定義関数

- 外部名の指定 [661](#)

- 構造化 [659](#)

- コーディング [659](#)

- コンパイル [660](#)

- 使用 [659](#), [665](#)

- 呼び出し [660](#)

- リンク・エディット [661](#)

- CONTENT による引数の引き渡し [662](#)

ユーザー定義関数のコーディング [659](#)

ユーザー定義関数のコンパイル [660](#)

ユーザー定義関数の作成 [659](#)

ユーザー定義関数の使用 [659](#), [665](#)

ユーザー定義関数の呼び出し [660](#)

ユーザー定義関数のリンク・エディット [661](#)

ユーザー定義の条件 [103](#)

ユーザー出口作業域 [886](#)

ユーザー出口ルーチン

- DFSORT [262](#)

優先順位

- コピーブック探索順序 [312](#)

- コンパイラー・オプション

- バッチで [304](#)

- SYSOPTF データ・セットで [295](#), [429](#)

- z/OS のもとでの [299](#)

- z/OS ユニックス のもとでの [313](#)

- 算術演算子 [63](#), [860](#)

- CICS オプション [550](#)

優先順位番号、セグメンテーションの [844](#)

優先符号 [59](#)

ユーロ通貨記号 [71](#)

用語

- VSAM [201](#)

- VSAMDB [229](#)

用語集 [1033](#)

呼び出し

- インスタンス・メソッド [763](#)
- オーバーフロー条件 [275](#)
- オブジェクト指向プログラムとの間の [617](#)
- 言語環境プログラム 呼び出し可能サービスへ [855](#)
- 言語環境プログラム 呼び出し可能サービス [855](#)
- 言語間の [603](#)
- 再帰的 [617](#)
- 静的
 - 行う [606](#)
 - 動的呼び出しでの [611](#)
 - パフォーマンス [611](#)
 - 例 [612](#)
- データの受け渡し [623](#)
- 動的
 - 行う [607](#)
 - 静的呼び出しでの [611](#)
 - 制約事項 [607](#)
 - パフォーマンス [611](#)
 - 例 [612](#)
- パラメーターの受け取り [626](#)
- 引数の受け渡し [625](#)
- ファクトリーまたは静的メソッド [777](#)
- 例外条件 [275](#)
- 24 ビット・プログラムの AMODE 切り替え [609](#)
- 31 ビット・アドレッシング・モード [609](#)
- CICS の制約事項 [547](#)
- COBOL プログラム相互間の [603](#), [605](#)
- COBOL プログラムと非 COBOL プログラムの間での [603](#)
- JNI サービスへの [785](#)
- LINKAGE SECTION [627](#)
- OMITTED 引数 [627](#)
- z/OS プログラムのもとの COBOL [637](#)
- z/OS ユニックス プログラムのもとの COBOL [579](#)
- 読み取り、VSAM ファイルからのレコードの
 - 順次 [214](#)
 - 動的 [214](#)
 - ランダム [214](#)
- 読み取り、レコードの
 - 行順次ファイルから [240](#)
 - ブロック・サイズ [184](#)
- 予約語テーブル、CICS 代替
 - 概要 [553](#)
 - WORD による指定 [462](#)

[ラ行]

- ラージ・ブロック・インターフェース (LBI) [185](#)
- ライブラリー
 - 定義 [296](#)
 - ディレクトリー記入項目 [289](#)
 - パスの指定 [481](#)
 - BASIS [296](#)
 - COPY [296](#)
- ライブラリー名
 - 指定されなかった場合の代替 [317](#)
 - 使用されない場合 [888](#)
 - データ・セット名への相互参照 [533](#)
- ライン外の PERFORM [107](#)
- ラッパー、定義 [782](#)
- ラッピング、プロシージャー指向プログラムの [782](#)
- 乱数の生成 [65](#)
- ランタイム
 - オプション [468](#)

ランタイム・オプション

- 85 COBOL 標準 準拠 [376](#)
- AIXBLD [850](#)
- ALL31 [609](#)
- AMODE3164 [598](#)
- CBLOPTS [637](#)
- CBLPSHPOP [554](#)
- DEBUG [488](#)
- ENVAR [327](#)
- MSGFILE [431](#)
- POSIX
 - オブジェクト指向アプリケーションでの使用 [327](#)
 - DLL 探索順序 [644](#)
- TRAP
 - 行順次ファイルのクローズ [241](#)
 - ON SIZE ERROR [266](#)
 - QSAM のファイルのクローズ [190](#)
 - VSAM のファイルのクローズ [217](#)
- XPLINK
 - 設定 [329](#)
 - デフォルトとしては推奨されない [329](#)
- z/OS のもとでの指定 [637](#)
- z/OS ユニックス のもとでの指定 [579](#)
- リスト
 - 組み込みエラー・メッセージ [492](#)
 - 短縮リストの生成 [500](#)
 - データおよびプロシージャー名相互参照 [494](#)
 - テキスト名相互参照 [494](#)
 - テキスト名のソート済み相互参照 [533](#)
 - プログラム名のソート済み相互参照 [533](#)
 - ユーザー提供の行番号 [501](#)
 - MAP 出力で使用される用語 [509](#)
 - PROCEDURE DIVISION のアセンブラー拡張 [512](#)
- リスト出力 [500](#)
- リストのヘッダー [5](#)
- リソース・リスト [1077](#)
- リテラル
 - 英数字
 - 説明 [25](#)
 - DBCS の内容での [169](#)
 - 使用 [25](#)
 - 数字 [26](#)
 - 定義 [25](#)
 - 16 進数
 - 使用 [134](#)
 - DBCS
 - 最大長 [169](#)
 - 使用 [169](#)
 - 説明 [26](#)
 - national
 - 使用 [134](#)
 - 説明 [26](#)
- リトル・エンディアン、ビッグ・エンディアンへの変換 [133](#)
- 理由コード、XML 構文解析の [709](#), [872](#)
- 利用不能ファイル
 - QSAM [188](#)
 - VSAM [219](#)
- リンク・ステップ用 c89 コマンド [314](#)
- リンクする、オブジェクト指向アプリケーションを
 - cob2 コマンド [322](#)
 - JCL または TSO/E の使用
 - 概要 [327](#)
 - 例 [328](#)
 - z/OS ユニックス のもとでの

リンクする、オブジェクト指向アプリケーションを(続き)
z/OS ユニックスのもとでの(続き)

概要 [322](#)

例 [323](#)

リンク・リスト処理、例 [630](#)

ループ

コーディング [107](#)

条件 [109](#)

テーブル内 [109](#)

明示的に指定した回数だけ実行される [108](#)

do [108](#)

例: COBPROD アプリケーション

例: COBPROD アプリケーション [338](#)

例外条件

CALL [275](#)

XML GENERATE [732](#)

XML PARSE [711](#)

例外処理

Java との [786](#)

XML GENERATE での [732](#)

XML PARSE での [709](#)

レコード

形式

可変長 QSAM [180](#), [181](#)

可変長 VSAM [208](#)

形式 D [180](#), [181](#), [199](#)

形式 F [179](#), [199](#)

形式 S [181-183](#)

形式 U [183](#), [184](#), [199](#)

形式 V [180](#), [181](#), [199](#)

固定長 QSAM [179](#)

固定長 VSAM [208](#)

スパン [181-183](#)

未定義 [183](#), [184](#)

QSAM ASCII テープ [199](#)

順序への編成の影響 [171](#)

説明 [12](#)

レジスター、EXIT コンパイラー・オプションが使用する
[886](#)

列、テーブルの [73](#)

レベル 88 項目

条件式 [104](#)

スイッチおよびフラグ [104](#)

スイッチをオフに設定する例 [107](#)

スイッチをオンに設定する例 [106](#)

単一値のテストの例 [105](#)

複数値のテストの例 [105](#)

レベル番号 [506](#)

レポート COBOL ランタイム・メッセージ (IGZRPTMSG)
[472](#)

ローカル参照、グローバルへの変換 [767](#)

ローカル名 [616](#)

論理レコード

可変長形式

QSAM のレイアウト [181](#)

QSAM 用の要求 [180](#)

VSAM 用定義 [208](#)

固定長形式

QSAM 用の要求 [179](#)

VSAM 用定義 [208](#)

説明 [171](#)

QSAM、定義 [178](#)

[ワ行]

ワークスペース

ソート時の使用 [261](#)

割り込み [823](#)

[数字]

16 MB 境界

パフォーマンス・オプション [845](#)

CICS プログラム [546](#)

IMS プログラム [546](#)

16 進数字から変換 [125](#)

16 進数字に変換 [125](#)

16 進数リテラル

通貨符号として [71](#)

national

使用 [134](#)

説明 [26](#)

2 進数字から変換 [125](#)

2 進数字に変換 [125](#)

2 進数データ [422](#)

2 進数データ項目

一般的説明 [54](#)

効率的な使用 [54](#), [839](#)

中間結果 [864](#)

同義語 [52](#)

24 ビット・アドレッシング・モード [40](#)

31 ビット・アドレッシング・モード

動的呼び出し [609](#)

64 ビットのアドレッシング

サポートなし [40](#)

85 COBOL 標準

チェックポイント [824](#)

必要なコンパイラー・オプション [375](#)

必要なランタイム・オプション [376](#)

A

a.out ファイル、cob2 からの [318](#)

ACCEPT ステートメント

入力データの割り当て [35](#)

CICS のもとで [547](#)

stdin からの読み取り [35](#)

ADATA コンパイラー・オプション [378](#)

ADDRESS OF 特殊レジスター

CALL ステートメントでの使用 [624](#)

ADEXIT サブオプション、EXIT オプションの

構文 [400](#)

処理 [892](#)

ADMODE 属性

マルチスレッド化 [657](#)

adt サフィックス、cob2 での [318](#)

ADV コンパイラー・オプション [378](#)

AFP コンパイラー・オプション

パフォーマンスの考慮事項 [845](#)

マルチオプションの相互作用 [376](#)

AIXBLD ランタイム・オプション

パフォーマンスへの影響 [850](#)

ALL 添え字

関数引数としてのテーブル・エレメント [64](#)

テーブル・エレメントの反復処理 [94](#)

例 [95](#)

ALL31 ランタイム・オプション
マルチオプションの相互作用 [41](#)
AMODE 切り替えの OFF [609](#)
ALLOCATE コマンド (TSO)
コンパイラー・データ・セット [287](#)
z/OS ユニックスファイル [287](#)
ALPHABET 節による照合シーケンスの設定 [7](#)
ALTERNATE RECORD KEY 節
代替索引の識別 [221](#)
KSDS ファイルの代替キーの識別 [205](#)
AMODE
および DLL [645](#)
切り替え
概要 [609](#)
例 [609](#)
ALL31(OFF) [609](#)
説明 [40](#)
EXIT モジュール [886](#)
AMODE 31/64
考慮事項と制約事項 [598](#)
AMODE 64
アプリケーション [367](#), [369](#)
考慮事項 [590](#), [593-596](#)
コンパイラー・オプション [588](#)
コンパイラー限界値 [588](#)
プログラムの開発 [587](#)
CALL ステートメント [588](#)
DLL に関する考慮事項 [368](#)
AMODE 64 プログラム
POINTER データ項目 [587](#)
AMODE 64 プログラムのコンパイルおよびバインド
z/OS ユニックスのもとでの [368](#)
AMODE 64z/OS ユニックスのもとでの
コンパイルおよびバインド [368](#)
AMODE COBOL/JAVA
考慮事項と制約事項 [596](#)
AMODE3164 [598](#)
AMP パラメーター [223](#)
ANNUITY 組み込み関数 [68](#)
API、UNIX、および POSIX
呼び出し [582](#)
APOST コンパイラー・オプション [379](#)
APPLY WRITE-ONLY 節 [11](#)
ARCH コンパイラー・オプション
パフォーマンスの考慮事項 [845](#)
ARITH コンパイラー・オプション
説明 [381](#)
パフォーマンスの考慮事項 [845](#)
ASCII
アルファベット、QSAM [198](#)
ジョブ制御言語 (JCL) (job control language (JCL)) [199](#)
テープ・ファイル、QSAM [198](#)
レコード形式、QSAM [199](#)
EBCDIC への変換 [124](#)
XML 文書でサポートされるコード・ページ [704](#)
ASCII ファイル
CODE-SET 節 [14](#)
DCB 内の OPTCD= パラメーター [14](#)
ASSIGN 節
DD 名に対応する [9](#)
QSAM ファイル [177](#)
AT END (ファイルの終わり) 句 [269](#)
ATTACH マクロ [289](#)
AWO コンパイラー・オプション

