

Enterprise COBOL for z/OS



プログラミング・ガイド

バージョン 6.2

Enterprise COBOL for z/OS



プログラミング・ガイド

バージョン 6.2

注記

本書および本書で紹介する製品をご使用になる前に、 963 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM Enterprise COBOL for z/OS バージョン 6 リリース 2 (プログラム番号 5655-EC6) および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。製品のレベルに応じた正しい版を使用していることを確認してください。

Enterprise COBOL for z/OS libraryで、ソフトコピー資料を無料で表示またはダウンロードできます。Enterprise COBOL for z/OS は継続的デリバリー (CD) モデルをサポートしており、資料は CD モデルで配布されるフィチャーを記述するために更新されるので、2 カ月ごとに更新の有無を確認することをお勧めします。

資料番号を更新することなく、本リリースの製品資料を定期的に更新するのが当社の意図です。製品資料のバージョンを一意的に参照する必要がある場合は、更新日とともに資料番号を参照してください。

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

原典： SC27-8714-01 (31 May 2019 update)
Enterprise COBOL for z/OS
Programming Guide
Version 6.2
Second edition (May 2019)

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

© Copyright IBM Corporation 1991, 2019.

目次

表	xiii
-------------	------

前書き	xv
---------------	----

本書について	xv
本書の使い方	xv
略語	xv
一般に使用される用語の比較	xvi
構文図の読み方	xvii
例の示し方	xix
追加の資料およびサポート	xix
変更の要約	xix
バージョン 6 リリース 2 (PTF インストール済 み)	xix
バージョン 6 リリース 2	xxii

第 1 部 プログラムのコーディング . . 1

第 1 章 プログラムの構造 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 のマークを付ける	5
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	6
例: FILE-CONTROL の記入項目	7
照合シーケンスの指定	7
シンボリック文字を定義する	9
ユーザー定義のクラスを定義する	9
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	15
別のプログラムからのデータの使用	17
データの処理	19
PROCEDURE DIVISION 内でロジックが分割さ れる方法	20
宣言部分	24

第 2 章 データの使用 25

変数、構造、リテラル、および定数の使用	25
変数の使用	25
データ項目とグループ項目の使用	26
リテラルの使用	28
定数の使用	28
形象定数の使用	29
データ項目への値の割り当て	29
例: データ項目の初期化	30

構造の初期化 (INITIALIZE)	33
基本データ項目への値の割り当て (MOVE)	35
グループ・データ項目への値の割り当て (MOVE)	36
算術結果の割り当て (MOVE または COMPUTE)	37
画面またはファイルからの入力への割り当て (ACCEPT)	38
画面上またはファイル内での値の表示 (DISPLAY)	39
システム論理出力装置上でのデータの表示	40
WITH NO ADVANCING の使用	40
組み込み関数の使用	41
テーブル (配列) とポインターの使用	42
ストレージとそのアドレス可能度	43
AMODE の制約事項	44
RMODE の設定	45
データの受け渡しに関するストレージ制限	46
データ域のロケーション	46
LOCAL-STORAGE データのストレージ	46
外部データに対するストレージ	46
QSAM 入出力バッファ用ストレージ	47
ALLOCATE ステートメントのストレージ	47

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	54
外部 10 進数 (DISPLAY および NATIONAL) 項目	54
外部浮動小数点 (DISPLAY および NATIONAL) 項目	54
2 進数 (COMP) 項目	55
固有 2 進数 (COMP-5) 項目	55
バック 10 進数 (COMP-3) 項目	56
内部浮動小数点 (COMP-1 および COMP-2) 項目	56
例: 数値データおよび内部表現	57
データ形式の変換	58
変換および精度	59
ゾーンおよびバック 10 進数データのサイン表記	60
非互換データの検査 (数値のクラス・テスト)	61
算術の実行	62
COMPUTE およびその他の算術ステートメント の使用	63
算術式の使用	63
数字組み込み関数の使用	64
数学用の呼び出し可能サービスの使用	66
データ呼び出し可能サービスの使用	67
例: 数字組み込み関数	68
固定小数点演算と浮動小数点演算の対比	70
浮動小数点計算	70
固定小数点計算	71
算術比較 (比較条件)	71