AWO コンパイラー・オプション (続き)
説明 [382](#)
パフォーマンスの考慮事項 [845](#)
APPLY-WRITE ONLY 節のパフォーマンス [11](#)

B

Base クラス
java.lang.Object に相当 [748](#)
java.lang.Object のために使用 [748](#)
BASIS ステートメント [481](#)
BIT-OF 組み込み関数 [125](#)
BIT-TO-CHAR 組み込み関数 [125](#)
BLANK WHEN ZERO 節
数字編集データを含んだ例 [51](#)
数値データ用にコーディングされる [134](#)
BLOCK CONTAINS 節
FILE SECTION 記入項目 [13](#)
QSAM ファイル [179](#), [184](#), [383](#)
VSAM ファイルでは無意味 [207](#)
BLOCK0 コンパイラー・オプション
説明 [383](#)
パフォーマンスの考慮事項 [845](#)
BPXBATCH ユーティリティ
実行する、オブジェクト指向アプリケーションを [327](#)
z/OS ユニックス プログラムの呼び出し [580](#)
BUFSIZE コンパイラー・オプション [384](#)
BY CONTENT [623](#)
BY REFERENCE [623](#)
BY VALUE
制約事項 [625](#)
説明 [623](#)
有効なデータ型 [625](#)
BYTE-LENGTH 組み込み関数
国別データを伴う [129](#)
使用 [126](#)

C

C/C++ プログラム
マルチスレッド化 [657](#)
COBOL DLL を使用した [648](#)
CALL ID
常時動的 [609](#)
動的呼び出し [607](#)
DLL から行う [643](#)
NODLL の場合 [607](#)
NODYNAM の場合 [611](#)
CALL literal
静的呼び出し [606](#)
動的呼び出し [607](#)
DYNAM の場合 [607](#)
NODLL の場合 [606](#), [607](#)
NODYNAM の場合 [606](#), [611](#)
CALL コマンド (TSO) [287](#)
CALL ステートメント
エラー処理に関する [275](#)
オーバーフロー条件 [275](#)
関数ポインター [619](#)
言語環境プログラム 呼び出し可能サービス [855](#)
その中のプログラム名の処理 [432](#)
代替入り口点への [620](#)
例外条件 [275](#)

CALL ステートメント (続き)

AMODE 64 サポート付き [588](#)

AMODE の処理 [609](#)

BY CONTENT [623](#)

BY REFERENCE [623](#)

BY VALUE

制約事項 [625](#)

説明 [623](#)

CANCEL を使用した [608](#)

CICS の制約事項 [547](#)

DYNAM の場合 [398](#)

EXIT オプションのレジスターへの影響 [886](#)

ON EXCEPTION を指定した [275](#)

ON OVERFLOW を指定した [20](#), [275](#)

RETURNING [633](#)

USING [625](#)

CANCEL ステートメント

サブプログラムの場合 [608](#)

その中でのプログラム名の処理 [432](#)

動的 CALL を使用した [608](#)

DLL リンケージでは使用できない [645](#)

cbl サフィックス、cob2 での [318](#)

CBL ステートメント

概要 [481](#)

コンパイラー・オプションの指定 [300](#)

CBLPSHPOP ランタイム・オプション [554](#)

CBLQDA

DEBUG [369](#)

CBLQDA ランタイム・オプション [188](#)

CCSID

構文解析対象の XML 文書 [690](#)

定義 [132](#)

CODEPAGE オプションでの指定 [385](#)

Db2 スtring・データ [563](#)

EBCDIC マルチバイト CCSID [387](#)

PARSE ステートメント [690](#)

XML 文書での矛盾 [712](#)

XML 文書の [704](#)

CHAR 組み込み関数の例 [127](#)

CHKPT キーワード [261](#)

CICS

組み込みの変換プログラム

概要 [551](#)

コンパイラー・オプション [549](#)

ネストされたプログラムの呼び出し [548](#)

利点 [551](#)

このもとでの言語間通信 [548](#)

コマンドおよび PROCEDURE DIVISION [545](#)

コマンド・レベル・インターフェース [545](#)

システム日付の取得 [547](#)

実行するプログラムのコーディング

概要 [545](#)

制約事項 [545](#)

入出力 [546](#)

呼び出し [547](#)

DISPLAY ステートメント [546](#)

SORT ステートメント [553](#)

制約事項

オブジェクト指向プログラム [545](#), [743](#)

ソート [262](#)

ファイル [6](#)

分離型の変換プログラム [551](#)

16 MB 境界 [546](#)

FILE を使用した検証を伴う構文解析 [699](#)

CICS (続き)

制約事項 (続き)

OUTDD コンパイラー・オプション [431](#)

そのもとでのソート

概要 [262](#)

制約事項 [262](#)

予約語テーブルの変更 [553](#)

代替予約語テーブル [553](#)

ネストされたプログラムの呼び出し [548](#)

パフォーマンス

概要 [837](#)

パフォーマンスの考慮事項 [554](#), [850](#)

プログラムの開発 [545](#)

分離型の変換プログラム

コンパイラー・オプション [552](#)

使用 [551](#)

制約事項 [551](#)

ネストされたプログラムの呼び出し [548](#)

マクロ・レベル・インターフェース [545](#)

マルチスレッド化環境において [656](#)

CICS HANDLE

例 [554](#)

LABEL 値 [554](#)

CICS オプションを指定したコンパイル [549](#)

DFHCOMMAREA パラメーター

ネストされたプログラムの呼び出し [548](#)

別々にコンパイルされたプログラムの呼び出し [547](#)

DFHEIBLK パラメーター

ネストされたプログラムの呼び出し [548](#)

別々にコンパイルされたプログラムの呼び出し [547](#)

ECI 呼び出しと RETURN-CODE 特殊レジスター [549](#)

EXIT コンパイラー・オプション [904](#)

NODYNAM コンパイラー・オプション [547](#)

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 [551](#)

サブオプションの指定 [384](#), [550](#)

使用 [549](#)

説明 [384](#)

マルチオプションの相互作用 [376](#)

CISZ (制御インターバル・サイズ)、パフォーマンスの考慮事項 [227](#), [850](#)

cj ビルド

コマンド・リファレンス [361](#)

CKPT キーワード [261](#)

CLASSPATH 環境変数

設定の例 [327](#)

説明 [581](#)

Java クラスの場所の指定 [324](#)

CLOSE ステートメント

行順次ファイル [239](#)

QSAM [187](#)

VSAM [209](#)

cob2 コマンド

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 [321](#)

オブジェクト指向アプリケーションをリンクする場合 [322](#)

オプションおよび構文 [315](#)

これによるコンパイル

概要 [313](#)

例 [315](#)

説明 [315](#)

入出力 [318](#)

リンク

cob2 コマンド (続き)
リンク (続き)
概要 [313](#)
例 [315](#)
DLL 作成用 [314](#)
COBJVMINIOPTIONS 環境変数
説明 [581](#)
JVM オプションの指定 [325](#)
COBOL DLL プログラムの呼び出し [646](#)
COBOL 環境の事前初期設定
マルチスレッド化 [653](#)
COBOL クライアント
オブジェクト参照引き渡しの例 [765](#)
例 [777](#)
COBOL 専用
ランタイム・オプション指定 [369](#)
COBOL 用語 [23](#)
COBOL 呼び出し Java アプリケーション
構築と実行 [351](#)
COBOL ランタイム
オプション [470](#)
COBOL ランタイム オプション レポート オプションを無効にする
レポートを無効にする [475](#)
COBOL_INSTALL_DIR 環境変数 [312](#)
COBOL/Java 相互運用可能アプリケーションの構築と実行
AMODE の考慮事項 [361](#)
COBOL プログラムのリンク [338](#)
Java プログラムのビルド [338](#)
COBOL/Java 相互運用性
オブジェクト指向 COBOL の外部 [805](#), [807](#), [811](#)
既存のプログラムを Java から呼び出し可能にする [807](#)
トラブルシューティング [360](#)
COBOL からの静的 Java メソッドの呼び出し [811](#)
COBOL3 変換プログラム・オプション [552](#)
COBOL の
オブジェクト指向
実行 [324](#)
バインディング(binding) [327](#)
リンク [322](#)
IMS のもとでの [573](#)
JCL または TSO/E を使用したコンパイル [326](#)
z/OS ユニックス のもとでのコンパイル [321](#)
Java
アプリケーションの構造化 [782](#)
間の通信 [785](#)
互換 [330](#)
実行 [324](#), [327](#)
バインディング(binding) [327](#)
リンク [322](#)
IMS のもとでの [573](#)
JCL または TSO/E を使用したコンパイル [326](#)
z/OS ユニックス のもとでのコンパイル [321](#)
COBOL ランタイム
メッセージ (IGZUQMSG) [471](#)
COBOL ランタイムメッセージ
(IGZRPTMSG) [472](#)
COBOPT 環境変数 [311](#)
CODE-SET 節 [14](#)
CODEPAGE コンパイラ・オプション
影響を受けない項目 [386](#)
オーバーライドする操作 [386](#)
国別リテラルの場合 [148](#)
説明 [385](#)

CODEPAGE コンパイラ・オプション (続き)
DBCS コード・ページ [387](#)
COLLATING SEQUENCE 句
国別キーに適用されない [252](#)
PROGRAM COLLATING SEQUENCE 節のオーバーライド
[7](#), [254](#)
SORT または MERGE での使用 [254](#)
COMMON 属性 [4](#), [614](#)
COMP (COMPUTATIONAL) [54](#)
COMP-1 (COMPUTATIONAL-1)
形式 [55](#)
パフォーマンスに関するヒント [840](#)
COMP-2 (COMPUTATIONAL-2)
形式 [55](#)
パフォーマンスに関するヒント [840](#)
COMP-3 (COMPUTATIONAL-3) [55](#)
COMP-4 (COMPUTATIONAL-4) [54](#)
COMP-5 (COMPUTATIONAL-5) [54](#)
COMPILE コンパイラ・オプション
構文エラーを見つけるための NOCOMPILE の使用 [490](#)
説明 [388](#)
COMPUTATIONAL (COMP) [54](#)
COMPUTATIONAL-1 (COMP-1)
形式 [55](#)
パフォーマンスに関するヒント [840](#)
COMPUTATIONAL-2 (COMP-2)
形式 [55](#)
パフォーマンスに関するヒント [840](#)
COMPUTATIONAL-3 (COMP-3)
説明 [55](#)
COMPUTATIONAL-4 (COMP-4) [54](#)
COMPUTATIONAL-5 (COMP-5) [54](#)
COMPUTE ステートメント
コーディングが容易 [62](#)
算術結果の割り当て [34](#)
CONDCOMP コンパイラオプション
説明 [388](#)
CONFIGURATION SECTION [6](#)
CONTENT-CHARACTERS XML イベント
セグメントの構文解析時 [702](#)
例 [724](#)
CONTINUE ステートメント [100](#)
CONTROL ステートメント [481](#)
CONVERTING 句 (INSPECT) の例 [122](#)
COPY ステートメント
説明 [481](#)
ネストされた [852](#), [888](#)
例 [853](#)
z/OS ユニックスの考慮事項 [482](#)
z/OS の考慮事項 [296](#)
COPYLOC コンパイラ・オプション [389](#)
COPYRIGHT コンパイラ・オプション [390](#)
COUNT IN 句
UNSTRING [113](#)
XML GENERATE [733](#)
CRP (ファイル位置標識) [211](#), [214](#)
CURRENCY コンパイラ・オプション [390](#)
CURRENT-DATE 組み込み関数
例 [67](#)
CICS のもとで [547](#)