例: 固定小数点計算および浮動小数点計算	72
通貨記号の使用	72
例: 複数の通貨符号	74

第 4 章 テーブルの処理 75

テーブルの定義 (OCCURS)	75
テーブルのネスト	77
例: 添え字付け	78
例: 指標付け	78
テーブル内の項目の参照	79
添え字付け	79
索引付け	80
テーブルに値を入れる方法	81
テーブルの動的なロード	82
テーブルの初期化 (INITIALIZE)	82
テーブルの定義時の値の割り当て (VALUE)	83
例: PERFORM と添え字付け	85
例: PERFORM および索引付け	86
可変長テーブルの作成 (DEPENDING ON)	87
可変長テーブルのロード	89
可変長テーブルへの値の割り当て	90
複合 OCCURS DEPENDING ON	91
例: 複合 ODO	91
ODO オブジェクト値の変更の影響	92
テーブルの探索	95
逐次探索 (SEARCH)	95
二分探索 (SEARCH ALL)	97
テーブルのソート	98
組み込み関数を使用したテーブル項目の処理	98
例: 組み込み関数を使用したテーブルの処理	99
無制限テーブルおよび無制限グループの処理	100
例: XML 文書を構文解析するための無制限テーブルの使用	100

第 5 章 プログラム・アクションの選択と反復 103

プログラム・アクションの選択	103
アクションの選択項目のコーディング	103
条件式のコーディング	108
プログラム・アクションの繰り返し	112
インラインまたはライン外 PERFORM の選択	112
ループのコーディング	114
テーブルのループ処理	114
複数の段落またはセクションの実行	115

第 6 章 スtringの処理 117

データ項目の結合 (STRING)	117
例: STRING ステートメント	118
データ項目の分割 (UNSTRING)	120
例: UNSTRING ステートメント	121
ヌル終了Stringの取り扱い	123
例: ヌル終了String	124
データ項目のサブStringの参照	124
参照修飾子	126
例: 参照修飾子としての演算式	127
例: 参照修飾子としての組み込み関数	127

データ項目の計算および置換 (INSPECT)	128
例: INSPECT ステートメント	128
データ項目の変換 (組み込み関数)	130
大/小文字の変更 (UPPER-CASE、LOWER-CASE)	130
逆順への変換 (REVERSE)	131
数値への変換 (NUMVAL、NUMVAL-C、NUMVAL-F)	131
あるコード・ページから別のコード・ページへの変換	132
16 進データまたはビット・データへの変換 (HEX-OF、BIT-OF)	133
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)	133
データ項目の評価 (組み込み関数)	134
照合シーケンスに関する単一文字の評価	135
最大または最小データ項目の検出	135
データ項目の長さの検出	138
コンパイルの日付の検出	139

第 7 章 国際環境でのデータの処理 . . . 141

COBOL ステートメントと国別データ	142
組み込み関数と国別データ	145
Unicode および言語文字のエンコード	146
COBOL での国別データ (Unicode) の使用	147
国別データ項目の定義	148
国別リテラルの使用	149
国別文字形象定数の使用	150
国別数値データ項目の定義	151
国別グループ	151
国別グループの使用	152
文字データの保管	155
国別 (Unicode) 表現と間の変換	156
英数字、DBCS、および整数から国別への変換 (MOVE)	156
英数字または DBCS から国別への変換 (NATIONAL-OF)	157
国別の英数字への変換 (DISPLAY-OF)	157
デフォルト・コード・ページのオーバーライド	158
変換例外	158
例: 国別データと間の変換	158
UTF-8 データの処理	159
組み込み関数を使用した UTF-8 エンコード・データの処理	160
中国語 GB 18030 データの処理	164
国別 (UTF-16) データの比較	165
2 つのクラス国別オペランドの比較	166
クラス国別オペランドとクラス数値オペランドの比較	166
国別数値オペランドと他の数値オペランドの比較	167
国別と他の文字String・オペランドとの比較	167
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	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	168