D

D フォーマット・レコード

D フォーマット・レコード (続き)

要求 [180](#)

レイアウト [181](#)

DASD (直接アクセス・ストレージ・デバイス) [227](#)

DATA DIVISION

インスタンス・データ [750](#), [772](#)

インスタンス・メソッド [753](#)

行順次ファイルについての記入項目 [238](#)

クライアント [761](#)

グループ・レベルの USAGE NATIONAL 節 [142](#), [158](#)

グループ・レベルの USAGE 節 [25](#)

項目 [513](#)

項目のマッピング [418](#), [500](#)

コーディング [12](#)

シグニチャー情報バイト [513](#)

制限 [12](#)

制約事項 [12](#)

説明 [12](#)

ファクトリー・データ [774](#)

ファクトリー・メソッド [776](#)

リスト [500](#)

FD 記入項目 [12](#)

FILE SECTION [12](#)

GROUP-USAGE NATIONAL 節 [74](#)

LINKAGE SECTION [12](#), [17](#)

LOCAL-STORAGE SECTION [12](#)

OCCURS DEPENDING ON (ODO) 節 [84](#)

OCCURS 文節 [73](#)

QSAM ファイルについての記入項目 [178](#)

REDEFINES 節 [81](#)

USAGE IS INDEX 節 [78](#)

VSAM ファイルについての記入項目 [207](#)

WORKING-STORAGE SECTION [12](#)

DATA RECORDS 節 [14](#)

DATA コンパイラー・オプション

説明 [391](#)

データの受け渡し時 [42](#)

データ・ロケーションへの影響 [42](#)

パフォーマンスの考慮事項 [845](#)

マルチオプションの相互作用 [40](#), [376](#)

DATE-COMPILED 段落 [3](#)

DATE-OF-INTEGER 組み込み関数 [67](#)

Db2

コーディングに関する考慮事項 [557](#)

コプロセッサ

概要 [557](#)

推奨コンパイラー・オプション SQLCCSID [564](#)

ストリング・データの CCSID の決定 [563](#)

データベース要求モジュール (DBRM) [557](#), [562](#)

プリコンパイラーとの相違 [565](#)

SQL INCLUDE の使用 [559](#)

SQL コンパイラー・オプションで使用可能にする [561](#)

プリコンパイラー

コプロセッサとの相違 [565](#)

使用 [558](#)

推奨コンパイラー・オプション NOSQLCCSID [564](#)

ホスト変数用のコード・ページの指定 [559](#)

CICS または CAF の場合の NODYNAM コンパイラー・オプション [567](#)

SQL コンパイラー・オプション [561](#)

SQL ステートメント

概要 [557](#)

国別 10 進数データの使用 [560](#)

Db2 (続き)

SQL ステートメント (続き)

コーディング [558](#)

バイナリー・データの使用 [561](#)

文字データの使用 [559](#)

戻りコード [561](#)

CCSID の決定 [563](#)

SQL DECLARE [559](#)

SQL INCLUDE [559](#)

SQLCCSID コンパイラー・オプション [563](#)

TSO または IMS の場合の DYNAM コンパイラー・オプション [567](#)

Db2 プリコンパイラー

使用 [558](#)

DBCS コンパイラー・オプション

オブジェクト指向 COBOL のための [321](#), [326](#)

説明 [392](#)

マルチオプションの相互作用 [377](#)

Java とのインターオペラビリティのための [321](#), [326](#)

DBCS データ

エンコードおよびストレージ [148](#)

これを伴う MOVE ステートメント [32](#)

宣言する [168](#)

テスト [169](#)

比較する

国別と [150](#)

表記 [868](#)

変換

英数字との間の [867](#)

国別への、概要 [170](#)

IGZCD2A による英数字への [870](#)

リテラル

最大長 [169](#)

使用 [169](#)

説明 [26](#)

DBCS 比較 [104](#)

dbg サフィックス、cob2 での [318](#)

dbrm サフィックス、cob2 での [318](#)

DBRM データ・セット

説明 [557](#)

定義 [562](#)

DBRM に使用されるロケーションを指定するための -dbrmllib
cob2 オプション [316](#)

DBRMLIB DD DD ステートメント [557](#), [562](#)

DCB [186](#)

DD 制御ステートメント

行順次ファイルの割り振り [238](#)

ソート・データ・セットの定義 [250](#)

ファイルの定義 [9](#)

マージ・データ・セットの定義 [250](#)

AMP パラメーター [223](#)

ASCII テープ・ファイル [199](#)

DBRMLIB [562](#)

DCB はデータ・セット・ラベルを指定変更する [192](#)

JAVAERR [327](#)

JAVAIN [327](#)

JAVAOUT [327](#)

QSAM ファイルの作成 [191](#), [193](#)

RLS パラメーター [223](#)

SYSADATA [297](#)

SYSDEBUG [298](#)

SYSIN [294](#)

SYSJAVA [297](#)

SYSLIB [296](#)

DD 制御ステートメント (続き)
 SYSLIN [297](#)
 SYSMDECK [298](#)
 SYSOPTF [295](#)
 SYSPRINT [296](#)
 SYSPUNCH [297](#)

DD 名の定義 [9](#)

DECK コンパイラー・オプション [393](#)

DEFINE コンパイラー・オプション [393](#)

dek サフィックス、cob2 での [318](#)

DELETE ステートメント
 コンパイラー指示 [483](#)
 マルチスレッド化シリアライゼーション [654](#)
 VSAM、コーディング [209](#)

DEPENDING ON 節 [180, 208](#)

DFHCOMMAREA パラメーター
 ネストされた CICS プログラムの呼び出し [548](#)
 別個にコンパイルされた CICS プログラムの呼び出し [547](#)

DFHEIBLK パラメーター
 ネストされた CICS プログラムの呼び出し [548](#)
 別個にコンパイルされた CICS プログラムの呼び出し [547](#)

DFSORT
 データ・セットの定義 [250](#)
 RETURN ステートメントのエラー・メッセージ [249](#)

DIAGTRUNC コンパイラー・オプション [394](#)

DISPLAY (USAGE IS)
 エンコードおよびストレージ [148](#)
 外部 10 進数 [53](#)
 浮動小数点 [54](#)

DISPLAY ステートメント
 行送りの抑止 [37](#)
 システム論理出力装置での表示 [37](#)
 出力の送信 [430](#)
 データ値の表示 [36](#)
 デバッグでの使用 [486](#)
 CICS のもとで [546](#)
 OUTDD との相互作用 [37](#)
 STDOUT または STDERR への書き込み [37](#)

DISPLAY-1 (USAGE IS)
 エンコードおよびストレージ [148](#)

DISPLAY-OF 組み込み関数
 ギリシャ語データでの例 [144](#)
 使用 [143, 160](#)
 中国語データでの例 [167](#)
 UTF-8 データでの例 [162](#)
 XML 文書での [706](#)

DISPSIGN コンパイラー・オプション [395](#)

DLL
 考慮事項 [368](#)

DLL (ダイナミック・リンク・ライブラリーを参照) [639](#)

DLL igzjava.x
 バインディング
 オブジェクト指向アプリケーションの準備 [327](#)
 例 [328](#)
 リンク
 オブジェクト指向アプリケーションの準備 [322](#)
 例 [323](#)

DLL libjvm.x
 バインディング
 オブジェクト指向アプリケーションの準備 [327](#)
 例 [328](#)
 リンク

DLL libjvm.x (続き)
 リンク (続き)
 オブジェクト指向アプリケーションの準備 [322](#)
 例 [323](#)
 EBCDIC サービスで [796](#)

DLL コンパイラー・オプション
 オブジェクト指向 COBOL のための [321, 326](#)
 説明 [396](#)
 マルチオプションの相互作用 [376](#)
 Java とのインターオペラビリティのための [321, 326](#)

DO ループ [108](#)

do-until [109](#)

do-while [109](#)

DSA メモリー・マップ [512](#)

DUMP コンパイラー・オプション
 出力 [301](#)
 説明 [398](#)

DYNAM コンパイラー・オプション
 説明 [398](#)
 動的呼び出しでの [607](#)
 パフォーマンスの考慮事項 [846](#)
 マルチオプションの相互作用 [376](#)
 TSO または IMS、および Db2 の場合 [567](#)

E

E レベルのエラー・メッセージ [309, 493](#)

EBCDIC
 ASCII への変換 [124](#)
 DBCS でサポートされているマルチバイト CCSID [387](#)
 JNI サービス [795](#)
 XML 文書でサポートされるコード・ページ [704](#)

ECI 呼び出しと RETURN-CODE 特殊レジスター [549](#)

EGCS [1048](#)

EJECT ステートメント [483](#)

END-OF-INPUT XML イベント
 セグメントの構文解析時 [701](#)
 例 [724](#)

ENTER ステートメント [483](#)

ENTRY ステートメント
 その中でのプログラム名の処理 [432](#)
 代替入り口点の [617](#)

ENVAR ランタイム・オプション [327](#)

ENVIRONMENT DIVISION
 インスタンス・メソッド [752](#)
 行順次ファイルについての記入項目 [237](#)
 クライアント [760](#)
 クラス [748](#)
 項目、プログラム初期設定コード [513](#)
 サブクラス [771](#)
 シグニチャー情報バイト [513](#)
 照合シーケンスのコーディング [7](#)
 説明 [5](#)

CONFIGURATION SECTION [6](#)

INPUT-OUTPUT SECTION [6](#)
 QSAM ファイルについての記入項目 [177](#)
 VSAM ファイルについての記入項目 [203](#)

ERRMSG、エラー・メッセージのリストの生成 [307](#)

ESDS (入力順データ・セット)
 ファイル・アクセス・モード [207](#)
 編成 [204](#)

EVALUATE ステートメント
 いくつかの条件をテストする例 [103](#)
 ケース構造 [101](#)

EVALUATE ステートメント (続き)
構造化プログラミング [838](#)
コーディング [101](#)
ネストされた IF と対比 [102](#), [103](#)
パフォーマンス [102](#)
複数値のテストの例 [105](#), [106](#)
複数条件のテストに使用 [99](#)
複数の WHEN 句の例 [102](#)
THRU 句の例 [102](#)
EXCEPTION XML イベント [711](#)
EXCEPTION/ERROR 宣言
行順次エラー処理 [241](#)
説明 [270](#)
ファイル状況キー [272](#)
QSAM エラー処理 [190](#)
VSAM エラー処理 [218](#)
EXEC 制御ステートメントの RD パラメーター [826](#)
EXIT PROGRAM ステートメント
サブプログラムにおける [604](#)
マルチスレッド化 [604](#)
EXIT コンパイラー・オプション
使用 [399](#)
説明 [399](#)
ユーザー出口作業域 [886](#)
レジスターの使用 [886](#)
MSGEXIT サブオプション [893](#)
SQL および CICS ステートメントについての考慮事項
[904](#)
EXPORTALL コンパイラー・オプション
説明 [401](#)
マルチオプションの相互作用 [376](#)
DLL に関する考慮事項 [640](#)
EXTERNAL 節
データ項目の [633](#)
ファイルの共用 [13](#), [634](#)
ファイルの場合の例 [634](#)