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	174
ファイルの割り振り	176
入出力エラーの検査	177
第 9 章 QSAM ファイルの処理	179
COBOL での QSAM ファイルおよびレコードの定義	179
レコード形式の指定	180
ブロック・サイズの設定	188
QSAM ファイルの入出力ステートメントのコーディング	191
QSAM ファイルのオープン	192
QSAM ファイルの動的作成	193
QSAM ファイルへのレコードの追加	194
QSAM ファイルの更新	194
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	194
QSAM ファイルのクローズ	195
QSAM ファイルのエラーの処理	196
QSAM ファイルの操作	196
QSAM ファイルの定義および割り振り	197
QSAM ファイルの検索	200
ファイル属性をプログラムと一致させる	201
ストライプ拡張形式 QSAM データ・セットの使用	203
QSAM ファイル用のバッファの割り振り	204
QSAM を使用する z/OS UNIX ファイルへのアクセス	205
テープ上の QSAM ASCII ファイルの処理	206
第 10 章 VSAM ファイルの処理	209
VSAM ファイル	210
VSAM ファイル編成およびレコードの定義	212
VSAM ファイルの順次編成の指定方法	212
VSAM ファイルの索引編成の指定方法	213
VSAM ファイルの相対編成の指定方法	214
VSAM ファイルのアクセス・モードの指定	215
VSAM ファイルのレコード長の定義	216
VSAM ファイルの入出力ステートメントのコーディング	218
ファイル位置標識	220
ファイルのオープン (ESDS、KSDS、または RRDS)	220
VSAM ファイルからのレコードの読み取り	223
VSAM ファイル内のレコードの更新	224
VSAM ファイルへのレコードの追加	225
VSAM ファイル内のレコードの置換	226
VSAM ファイルからのレコードの削除	226
VSAM ファイルのクローズ	227
VSAM ファイルでのエラー処理	227

パスワードによる VSAM ファイルの保護	228
例: VSAM 索引付きファイルのパスワード保護	229
z/OS および z/OS UNIX のもとでの VSAM データ・セットの操作	229
VSAM ファイルの定義	230
代替索引の作成	231
VSAM ファイルの割り振り	233
RLS による VSAM ファイルの共用	234
VSAM ファイル用のレコード域の割り振り	236
VSAM パフォーマンスの向上	237
拡張アドレッシング機能サポート	239
第 11 章 行順次ファイルの処理	241
COBOL での行順次ファイルおよびレコードの定義	241
行順次ファイルの構造の記述	242
行順次ファイルの制御文字	242
行順次ファイルの割り振り	243
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	243
行順次ファイルのオープン	244
行順次ファイルからのレコードの読み取り	244
行順次ファイルへのレコードの追加	245
行順次ファイルのクローズ	245
行順次ファイルのエラーの処理	246
第 12 章 ファイルのソートおよびマー	247
ジ	247
ソートおよびマージ・プロセス	248
ソートまたはマージ・ファイルの記述	249
ソートまたはマージへの入力 of 記述	249
例: SORT 用のソート・ファイルおよび入力ファイルの記述	250
入力プロシージャのコーディング	251
ソートまたはマージからの出力の記述	252
出力プロシージャのコーディング	252
例: DFSORT を使用する際の出力プロシージャのコーディング	253
入出力プロシージャに関する制約事項	254
ソート・データ・セットおよびマージ・データ・セットの定義	255
可変長レコードのソート	255
ソートまたはマージの要求	256
ソートまたはマージ基準の設定	257
例: 入出力プロシージャを使用したソート	258
代替照合シーケンスの選択	259
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	259
ソートまたはマージの成否の判断	260
ソートまたはマージ操作の途中停止	260
FASTSORT によるソート・パフォーマンスの向上	261
JCL に関する FASTSORT の要件	261
ソート入出力ファイルに関する FASTSORT の要件	261
NOFASTSORT によるソート・エラーの検査	264
ソート動作の制御	264

制御ステートメントによる DFSORT デフォルト の変更	266
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	266
ソート・ファイル用のスペースの割り振り	267
DFSORT によるチェックポイント・リスタートの 使用	267
CICS のもとでのソート	268
CICS SORT アプリケーションの制約事項	268