F

F フォーマット・レコード
要求 [179](#)
レイアウト [179](#)
FACTORY 段落
ファクトリー・データ [774](#)
ファクトリー・メソッド [775](#)
FASTSRT コンパイラー・オプション
説明 [402](#)
ソート・パフォーマンスの向上 [255](#), [846](#)
通知メッセージ [256](#)
の要件
ソート入出力ファイル [256](#)
JCL [256](#)
QSAM [257](#)
VSAM [257](#)
FD (ファイル記述) 項目 [13](#)
FILE SECTION
説明 [12](#)
レコードの説明 [12](#)
BLOCK CONTAINS 節 [13](#)
CODE-SET 節 [14](#)
DATA RECORDS 節 [14](#)
EXTERNAL 節 [13](#)
FD 記入項目 [13](#)
GLOBAL 節 [13](#)

FILE SECTION (続き)
LABEL RECORDS 節 [14](#)
LINAGE 節 [14](#)
OMITTED [14](#)
RECORD CONTAINS 節 [14](#)
RECORD IS VARYING [14](#)
RECORDING MODE 節 [14](#)
VALUE OF [14](#)
FILE STATUS 節
行順次エラー処理 [241](#)
使用 [271](#)
説明 [176](#)
例 [275](#)
NOFASTSRT エラー処理 [258](#)
QSAM エラー処理 [190](#)
VSAM エラー処理 [218](#)
VSAM 状況コードを持つ [273](#)
FILE-CONTROL 段落
項目の例 [6](#)
FD 項目との関係 [9](#)
FIPS メッセージ
カテゴリ [896](#)
FLAGSTD コンパイラー・オプション [403](#)
FLAG コンパイラー・オプション
コンパイラー出力 [493](#)
使用 [492](#)
説明 [402](#)
FLAGSTD コンパイラー・オプション
マルチオプションの相互作用 [377](#)

G

GB 18030 データ
国別との間の変換 [167](#)
プロセッシング [167](#)
get メソッドおよび set メソッド [756](#)
GETMAIN のアドレスの保管 [886](#)
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 [605](#)
マルチスレッド化 [605](#)
メインプログラムにおける [605](#)
GROUP-USAGE NATIONAL 節
国別グループの初期化 [31](#)
国別グループの宣言の例 [24](#)
国別グループの定義 [145](#)
テーブルの定義 [74](#)
Java との通信 [790](#)

H

HEAP ランタイム・オプション
データ・ロケーションへの影響 [42](#)
マルチオプションの相互作用 [40](#)
HEX-OF 組み込み関数 [125](#)
HEX-TO-CHAR 組み込み関数 [125](#)
HGPR コンパイラー・オプション
パフォーマンスの考慮事項 [846](#)
マルチオプションの相互作用 [376](#)

I

I レベルのメッセージ [308](#), [493](#)

IDENTIFICATION DIVISION

エラー [3](#)

クライアント [759](#)

クラス [748](#)

コーディング [3](#)

サブクラス [771](#)

必要な段落 [3](#)

メソッド [752](#)

リストのヘッダーの例 [5](#)

CLASS-ID 段落 [748](#), [771](#)

DATE-COMPILED 段落 [3](#)

PROGRAM-ID 段落 [3](#)

TITLE ステートメント [5](#)

IF ステートメント

コーディング [99](#)

ネストされた [100](#)

複数条件の場合に、代わりに EVALUATE を使用 [100](#)

NULL ブランチを伴う [99](#)

IGZCA2D サービス・ルーチン [868](#)

IGZCD2A サービス・ルーチン [870](#)

igzcjava.x

バインディング

オブジェクト指向アプリケーションの準備 [327](#)

例 [328](#)

リンク

オブジェクト指向アプリケーションの準備 [322](#)

例 [323](#)

IGZEOPT モジュール

マルチスレッド化 [657](#)

IGZETUN モジュール

マルチスレッド化 [657](#)

IGZSRTCD データ・セット [259](#)

IGZUOPT および IGZQUOPT ランタイム オプション [476](#)

IGZUQMSG [471](#)

IMS

コプロセッサ

概要 [569](#)

コンパイルおよびリンク [573](#)

のもとでのプログラムのコーディング

概要 [569](#)

制約事項 [6](#), [569](#)

パフォーマンスの考慮事項 [850](#)

COBOL-Java インターオペラビリティ

データベースへのアクセス [576](#)

トランザクションの同期 [576](#)

メッセージ [576](#)

AIB の使用 [576](#)

COBOL からの Java メソッドの呼び出し [575](#)

EXEC SQL の制約事項 [576](#)

Java からの COBOL メソッドの呼び出し [574](#)

STOP RUN [575](#)

IMS のもとでの EXEC SQL の使用 [576](#)

SQLIMS コンパイラー・オプション [571](#)

SQLIMS ステートメント

文字データの使用 [570](#)

戻りコード [571](#)

SQLIMS INCLUDE [570](#)

IMS SQL

コプロセッサ [570](#)

INEXIT サブオプション、EXIT オプションの

構文 [400](#)

処理 [887](#)

INITCHECK コンパイラー・オプション

説明 [405](#)

INITCHECK コンパイラー・オプション (続き)

無効な COBOL データ [491](#)

INITIAL コンパイラー・オプション

説明 [407](#)

INITIAL 節

ネストされたプログラムへの影響 [5](#)

プログラムを初期状態に設定 [5](#)

メインプログラムへの影響 [605](#)

INITIAL 属性

代わりに動的呼び出し および CANCEL を使用 [608](#)

サブプログラムへの影響 [606](#), [607](#)

INITIALIZE ステートメント

国別グループ値のロード [31](#)

グループ値のロード [31](#)

テーブルの値のロード [79](#)

デバッグ用の使用 [488](#)

例 [28](#)

REPLACING 句 [79](#)

INLINE コンパイラー・オプション

説明 [407](#)

INPUT-OUTPUT SECTION [6](#)

INSERT ステートメント [483](#)

INSPECT ステートメント

使用 [120](#)

例 [121](#)

UTF-8 データでの使用を避ける [709](#)

INTDATE コンパイラー・オプション

カレンダー開始日への影響 [66](#)

説明 [408](#)

INTEGER 組み込み関数の例 [120](#)

INTEGER-OF-DATE 組み込み関数 [67](#)

INTEGER-PART 組み込み関数 [120](#)

Interactive System Productivity Facility (ISPF) [1010](#)

INVALID KEY 句

説明 [275](#)

例 [275](#)

INVDATA コンパイラー・オプション

説明 [408](#)

INVOKE ステートメント

オブジェクトの作成に使用 [767](#)

メソッドの呼び出しに使用 [763](#)

ON EXCEPTION を指定した [764](#), [777](#)

PROCEDURE DIVISION RETURNING で [632](#)

RETURNING 句 [766](#)

USING 句 [764](#)

ISAM データ・セット、VSAM KSDS データ・セットと類似

[201](#)

ISPF (Interactive System Productivity Facility) [1010](#)

J

J2EE クライアント

実行 [325](#)

例 [797](#)

Java

インターオペラビリティ [785](#)

オブジェクト配列 [791](#)

および COBOL

アプリケーションの構造化 [782](#)

間の通信 [785](#)

互換 [330](#)

実行 [324](#), [327](#)

バインディング(binding) [327](#)

リンク [322](#)

Java (続き)

および COBOL (続き)

JCL または TSO/E を使用したコンパイル [326](#)

z/OS ユニックス のもとでのコンパイル [321](#)

クラス型 [790](#)

グローバル参照

受け渡し [788](#)

オブジェクト [788](#)

管理 [787](#)

JNI サービス [789](#)

除外、

処理 [786](#)

例 [787](#)

catch [787](#)

throw [786](#)

ストリング

宣言する [791](#)

取り扱い [795](#)

ストリング配列 [791](#)

相互運用可能データ型、コーディング [790](#)

データ共用 [789](#)

長い配列 [791](#)

二重配列 [792](#)

バイト配列 [791](#)

配列

宣言する [791](#)

取り扱い [792](#)

例 [794](#)

配列クラス [789](#)

バッチ COBOL プログラムから呼び出す [800](#)

ブール配列 [791](#)

浮動配列 [792](#)

短い配列 [791](#)

メソッド

アクセス制御 [789](#)

文字配列 [791](#)

例

整数配列の処理 [794](#)

例外処理 [787](#)

J2EE クライアント [797](#)

ローカル参照

受け渡し [788](#)

オブジェクト [788](#)

解放 [788](#)

管理 [787](#)

削除 [788](#)

保管 [788](#)

マルチスレッド化ごと [788](#)

JNI サービス [789](#)

boolean 型 [790](#)

byte 型 [790](#)

char 型 [790](#)

COBOL での実行

JCL または TSO/E の使用 [327](#)

XPLINK リンケージ [329](#)

z/OS ユニックス のもとでの [324](#)

double 型 [790](#)

float 型 [790](#)

int 型 [790](#)

int 配列 [791](#)

jstring クラス [789](#)

long 型 [790](#)

short 型 [790](#)

Java 仮想マシン

Java 仮想マシン (続き)

オブジェクト参照 [787](#)

初期化 [324](#)

例外 [787](#)

Java との相互運用が可能なデータ型 [790](#)

java.lang.Object

Base として参照 [748](#)

javac コマンド

Java 6 以降用に再コンパイル [330](#)

Java クラス定義のコンパイル [321](#)

JAVAERR データ・セット [327](#)

JAVAIN データ・セット [327](#)

JAVAIOP コンパイラー・オプション

説明 [412](#)

JAVAOUT データ・セット [327](#)

JCL

オブジェクト指向アプリケーションで使用する

例 [328](#)

カタログ式プロシーチャー [280](#)

行順次ファイル用 [238](#)

コンパイル用 [279](#)

ソート用 [250](#)

チェックポイント/再始動の例 [829](#)

マージ用 [250](#)

ASCII テープ・ファイル [199](#)

FASTSORT の要件 [255](#)

QSAM ファイル用 [192](#)

VSAM データ・セット用 [222](#)

z/OS UNIX ファイルシステム でのコンパイル用 [282](#)

JCL のサンプル

COBOL 呼び出し Java [355](#)

JCL のサンプル [345](#)

JCL パラメーターへのアクセス

コーディング [1025](#)

CEE3PR2 [1025](#)

LINKAGE SECTION [1025](#)

JNI

オブジェクト参照の比較 [762](#)

クラス・オブジェクト参照の取得 [786](#)

構造環境

アドレス可能度の場合 [786](#)

サービスへアクセス [785](#)

使用した場合の制限 [786](#)

例外取り扱いサービス [786](#)

ローカル参照からグローバルへの変換 [767](#)

EBCDIC サービス [795](#)

Java ストリング・サービス [795](#)

Java 配列サービス [792](#)

Unicode サービス [795](#)

UTF-8 サービス [797](#)

JNI.cpy

コンパイル用 [322](#)

リスト [906](#)

JNINativeInterface の場合 [785](#)

JNIEnvPtr 特殊レジスター

JNI 呼び出し可能サービスのための使用 [786](#)

JNINativeInterface

構造環境 [785](#)

JNI.cpy [785](#)

JOB 制御ステートメントの RD パラメーター [826](#)

JSON null 値の処理 [678](#)

JSON null 値の生成 [685](#)

JSON 出力

生成

JSON 出力 (続き)
 生成 (続き)
 概要 [683](#)
JSON 出力の生成 [683](#)
JSON 入力
 構文解析
 概要 [673](#)
 例 [679](#)
JSON 入力の処理 [673](#)
JSON の生成
 概要 [683](#)
JSON パーサー
 概要 [673](#)
 例 [679](#)
JSON 文書
 構文解析
 概要 [673](#)
 説明 [673](#)
 ヌル値 [678](#)
 例 [679](#)
 生成
 概要 [683](#)
 ヌル値 [685](#)
JSON 文書の構文解析 [673](#)
JSON 例外コード
 構文解析 [882](#)
 生成 [881](#)
JSON-CODE 特殊レジスター
 構文解析の例外コード [882](#)
 生成の例外コード [881](#)
JSON-STATUS 特殊レジスター
 構文解析の非例外理由コード [882](#)
jstring Java クラス [789](#)
JZOS
 例 [800](#)
 Java Batch Launcher and Toolkit for z/OS [803](#)