第 13 章 エラーの処理 271

ダンプの要求	271
ストリングの結合および分割におけるエラーの処理	272
算術演算でのエラーの処理	273
例: 0 による除算の検査	273
入出力操作でのエラーの処理	273
ファイルの終わり条件 (AT END) の使用	276
ERROR 宣言のコーディング	276
ファイル状況キーの使用	277
例: ファイル状況キー	279
VSAM 状況コードの使用 (VSAM ファイルの み)	279
例: VSAM 状況コードの検査	280
INVALID KEY 句のコーディング	281
例: FILE STATUS および INVALID KEY	282
プログラム呼び出し時のエラーの処理	283
エラー処理用のルーチンの作成	283

第 2 部 プログラムのコンパイルお よびデバッグ 285

第 14 章 z/OS のもとでのコンパイル 287

JCL を使用したコンパイル	288
カタログ式プロシージャの使用	288
プログラムをコンパイルするための JCL の作成	293
TSO のもとでのコンパイル	295
例: TSO のもとでコンパイルするための ALLOCATE および CALL	296
例: TSO のもとでコンパイルするための CLIST	297
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
コンパイラー入出力の定義	300
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	300
ソース・コード・データ・セットの定義 (SYSIN)	303
コンパイラー・オプション・データ・セットの定 義 (SYSOPTF)	303
ソース・ライブラリーの指定 (SYSLIB)	304
出力データ・セットの定義 (SYSPRINT)	304
コンパイラー・メッセージの端末への送信 (SYSTEM)	305
オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)	305
関連データ・ファイル (SYSADATA) の定義	306

Java ソース出力ファイル (SYSJAVA) の作成	306
デバッグ・データ・セットの定義 (SYSDEBUG)	306
ライブラリー処理出力ファイル (SYSMDCK)	307
の定義	307
z/OS のもとでのコンパイラー・オプションの指定	307
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	308
例: JCL によるコンパイラー・オプションの指定	309
例: TSO のもとでのコンパイラー・オプション の指定	309
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	310
複数プログラムのコンパイル (バッチ・コンパイル)	311
例: バッチ・コンパイル	312
バッチ・コンパイルでのコンパイラー・オプショ ンの指定	313
例: バッチ・コンパイルでのオプションの優先順 位	314
例: バッチ・コンパイルでの LANGUAGE オプ ション	315
ソース・プログラムのエラーの訂正	316
コンパイラー・メッセージのリストの生成	317
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	317
コンパイラー診断メッセージの形式	317
コンパイラー診断メッセージの重大度コード	318

第 15 章 z/OS UNIX のもとでのコンパ イル 321

z/OS UNIX のもとでの環境変数の設定	321
z/OS UNIX のもとでのコンパイラー・オプション の指定	323
cob2 コマンドを使用したコンパイルおよびリンク	324
z/OS UNIX のもとでの DLL の作成	325
例: z/OS UNIX のもとでの cob2 によるコンパ イルおよびリンク	326
cob2 の構文およびオプション	327
cob2 入出力ファイル	329
スクリプトを使用したコンパイル	330

第 16 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 333

z/OS UNIX 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	333
z/OS UNIX のもとでのオブジェクト指向アプ 리케이션のコンパイル	333
z/OS UNIX のもとでのオブジェクト指向アプ 리케이션の準備	334
例: z/OS UNIX のもとでの COBOL クラス定 義のコンパイルとリンク	335
z/OS UNIX のもとでのオブジェクト指向アプ 리케이션の実行	336
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	338