K

KSDS (キー順データ・セット)
 ファイル・アクセス・モード [207](#)
 編成 [204](#)

L

LABEL RECORDS 節
 FILE SECTION 記入項目 [14](#)
LANGUAGE コンパイラー・オプション
 説明 [413](#)
LBI (ラージ・ブロック・インターフェース) [185](#)
LENGTH OF 特殊レジスター
 受け渡し [624](#)
 使用 [129](#)
LENGTH 組み込み関数
 可変長の結果 [128](#)
 国別データを伴う [129](#)
 使用 [126](#)
 例 [67](#), [129](#)
 LENGTH OF 特殊レジスターと比較 [129](#)
LIBEXIT サブオプション、EXIT オプションの
 構文 [400](#)
 処理 [888](#)
libjvm.x

libjvm.x (続き)
 バインディング
 オブジェクト指向アプリケーションの準備 [327](#)
 例 [328](#)
 リンク
 オブジェクト指向アプリケーションの準備 [322](#)
 例 [323](#)
 EBCDIC サービスで [796](#)
LIBPATH 環境変数
 設定の例 [327](#)
 説明 [581](#)
 COBOL クラスの場所の指定 [324](#)
library-name 環境変数 [312](#)
LINECOUNT コンパイラー・オプション [414](#)
LINK マクロ [289](#)
LINKAGE SECTION
 コーディング [627](#)
 再帰呼び出し [17](#)
 パラメーターを記述するための [626](#)
 THREAD オプションを指定した [17](#)
LIST コンパイラー・オプション
 外部シンボル・セクション [530](#)
 コンパイラー出力 [513](#), [522](#)
 出力で使用される記号 [510](#)
 出力の取得 [500](#)
 出力の読み取り [512](#)
 初期ヒープ・ストレージ・マップ・セクション [528](#)
 説明 [414](#)
 ソース・プログラムのアセンブラー・コード [512](#)
 タイム・スタンプおよびバージョン情報の例 [525](#)
 定数域セクション [529](#)
 特殊レジスター・テーブル [530](#)
 ベース・ロケーター・テーブル [529](#)
 マルチオプションの相互作用 [377](#)
 DSA メモリー・マップ [512](#), [531](#)
 MD5 シグニチャーの例 [524](#)
 OFFSET オプションとの対立 [500](#)
LOCAL-STORAGE SECTION
 クライアント [761](#)
 ロケーションの決定 [42](#)
 WORKING-STORAGE との比較
 概要 [14](#)
 例 [15](#)
 OO クライアント [761](#)
LOG 組み込み関数 [68](#)
LOWER-CASE 組み込み関数 [123](#)
LP コンパイラー・オプション
 マルチオプションの相互作用 [376](#), [377](#)
LSACHECK コンパイラ・オプション
 説明 [416](#)
lst サフィックス、cob2 での [318](#)

M

MAP コンパイラー・オプション
 組み込みマップ要約 [500](#)
 出力で使用される記号 [510](#)
 出力で使用される用語 [509](#)
 使用 [495](#), [500](#)
 説明 [418](#)
 データ項目と相対アドレス [301](#)
 ネストされたプログラム・マップ
 例 [511](#)
 例 [505](#), [511](#)

MAP 出力で使用する用語 [509](#)

MAX 組み込み関数

関数の例 [67](#)

使用 [127](#)

テーブル計算の例 [95](#)

MAXPCF コンパイラー・オプション [419](#)

MDECK コンパイラー・オプション

説明 [419](#)

MEAN 組み込み関数

テーブル計算の例 [95](#)

統計計算の例 [68](#)

MEDIAN 組み込み関数

テーブル計算の例 [95](#)

統計計算の例 [68](#)

MERGE ステートメント

概要 [243](#)

制約事項 [243](#)

説明 [251](#)

ASCENDING|DESCENDING KEY 句 [252](#)

COLLATING SEQUENCE 句 [7](#), [254](#)

GIVING 句 [251](#)

USING 句 [251](#)

METHOD-ID 段落 [752](#)

MIN 組み込み関数

使用 [127](#)

例 [120](#)

MOVE ステートメント

基本受信項目を伴う [32](#)

国別項目を伴う [32](#)

国別データへの変換 [142](#)

グループ移動と基本移動の対比 [33](#), [146](#)

グループ受信項目を伴う [33](#)

算術結果の割り当て [34](#)

送信項目および受信項目の長さに対する ODO の影響 [85](#)

CORRESPONDING [33](#)

UTF-8 データへの変換 [159](#)

MSGEXIT サブオプション、EXIT オプションの

構文 [401](#)

コンパイル戻りコードへの影響 [897](#)

処理 [893](#)

メッセージ重大度レベル [896](#)

ユーザー出口の例 [898](#)

MSGFILE ランタイム・オプション [431](#)

N

N 区切り文字、国別または DBCS リテラル用の [26](#)

NAME コンパイラー・オプション

使用 [3](#)

説明 [421](#)

NAMESPACE-DECLARATION XML イベント [697](#)

NATIONAL (USAGE IS)

外部 10 進数 [53](#)

浮動小数点 [54](#)

NATIONAL-OF 組み込み関数

ギリシャ語データでの例 [144](#)

使用 [143](#), [160](#)

中国語データでの例 [167](#)

UTF-8 データでの例 [162](#)

XML 文書での [706](#)

NEXT SENTENCE ステートメント [100](#)

NOCBLCARD 変換プログラム・オプション [552](#)

NOCOMPILE コンパイラー・オプション

構文エラーの検出に使用 [490](#)

NODLL コンパイラー・オプション

静的呼び出しでの [606](#)

動的呼び出しでの [607](#)

NODYNAM コンパイラー・オプション

ストアド・プロシージャの場合 [568](#)

静的および動的呼び出しでの [611](#)

静的呼び出しでの [606](#)

CICS のもとで [547](#)

CICS または CAF、および Db2 の場合 [567](#)

NOFASTSRT コンパイラー・オプション [258](#), [261](#)

NORENT コンパイラー・オプション

マルチオプションの相互作用 [377](#)

NOSQLCCSID コンパイラー・オプション、Db2 プリコンパイラーとの互換性のために推奨される [564](#)

NSYMBOL コンパイラー・オプション

国別データ項目の [134](#)

国別リテラルの場合 [134](#)

説明 [421](#)

マルチオプションの相互作用 [377](#)

DBCS リテラル用の [134](#)

N リテラルへの影響 [26](#)

NULL ブランチ [99](#)

NUMBER コンパイラー・オプション

説明 [422](#)

デバッグ用 [501](#)

NUMCHECK コンパイラー・オプション

マルチオプションの相互作用 [377](#)

無効な COBOL データ [491](#)

NUMCLS インストール・オプション、数値のクラス・テストへの影響 [60](#)

NUMPROC コンパイラー・オプション

説明 [426](#)

パフォーマンスの考慮事項 [847](#)

符号処理への影響 [59](#)

マルチオプションの相互作用 [377](#)

NUMCLS による影響 [60](#)

NUMVAL 組み込み関数

説明 [123](#)

NUMVAL-C 組み込み関数

説明 [123](#)

例 [67](#)

NX 区切り文字、国別リテラル用の [26](#)

O

o サフィックス、cob2 での [318](#)

OBJECT コンパイラー・オプション

説明 [427](#)

マルチオプションの相互作用 [377](#)

OBJECT 段落

インスタンス・データ [750](#), [772](#)

インスタンス・メソッド [751](#)

OBJECT-COMPUTER 段落 [6](#)

OCCURS DEPENDING ON (ODO) 節

可変長テーブル作成用 [84](#)

可変長レコード

QSAM [180](#)

VSAM [208](#)

最適化 [841](#)

単純 [84](#)

複合 [87](#)

ODO エレメントの初期化 [87](#)

ODO オブジェクト [84](#)

ODO サブジェクト [84](#)

OCCURS INDEXED BY 節による指標の作成 [77](#)
OCCURS 文節
指標作成用の INDEXED BY 句 [77](#)
多次元テーブルを作成するためにネストされた [74](#)
テーブル・エレメントの定義 [74](#)
テーブルの定義 [73](#)
レベル 01 項目では使用できない [74](#)
ASCENDING|DESCENDING KEY 句
テーブル・エレメントの順序の指定 [74](#)
二分探索に必要な [93](#)
例 [93](#)
ODO オブジェクト [84](#)
ODO サブジェクト [84](#)
OFFSET コンパイラー・オプション
出力 [536](#)
説明 [427](#)
マルチオプションの相互作用 [377](#)
OMITTED 節、FILE SECTION [14](#)
OMITTED パラメーター [856](#)
ON EXCEPTION 句
INVOKE ステートメント [764](#), [777](#)
OO アプリケーションの XPLINK リンケージ規約 [329](#)
OO アプリケーションの構造化 [782](#)
OPEN ステートメント
行順次ファイル [239](#)
ファイル状況キー [271](#)
ファイルの可用性 [187](#), [211](#), [240](#)
マルチスレッド化シリアライゼーション [654](#)
QSAM ファイル [186](#)
VSAM ファイル [209](#)
OPTFILE コンパイラー・オプション [428](#)
OPTIMIZE コンパイラー・オプション
説明 [429](#)
パフォーマンスの考慮事項 [843](#), [847](#)
パラメーター引き渡しの影響 [626](#)
ORD 組み込み関数の例 [127](#)
ORD-MAX 組み込み関数
使用 [127](#)
テーブル計算の例 [95](#)
ORD-MIN 組み込み関数 [127](#)
OUTDD コンパイラー・オプション
説明 [430](#)
割り振られない DD [37](#)
DISPLAY との対話 [37](#)
overriding
インスタンス・メソッド [755](#)
ファクトリー・メソッド [776](#)

P

PARMCHECK コンパイラー・オプション
無効な COBOL プログラム [491](#)
PASSWORD 節 [218](#)
PATH 環境変数
設定の例 [327](#)
説明 [581](#)
PERFORM ステートメント
インライン [107](#)
行外 [107](#)
指標を変更するための [78](#)
テーブル用の
指標付けを使用した例 [83](#)
添え字付けを使用した例 [82](#)
明示的に指定した回数だけ実行される [108](#)

PERFORM ステートメント (続き)
ループのコーディング [107](#)
TEST AFTER [109](#)
TEST BEFORE [109](#)
THRU [110](#)
TIMES [108](#)
UNTIL [109](#)
VARYING [109](#)
VARYING WITH TEST AFTER [109](#)
WITH TEST AFTER ... UNTIL [109](#)
WITH TEST BEFORE ... UNTIL [109](#)
PGMNAME コンパイラー・オプション
説明 [432](#)
COMPAT サブオプション [433](#)
LONGMIXED サブオプション [434](#)
LONGUPPER サブオプション [433](#)
PICTURE 節
国別データを表す N [134](#)
国別編集データ [134](#)
使用される記号の判別 [390](#)
数字編集データ [134](#)
数値データ [49](#)
ゼロ抑制用の Z [51](#)
内部浮動小数点に使用できない [50](#)
非互換データ [60](#)
PL/I タスク
COBOL との [657](#)
POSIX ランタイム・オプション [657](#)
POSIX
スレッド [657](#)
API の呼び出し [582](#)
POSIX ランタイム・オプション
オブジェクト指向アプリケーションでの使用 [327](#)
DLL 探索順序への影響 [644](#)
PRESENT-VALUE 組み込み関数 [67](#)
PROCEDURE DIVISION
インスタンス・メソッド [754](#)
クライアント [759](#)
サブプログラムでの [628](#)
シグニチャー情報バイト [513](#)
ステートメント
コンパイラー指示 [20](#)
条件 [19](#)
範囲区切り [20](#)
命令ステートメント [19](#)
説明 [18](#)
存在する [513](#)
追加情報 [513](#)
用語 [18](#)
RETURNING
使用 [633](#)
パラメーターの戻し [18](#)
USING
パラメーターの受け取り [18](#), [626](#)
BY VALUE [628](#)
PROCESS (CBL) ステートメント
概要 [484](#)
コンパイラー・オプションの指定 [300](#)
パッチ・コンパイル [304](#)
矛盾するオプション [376](#)
優先順位
バッチで [304](#)
z/OS のもとでの [299](#)
z/OS ユニックス のもとでの [313](#)

PROGRAM COLLATING SEQUENCE 節
国別または DBCS オペランドに影響を与えない [7](#)
照合シーケンスの設定 [7](#)
デフォルト照合シーケンスのオーバーライド [254](#)
COLLATING SEQUENCE 句によるオーバーライド [7](#)
PROGRAM-ID 段落
コーディング [3](#)
COMMON 属性 [4](#)
INITIAL 節 [5](#)
PRTEXT サブオプション、EXIT オプションの
構文 [400](#)
処理 [891](#)

Q

QSAM ファイル
オープン [187](#)
クローズ [190](#)
検索 [193](#)
再オープン防止のためのクローズ [187](#)
ストライプ拡張形式 [197](#)
属性 [194](#)
テープのパフォーマンス [185](#)
入出力エラー処理 [190](#), [267](#)
入出力ステートメント [186](#)
バッファの入手 [197](#)
パフォーマンスを向上させるブロック化 [184](#), [383](#)
ファイルの更新 [188](#)
プリンターへの書き込み [189](#)
プロセッシング
概要 [177](#)
既存ファイル [195](#)
逆順の [187](#)
新規ファイル [196](#)
z/OS ユニックスファイル [s 198](#)
ブロック化、レコードの [184](#), [197](#)
ブロック・サイズ [184](#), [383](#)
レコードの置換 [189](#)
レコードの追加 [188](#)
ASCII テープ・ファイル [198](#)
ASSIGN 節 [177](#)
BLOCK CONTAINS 節 [184](#), [383](#)
DATA DIVISION 記入項目 [178](#)
ENVIRONMENT DIVISION 記入項目 [177](#)
FASTSORT のもとで同じ入出力ファイルを使用 [257](#)
FASTSORT の要件 [257](#)
z/OS のもとでの
環境変数 [191](#)
ジョブ制御言語 (JCL) (job control language (JCL))
[192](#)
定義 [191](#), [193](#)
ファイルの可用性 [188](#)
ファイルの作成 [191](#), [193](#)
DD ステートメント [191](#), [193](#)
QUALIFY コンパイラー・オプション [434](#)
QUOTE コンパイラー・オプション [379](#)

R

RANGE 組み込み関数
テーブル計算の例 [95](#)
統計計算の例 [68](#)
RCF

RCF (続き)
送付 [xxxiii](#)
RD パラメーター、JOB または EXEC ステートメント [826](#)
READ NEXT ステートメント [209](#)
READ ステートメント
行順次ファイル [239](#)
マルチスレッド化シリアライゼーション [654](#)
AT END 句 [269](#)
QSAM [187](#)
VSAM [209](#)
RECORD CONTAINS 節
FILE SECTION 記入項目 [14](#)
RECORD KEY 節
KSDS ファイルの基本キーの識別 [204](#)
RECORDING MODE 節
可変長レコード、QSAM [180](#), [181](#)
固定長レコード、QSAM [179](#)
レコード形式の指定 [178](#)
QSAM ファイル [14](#)
REDEFINES 節を使用して、レコードをテーブルに作成 [81](#)
RELEASE FROM ステートメント
例 [247](#)
RELEASE との比較 [247](#)
RELEASE ステートメント
RELEASE FROM との比較 [247](#)
SORT での [247](#)
REM 組み込み関数 [68](#)
RENT コンパイラー・オプション
アドレス可能性への影響 [41](#)
オブジェクト指向 COBOL のための [321](#), [326](#)
説明 [435](#)
データの受け渡し時 [42](#)
パフォーマンスの考慮事項 [847](#)
マルチオプションの相互作用 [40](#), [376](#)
DLL 用 [640](#)
IMS 用 [573](#)
Java とのインターオペラビリティのための [321](#), [326](#)
REPLACE ステートメント
説明 [484](#)
Db2 に関する考慮事項 [566](#)
REPLACING 句 (INSPECT) の例 [121](#)
REPOSITORY 段落
クライアント [760](#)
クラス [748](#)
コーディング [6](#)
サブクラス [771](#)
RERUN 節
チェックポイント・リスタート [261](#)
RETURN ステートメント
出力プロシージャで必要 [248](#)
INTO 句を指定した [248](#)
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 [632](#)
オペレーティング・システムに制御権が戻されるとき
[632](#)
言語環境プログラム・サービスへの呼び出し [856](#)
プログラム間での戻りコードの共用 [632](#)
CICS ECI 呼び出し [549](#)
Db2 についての考慮事項 [561](#)
INVOKE で設定しない [764](#)
RETURNING 句
CALL ステートメント [633](#)
INVOKE ステートメント [766](#)
PROCEDURE DIVISION ヘッダー [633](#), [754](#)

REVERSE 組み込み関数 [123](#)
 REWRITE ステートメント
 マルチスレッド化シリアライゼーション [654](#)
 QSAM [187](#)
 VSAM [209](#)
 RLS パラメーター [223](#)
 RMODE
 説明 [40](#)
 EXIT モジュール [886](#)
 RMODE コンパイラー・オプション
 アドレス可能度への影響 [40](#)
 説明 [436](#)
 データの受け渡し時 [42](#)
 パフォーマンスの考慮事項 [847](#)
 マルチオプションの相互作用 [40](#)
 ROUNDED 句 [860](#)
 RRDS (相対レコード・データ・セット)
 可変長レコード [202](#), [206](#)
 可変長レコードのシミュレート [206](#)
 固定長レコード [202](#), [206](#)
 パフォーマンスの考慮事項 [226](#)
 ファイル・アクセス・モード [207](#)
 編成 [206](#)
 RULES コンパイラー・オプション
 説明 [437](#)

S

S 形式レコード
 概要 [182](#)
 要求 [181](#)
 レイアウト [183](#)
 S レベルのエラー・メッセージ [309](#), [493](#)
 SD (ソート記述) 項目の例 [246](#)
 SEARCH ALL ステートメント
 指標を変更するための [78](#)
 テーブルは順序付けが必要 [93](#)
 二分探索 [92](#)
 例 [93](#)
 SEARCH ステートメント
 指標を変更するための [78](#)
 逐次探索 [91](#)
 テーブルの複数のレベルを検索するためのネスト [92](#)
 例 [92](#)
 SELECT OPTIONAL
 QSAM [188](#)
 VSAM [212](#)
 SELECT 節
 入出力ファイルの変更 [10](#)
 ファイルの命名 [9](#)
 ASSIGN 節 [9](#)
 SELF [762](#)
 SEQUENCE コンパイラー・オプション [439](#)
 SERVICE LABEL ステートメント [484](#)
 SERVICE コンパイラー・オプション [440](#)
 SET 条件名 TO TRUE ステートメント
 スイッチおよびフラグ [106](#)
 例 [108](#), [109](#)
 SET ステートメント
 オブジェクト参照用 [762](#)
 関数ポインター・データ項目用 [617](#)
 指標データ項目を変更するための [78](#)
 指標を変更するための [78](#)
 条件設定用の、例 [106](#)

SET ステートメント (続き)
 その中でのプログラム名の処理 [432](#)
 デバッグ用の使用 [488](#)
 プロシージャー・ポインター・データ項目用 [617](#)
 SIGN IS SEPARATE 節
 移植性 [50](#)
 印刷 [50](#)
 行順次ファイル用 [241](#)
 符号付き国別 10 進数データに必要 [50](#)
 SMARTBIN コンパイラー・オプション
 パフォーマンスの考慮事項 [847](#)
 SORT ステートメント
 概要 [243](#)
 制約事項 [243](#)
 説明 [251](#)
 ASCENDING|DESCENDING KEY 句 [252](#)
 CICS アプリケーションの場合の制約事項 [262](#)
 CICS のもとで
 予約語テーブルの変更 [553](#)
 COLLATING SEQUENCE 句 [7](#), [254](#)
 GIVING 句 [251](#)
 USING 句 [251](#)
 SORT-CONTROL 特殊レジスター [259](#)
 SORT-CORE-SIZE 特殊レジスター [259](#)
 SORT-FILE-SIZE 特殊レジスター [259](#)
 SORT-MESSAGE 特殊レジスター [259](#)
 SORT-MODE-SIZE 特殊レジスター [259](#)
 SORT-RETURN 特殊レジスター
 ソートまたはマージの終了 [255](#)
 ソートまたはマージの成功の判断 [254](#)
 SORTCKPT DD ステートメント [261](#)
 SOURCE および NUMBER 出力の例 [504](#)
 SOURCE コンパイラー・オプション
 出力の取得 [500](#)
 説明 [442](#)
 SOURCE-COMPUTER 段落 [6](#)
 SPACE コンパイラー・オプション [443](#)
 SPECIAL-NAMES 段落
 コーディング [6](#)
 QSAM ファイル [199](#)
 SQL コンパイラー・オプション
 使用 [561](#)
 制約事項
 オブジェクト指向プログラム [743](#)
 バッチでのコンパイル [562](#)
 説明 [443](#)
 SQL ステートメント
 概要 [557](#)
 国別 10 進数データの使用 [560](#)
 コーディング
 概要 [558](#)
 制約事項 [558](#)
 制約事項 [558](#)
 バイナリー・データの使用 [561](#)
 文字データの使用 [559](#)
 戻りコード [561](#)
 CCSID の決定 [563](#)
 Db2 サービスのための使用 [557](#)
 EXIT コンパイラー・オプション [904](#)
 SQL DECLARE [559](#)
 SQL INCLUDE [559](#)
 SQLCA
 Db2 からの戻りコード [561](#)

SQLCA (続き)
SQL ステートメントを使用するプログラムについて宣言 [558](#)
SQLIMS ステートメントを使用するプログラムについて宣言 [570](#)
SQLCCSID コンパイラー・オプション
ストリング・データの CCSID への影響 [563](#)
説明 [444](#)
パフォーマンスの考慮事項 [565](#)
Db2 コプロセッサで推奨する [564](#)
SQLIMS コンパイラー・オプション
使用 [571](#)
制約事項
パッチでのコンパイル [572](#)
SQLIMS ステートメント
コーディング
概要 [570](#)
EXIT コンパイラー・オプション [904](#)
SQLIMS INCLUDE [570](#)
SQRT 組み込み関数 [68](#)
SSRANGE コンパイラー・オプション
参照変更 [118](#)
使用 [492](#)
説明 [446](#)
パフォーマンスの考慮事項 [848](#)
STACK ランタイム・オプション
データ・ロケーションへの影響 [42](#)
マルチオプションの相互作用 [41](#)
STANDARD 節、FD 記入項目 [14](#)
START ステートメント
マルチスレッド化シリアライゼーション [654](#)
VSAM [209](#)
stderr
行送りの制御 [37](#)
DISPLAY による送信 [37](#)
DISPLAY の設定 [581](#)
stdin
ACCEPT による読み取り [35](#)
stdout
行送りの制御 [37](#)
DISPLAY による送信 [37](#)
DISPLAY の設定 [581](#)
STEPLIB 環境変数
コンパイラーの指定の例 [314](#)
説明 [581](#)
STGOPT コンパイラー・オプション [447](#)
STOP RUN ステートメント
サブプログラムにおける [604](#)
マルチスレッド化 [604](#)
メインプログラムにおける [604](#)
STRING ステートメント
オーバーフロー条件 [266](#)
使用 [111](#)
例 [112](#)
DBCS データを使用する [867](#)
SUM 組み込み関数、テーブル計算の例 [95](#)
SUPER [767](#)
SUPPRESS コンパイラー・オプション [448](#)
SYMBOLIC CHARACTERS 節 [8](#)
SYSABEND ファイル
説明 [293](#)
SYSADATA
出力 [378](#)
ファイルの作成 [297](#)

SYSADATA (続き)
レコード、出口モジュール [892](#)
SYSADATA ファイル
説明 [293](#)
ファイル内容 [909](#)
例 [912](#)
レコード記述 [913](#)
レコード・タイプ [911](#)
SYSDEBUG データ・セット
定義 [298](#)
の使用 [451](#)
SYSDEBUG ファイル
説明 [293](#)
SYSIN データ・セット
説明 [292](#)
定義 [294](#)
SYSJAVA ファイル
説明 [293](#)
定義 [297](#)
SYSLIB 環境変数
説明 [312](#)
JNI.cpy の場所の指定 [322](#)
SYSLIB データ・セット
使用されない場合 [888](#)
説明 [292](#)
定義 [296](#)
SYSLIN データ・セット
説明 [293](#)
SYSMDECK ファイル
説明 [293](#)
定義 [298](#)
SYSMDUMP ファイル
説明 [293](#)
SYSOPTF データ・セット
説明 [292](#)
定義 [295](#)
SYSPRINT データ・セット
使用されない場合 [891](#)
説明 [293](#)
定義 [296](#)
SYS PUNCH データ・セット
説明 [293](#), [297](#)
DECK コンパイラー・オプションの要件 [393](#)
SYSTEM データ・セット
説明 [293](#)
定義 [296](#)
メッセージの送信 [448](#)
SYSUDUMP ファイル
説明 [293](#)
SYSUT データ・セット [292](#)

T

TALLYING 句 (INSPECT)、例 [121](#)
TERMINAL コンパイラー・オプション [448](#)
TEST AFTER [109](#)
TEST BEFORE [109](#)
TEST コンパイラー・オプション
説明 [448](#)
デバッグに使用 [499](#)
パフォーマンスの考慮事項 [848](#)
マルチオプションの相互作用 [377](#)
text-name 環境変数 [312](#)
THREAD コンパイラー・オプション

THREAD コンパイラー・オプション (続き)
 オブジェクト指向 COBOL のための [321, 326](#)
 説明 [453](#)
 ネストされたプログラムと一緒に使用できない [614](#)
 パフォーマンスの考慮事項 [848](#)
 マルチオプションの相互作用 [377](#)
 Java とのインターオペラビリティのための [321, 326](#)
 LINKAGE SECTION [17](#)
 TITLE ステートメント
 リストのヘッダーの制御 [5](#)
 TRACK OVERFLOW オプション [186](#)
 TRAP ランタイム・オプション
 行順次ファイルのクローズ [241](#)
 ON SIZE ERROR [266](#)
 QSAM ファイルのクローズ [190](#)
 VSAM ファイルのクローズ [217](#)
 TRUNC コンパイラー・オプション
 説明 [454](#)
 パフォーマンスの考慮事項 [848](#)
 分離型の CICS 変換プログラムのサブオプション [552](#)
 TSO
 コンパイラー・メッセージ用の SYSTERM [296](#)
 ALLOCATE コマンド [287](#)
 CALL コマンド [287](#)
 UNIX のもとでのコンパイル
 概要 [287](#)
 CLIST の例 [288](#)
 TSO のもとでコンパイルするための CLIST [288](#)
 TUNE コンパイラー・オプション
 パフォーマンスの考慮事項 [848](#)

U

U フォーマット・レコード
 要求 [183](#)
 レイアウト [184](#)
 U レベルのエラー・メッセージ [309, 493](#)
 Unicode
 エンコードおよびストレージ [148](#)
 説明 [132](#)
 データ処理 [131](#)
 Db2 での使用 [559](#)
 JNI サービス [795](#)
 UNIX
 API の呼び出し [582](#)
 UNSTRING ステートメント
 オーバーフロー条件 [266](#)
 使用 [113](#)
 例 [114](#)
 DBCS データを使用する [867](#)
 UPPER-CASE 組み込み関数 [123](#)
 USAGE 節
 グループ・レベルの [25](#)
 グループ・レベルの NATIONAL 句 [142](#)
 グループ・レベルの UTF-8 句 [158](#)
 非互換データ [60](#)
 INDEX 句による指標データ項目の作成 [78](#)
 OBJECT REFERENCE [761](#)
 USE FOR DEBUGGING 宣言
 概要 [488](#)
 USE ステートメント [484](#)
 USING 句
 INVOKE ステートメント [764](#)
 PROCEDURE DIVISION ヘッダー [628, 754](#)

UTF-16
 エンコード方式、国別データの [133](#)
 定義 [133](#)
 UTF-8
 「U」 pic 記号 [151](#)
 エンコードおよびストレージ [148](#)
 切り捨てる移動を避ける [709](#)
 国別との間の変換 [162](#)
 組み込み関数の使用 [163](#)
 固定バイト長 [151](#)
 固定文字長 [151](#)
 サポートされる組み込み関数 [155](#)
 サポート対象ステートメント [154](#)
 定義 [133](#)
 データ項目 [151](#)
 データ項目の処理 [162](#)
 動的長 [151](#)
 リテラル [153](#)
 ASCII インバリエント文字のエンコード方式 [133](#)
 character-data [153](#)
 Db2 に関する考慮事項 [156](#)
 DYNAMIC LENGTH [151](#)
 INSPECT の使用を避ける [709](#)
 JNI サービス [797](#)
 PICTURE [151](#)
 XML 文書で参照変更を回避 [162](#)
 XML 文書の構文解析 [708](#)
 XML 文書の生成例 [729](#)
 UTF-8 グループ項目
 英数字グループと比べた場合の利点 [158](#)
 概要 [158](#)
 基本項目として扱われる
 ほとんどの場合 [158](#)
 UTF-8 データ
 形象定数 [156](#)
 これを伴う MOVE ステートメント [159](#)
 条件式の [160, 161](#)
 比較する
 英字、英数字と [161](#)
 英数字グループと [161](#)
 概要 [160](#)
 2 つのオペランド [161](#)
 変換
 概要 [159](#)
 MOVE による英数字または国別からの [159](#)
 NATIONAL-OF による国別への [160](#)
 Unicode 組み込み関数の使用 [164](#)
 UTF-8 データ項目
 参照修飾子 [157](#)
 UTF-8 データ項目の変換
 英数字へ
 DISPLAY-OF を使用した [160](#)
 UTF-8 データ・タイプ [151](#)
 u 接尾辞 cob2 [319](#)

V

V フォーマット・レコード
 要求 [180](#)
 レイアウト [181](#)
 VALUE IS NULL [630](#)
 VALUE OF 節 [14](#)
 VALUE 節
 大きな、TRUNC(BIN) での [455](#)

VALUE 節 (続き)

- 外部浮動小数点に使用できない [54](#)
- 可変長グループへの割り当て [87](#)
- 国別グループでの英数字リテラル、例 [82](#)
- 国別データを持つ英数字リテラルの例 [128](#)
- 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 [82](#)
 - グループ・レベルの [82](#)
 - それぞれの項目に個別に [81](#)
- 内部浮動小数点リテラルの初期化 [50](#)
- COMP-5 を指定したラージ・リテラル [55](#)

VBREF コンパイラー・オプション

- 出力例 [536](#)
- 使用 [500](#)
- 説明 [458](#)

VLR コンパイラー・オプション

- 説明 [458](#)

VSAM

- 動的アクセス
 - オプション [475](#)

VSAM ファイル

- エラー処理 [267](#)
- オープン
 - 概要 [211](#)
 - 空の [212](#)
- 拡張アドレッシング機能 [227](#)
- 環境変数を使用した割り振り [222](#)
- クローズ [217](#)
- 状況コード
 - 説明 [273](#)
 - 例 [273](#)
- 代替索引の作成 [220](#)
- 入出力エラー処理 [217](#)
- 入出力ステートメントのコーディング [209](#)
- パスワードによる保護 [218](#)
- パフォーマンスの考慮事項 [225](#)
- ファイル位置標識 (CRP) [211](#), [214](#)
- ファイル状況キー [217](#)
- ファイルの処理 [201](#)
- ファイル編成の比較 [202](#)
- レコードの更新 [215](#)
- レコードの削除 [217](#)
- レコードの置換 [216](#)
- レコードの追加 [216](#)
- レコードの読み取り [214](#)
- レコード・レベル共用 (RLS)
 - エラー処理 [225](#)
 - 概要 [223](#)
 - 更新問題の帽子 [224](#)
 - 制約事項 [224](#)
- ロード
 - アクセス方式サービス・プログラムによる [213](#)
 - 拡張フォーマット [213](#)
 - 順次 [212](#)
 - 動的またはランダム [212](#)
- DATA DIVISION 記入項目 [207](#)
- ENVIRONMENT DIVISION 記入項目 [203](#)
- z/OS のもとでの
 - データ・セットの定義 [219](#)
 - ファイルの可用性 [219](#)
 - JCL [222](#)
 - RLS モード [223](#)

VSAM 用語

- 非 VSAM 用語との比較 [201](#)

VSAM 用語 (続き)

- BDAM データ・セット [201](#)
- BDAM に対応する RRDS [201](#)
- ISAM に対応する KSDS [201](#)
- QSAM に対応する ESDS [201](#)

VSAMDB ファイル

- ファイルの処理 [229](#)

VSAMOPENFS コンパイラー・オプション [461](#)

W

W レベルのメッセージ [308](#), [493](#)

WHEN 句

- EVALUATE ステートメント [101](#)
- SEARCH ALL ステートメント [93](#)
- SEARCH ステートメント [91](#)

WHEN-COMPILED 組み込み関数 [130](#)

WHEN-COMPILED 特殊レジスター [130](#)

WITH DEBUGGING MODE 節

- デバッグ行 [488](#)
- デバッグ・ステートメント [488](#)

WITH POINTER 句

- STRING [111](#)
- UNSTRING [113](#)

WORD コンパイラー・オプション

- 説明 [461](#)
- 分離型の CICS 変換プログラムの場合に推奨 [552](#)
- マルチオプションの相互作用 [377](#)
- CICS 統合変換プログラムの場合に推奨 [549](#)

WORKING-STORAGE SECTION

- インスタンス・データ [750](#), [772](#)
- インスタンス・メソッド [753](#)
- クライアント [761](#)
- データの保管場所 [391](#)
- ファクトリー・データ [774](#)
- マルチスレッド化の考慮事項 [761](#)

LOCAL-STORAGE との比較

- 概要 [14](#)
- 例 [15](#)

OO クライアント [761](#)

WRITE ADVANCING ステートメント [189](#)

WRITE ステートメント

- 行順次ファイル [239](#)
- マルチスレッド化シリアライゼーション [654](#)

QSAM [187](#)

VSAM [209](#)

X

x サフィックス、cob2 での [318](#), [319](#)

XML GENERATE ステートメント

- タイプ [731](#)
- 名前 [730](#)
- COUNT IN [733](#)
- NAMESPACE [729](#)
- NAMESPACE-PREFIX [729](#)
- NOT ON EXCEPTION [731](#)
- ON EXCEPTION [732](#)
- SUPPRESS [730](#)
- WITH ATTRIBUTES [728](#)
- WITH ENCODING [732](#)
- XML-DECLARATION [729](#)

XML PARSE ステートメント

XML PARSE ステートメント (続き)

概要 [688](#)

使用 [690](#)

NOT ON EXCEPTION [690](#)

ON EXCEPTION [690](#)

XML イベント

エンコードの競合 [712](#)

概要 [693](#)

処理プロシージャ [690](#)

致命的エラー [712](#)

プロセッシング [688](#), [691](#)

CONTENT-CHARACTERS

セグメントの構文解析時 [702](#)

例 [724](#)

END-OF-INPUT

セグメントの構文解析時 [701](#)

例 [724](#)

EXCEPTION [711](#)

NAMESPACE-DECLARATION [697](#)

XML 構文解析

エンコード競合の処理 [712](#)

概要 [687](#)

検証を伴う

概要 [698](#)

制約事項 [699](#)

パフォーマンスの考慮事項 [699](#)

例 [724](#)

終了 [713](#)

処理プロシージャでの制御フロー [694](#)

説明 [690](#)

致命的エラー [712](#)

特殊レジスター [691](#), [693](#)

戻りコード [709](#), [872](#)

理由コード [709](#), [872](#)

例外の処理 [709](#)

1 セグメントずつ

概要 [701](#)

例 [722](#)

XML 構文解析の終了 [713](#)

XML 出力

エンコードの制御 [732](#)

拡張

基本的原理と技法 [737](#)

データ定義の変更例 [738](#)

生成

概要 [727](#)

例 [733](#)

XML 出力の拡張

基本的原理と技法 [737](#)

データ定義の変更例 [738](#)

XML 出力の生成

概要 [727](#)

例 [733](#)

XML 処理プロシージャ

エンコード競合の処理 [713](#)

書き込み [691](#)

構文解析例外の処理 [709](#)

指定 [690](#)

特殊レジスターの使用 [691](#), [693](#)

パーサーでの制御フロー [694](#)

複数セグメント [701](#)

例

検証を伴う構文解析 [724](#)

1 セグメントずつ [722](#)

XML 処理プロシージャ (続き)

例 (続き)

XML の処理用プログラム [715](#)

EXIT PROGRAM または GOBACK でのエラー [692](#)

XML PARSE の制約事項 [692](#)

XML-CODE の設定 [713](#)

XML スキーマ [700](#)

XML 生成

エラーの処理 [732](#)

エレメントの生成 [728](#)

概要 [727](#)

指定された属性または要素の生成を抑止 [730](#)

出力の拡張

基本的原理と技法 [737](#)

データ定義の変更例 [738](#)

生成される文字のカウント [728](#)

説明 [727](#)

属性の生成 [728](#)

属性または要素の命名 [730](#)

名前空間接頭部の使用 [729](#)

名前空間の使用 [729](#)

バイト・オーダー・マークなし [732](#)

無視されるデータ項目 [728](#)

例 [733](#)

XML データのタイプの制御 [731](#)

XML 宣言

エンコード宣言の指定 [707](#)

空白文字 (white space) を前に置くことはできない [706](#)

生成 [729](#)

XML パーサー

エラー処理 [711](#)

概要 [688](#)

XML 文書

アクセス [689](#)

イベント

例 [719](#)

英数字の場合のエンコードの指定 [707](#)

エンコード [704](#), [705](#)

エンコードの制御 [732](#)

拡張

基本的原理と技法 [737](#)

データ定義の変更例 [738](#)

各国語 [705](#)

空白文字 (white space) [706](#)

検証を伴う構文解析

概要 [698](#)

制約事項 [699](#)

パフォーマンスの考慮事項 [699](#)

例 [724](#)

構文解析

大きい文書 [703](#)

説明 [690](#)

例 [715](#), [719](#), [722](#)

1 セグメントずつ [701](#)

UTF-8 [708](#)

構文解析例外の処理 [709](#)

サポートされるコード・ページ [704](#)

生成

概要 [727](#)

例 [733](#)

パーサー [688](#)

プロセッシング [687](#)

EBCDIC 特殊文字 [708](#)

XML 宣言 [706](#)

XML 文書の空白文字 (white space) [706](#)

XML 文書の検証

概要 [698](#)

制約事項 [699](#)

パフォーマンスの考慮事項 [699](#)

例 [724](#)

XML 文書の構文解析

概要 [688](#)

空白文字 (white space) [706](#)

検証を伴う

概要 [698](#)

制約事項 [699](#)

パフォーマンスの考慮事項 [699](#)

例 [724](#)

説明 [690](#)

1 セグメントずつ

概要 [701](#)

例 [722](#)

UTF-8 [708](#)

XML 宣言 [706](#)

XML 例外コード

構文解析 [872](#)

生成 [880](#)

XMLPARSE(COMPAT) が有効な構文解析の

処理可能でない [877](#)

処理可能な [874](#)

XMLPARSE(XMLSS) が有効な構文解析の [872](#)

XML-CODE 特殊レジスター

エンコード競合の [712](#)

構文解析での使用 [687](#)

構文解析の終了 [713](#)

構文解析の例外 [711](#)

構文解析の例外コード [872](#)

コード・ページの矛盾に関連した [712](#)

生成での使用 [731](#)

生成の例外 [732](#)

生成の例外コード [880](#)

説明 [692](#)

ゼロ以外の値の後の続行 [713](#)

致命的エラー [712](#)

内容 [694](#)

パーサーと処理プロシーチャー間の制御フロー [694](#)

戻りコード [709](#), [872](#)

理由コード [709](#), [872](#)

1 に設定 [701](#)

100,000 を減算 [712](#)

XMLPARSE(COMPAT) が有効な構文解析の例外コード

エンコードの競合 [710](#)

処理可能でない [877](#)

処理可能な [874](#)

XMLPARSE(XMLSS) が有効な構文解析の例外コード [872](#)

-1 に設定 [694](#), [713](#)

XML-EVENT 特殊レジスター

構文解析の例外 [711](#)

使用 [687](#), [691](#)

説明 [692](#)

内容 [693](#), [714](#)

XML-INFORMATION 特殊レジスター

説明 [692](#)

内容 [696](#)

XML-NAMESPACE 特殊レジスター

使用 [687](#)

説明 [692](#)

内容 [697](#)

XML-NAMESPACE-PREFIX 特殊レジスター

使用 [687](#)

説明 [692](#)

内容 [697](#)

XML-NNAMESPACE 特殊レジスター

使用 [687](#)

説明 [692](#)

内容 [697](#)

XML-NNAMESPACE-PREFIX 特殊レジスター

使用 [687](#)

説明 [692](#)

内容 [697](#)

XML-NTEXT 特殊レジスター

構文解析の例外 [711](#)

使用 [687](#)

説明 [692](#)

内容 [696](#)

XML-TEXT 特殊レジスター

構文解析の例外 [711](#)

使用 [687](#)

説明 [692](#)

内容 [696](#), [714](#)

XMLPARSE コンパイラー・オプション

説明 [462](#)

パーサーの選択 [687](#)

XPLINK ランタイム・オプション

設定 [329](#)

デフォルトとしては推奨されない [329](#)

XREF コンパイラー・オプション

コピーブック・データ・セットの検索 [494](#)

出力の取得 [500](#)

説明 [463](#)

データおよびプロシーチャー名の検出 [494](#)

XREF 出力

データ名相互参照 [531](#)

プログラム名相互参照 [533](#)

COPY/BASIS 相互参照 [533](#)

Z

z/OS

下でのプログラムの実行 [637](#)

メイン・パラメーターへのアクセス

概要 [637](#)

例 [637](#)

UNIX のもとでのコンパイル [279](#)

z/OS Debugger

説明 [485](#)

z/OS UNIX ファイルシステム

環境変数によるファイルの定義 [175](#)

コンパイラー・データ・セット [282](#)

ACCEPT によるファイルの読み取り [35](#)

DISPLAY によるファイルの書き込み [37](#)

DLL の探索順序 [644](#)

QSAM によるファイルの処理 [198](#)

z/OS デバッグ

コンパイラー・オプション [499](#)

z/OS ユニックス

オブジェクト指向アプリケーションのコンパイル

概要 [321](#)

例 [323](#)

オブジェクト指向アプリケーションの準備

概要 [322](#)

例 [323](#)

z/OS ユニックス (続き)

環境変数の設定

概要 [580](#)

例 [582](#)

環境変数へのアクセス

概要 [580](#)

例 [582](#)

コピーブック 482

コピーブック探索順序 [312](#), [317](#), [482](#)

コンパイラー・オプションの指定 [312](#)

コンパイラー環境変数 [311](#)

実行環境 [579](#)

実行する、オブジェクト指向アプリケーションを

概要 [324](#)

XPLINK リンケージ [329](#)

制約事項 [579](#)

添え字からのコンパイル [319](#)

ソートおよびマージの制限 [243](#)

プログラムの開発 [579](#)

プログラムの実行 [579](#)

プログラムは再入可能でなければならない [620](#)

メイン・パラメーターへのアクセス

概要 [584](#)

例 [584](#)

リンクする、オブジェクト指向アプリケーションを

概要 [322](#)

例 [323](#)

UNIX のもとのコンパイル [311](#)

z/OS ユニックス シェルでのリンク

情報を cob2 へ引き渡す [316](#)

c89 コマンド [314](#)

cob2 コマンドの使用

概要 [313](#)

例 [315](#)

DLL [314](#)

zlib [831](#)

ZONECHECK コンパイラー・オプション [464](#)

ZONEDATA コンパイラー・オプション [465](#)

ZWB コンパイラー・オプション [468](#)

-I cob2 オプション、コピーブックを検索するための [317](#)

-L cob2 オプション、アーカイブ・ライブラリー・パスを指定するための [317](#)

-l cob2 オプション、アーカイブ・ライブラリー名を指定するための [317](#)

-Mcob2 メイク依存ファイルを生成するオプション [317](#)

-o cob2 オプション、出力ファイルを指定するための [317](#)

-q cob2 オプション、コンパイラー・オプションを指定するための [317](#)

-v cob2 オプション、コンパイル・ステップおよびリンク・ステップを表示および実行するための [317](#)

! 文字、16 進値 [708](#)

.a サフィックス、cob2 での [318](#)

.adt サフィックス、cob2 での [318](#)

.adt ファイル [378](#)

.cbl サフィックス、cob2 での [318](#)

.dbg サフィックス、cob2 での [318](#)

.dbrm サフィックス、cob2 での [318](#)

.dek サフィックス、cob2 での [318](#)

.lst サフィックス、cob2 での [318](#)

.o サフィックス、cob2 での [318](#)

.u 接尾辞付き cob2 [319](#)

.x サフィックス、cob2 での [318](#), [319](#)

[文字、16 進値 [708](#)

] 文字、16 進値 [708](#)

*CBL ステートメント [481](#)

*CONTROL ステートメント [481](#)

文字、16 進値 [708](#)

| 文字、16 進値 [708](#)

[特殊文字]

_BPX_SHAREAS 環境変数 [583](#)

_CEE_ENVFILE 環境変数

説明 [581](#)

Java 設定値を示す [327](#)

_CEE_RUNOPTS 環境変数

説明 [581](#)

ランタイム・オプションの指定 [579](#)

XPLINK の設定 [329](#)

_IGZ_SYSOUT 環境変数

設定 [581](#)

STDOUT または STDERR への書き込み [37](#)

-# cob2 オプション、コンパイル・ステップおよびリンク・ステップを表示するための [317](#)

-b cob2 オプション

情報をリンカーに引き渡すための [316](#)

DLL 作成用 [314](#)

-c cob2 オプション、リンクではなくコンパイルのための [316](#)

-comprc_ok cob2 オプション、戻りコードに基づいてコンパイラーを制御するための [316](#)

-e cob2 オプション、入り口点を指定するための [316](#)

-g cob2 オプション、TEST を指定するのと同様 [316](#)



プログラム番号: 5655-EC6

SC27-8714-04