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	339	NUMPROC	398
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	340	OBJECT	399
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	341	OFFSET	399
Java SDKs for z/OS の使用	343	OPTFILE	400
オブジェクト指向構文、および Java 6 以降	344	OPTIMIZE	402
第 17 章 コンパイラー・オプション	345	OUTDD	403
85 COBOL 標準に準拠するオプション設定	348	PARMCHECK	404
矛盾するコンパイラー・オプション	349	PGMNAME	405
ADATA	350	PGMNAME(COMPAT)	406
ADV	351	PGMNAME(LONGUPPER)	406
AFP	351	PGMNAME(LONGMIXED)	407
APOST/QUOTE	352	使用上の注意	407
ARCH.	352	QUALIFY	408
ARITH	354	RENT	409
AWO	355	RMODE	410
BLOCK0	356	RULES	411
BUFSIZE	357	SEQUENCE	414
CICS	358	SERVICE	415
CODEPAGE	359	SOURCE	415
COMPILE	362	SPACE	416
COPYLOC	362	SQL	416
著作権	364	SQLCCSID	417
CURRENCY	365	SQLIMS	418
DATA	366	SSRANGE	420
DBCS	367	STGOPT	421
DECK	367	SUPPRESS	422
DEFINE	368	TERMINAL	423
DIAGTRUNC	369	TEST	423
DISPSIGN	370	THREAD	428
DLL	371	TRUNC	430
DUMP	372	TRUNC の例 1	431
DYNAM	373	TRUNC の例 2	432
EXIT	374	VBREF	433
EXPORTALL	377	VLR	433
FASTSRT	377	VSAMOPENFS	434
FLAG	378	WORD	435
FLAGSTD	379	XMLPARSE	436
HGPR	381	XREF	437
INITCHECK	382	ZONECHECK	438
INITIAL	383	ZONEDATA	440
INLINE	384	ZWB	443
INTDATE	385	第 18 章 コンパイラー指示ステートメント	445
LANGUAGE	385	第 19 章 デバッグ	451
LINECOUNT	386	ソース言語によるデバッグ	452
LIST	387	プログラム・ロジックのトレース	452
MAP	388	入出力エラーの検出および処理	453
MAXPCF	389	データの妥当性検査	454
MDECK	390	初期化されていないデータの移動、初期化、または設定	454
NAME	392	プロシージャに関する情報の生成	454
NSYMBOL	393	コンパイラー・オプションを使用したデバッグ	456
NUMBER	394	コーディング・エラーの検出	457
NUMCHECK	395	行シーケンス問題の検出	458

I	無効な COBOL データや無効な COBOL プログラムの検査	458
I	有効範囲の検査	459
	診断するエラーのレベルの選択	460
	プログラム・エンティティー定義および参照の検出	462
	データ項目のリスト	462
	デバッガーの使用	463
	リストの入手	464
	例: 短縮リスト	466
	例: SOURCE および NUMBER 出力	468
	例: MAP 出力	469
	LIST 出力の読み取り	474
	例: XREF 出力: データ名相互参照	491
	例: OFFSET コンパイラー出力	495
	例: VBREF コンパイラー出力	496
I	例: 条件付きコンパイル出力	496
I	CEEDUMP 処理での情報の抑止抑止	497

第 3 部 特定の環境に合わせた COBOL プログラムの目標 499

第 20 章 COBOL プログラムの開発 (CICS の場合) 501

	CICS のもとで実行する COBOL プログラムのコーディング	501
	CICS のもとでのシステム日付の取得	503
	COBOL プログラムとの間の呼び出し	504
	ECI 呼び出しの成否の判断	506
	CICS オプションを使用したコンパイル	506
	CICS サブオプションの分離	508
	組み込みの CICS 変換プログラム	508
	分離型の CICS 変換プログラムの使用	509
	CICS 予約語テーブル	511
	CICS HANDLE を使用したエラー処理	512
	例: CICS HANDLE を使用したエラー処理	513

第 21 章 Db2 環境用のプログラミング 515

	Db2 コプロセッサ	515
I	分離型 Db2 プリコンパイラーの使用	516
	SQL ステートメントのコーディング	517
	Db2 コプロセッサを用いた SQL INCLUDE の使用	517
	SQL ステートメントでの文字データの使用	518
	SQL ステートメントでの国別 10 進数データの使用	519
	SQL ステートメントでの国別グループ項目の使用	519
	SQL ステートメントでのバイナリー項目の使用	519
	SQL ステートメントの成否の判断	520
	SQL オプションを使用したコンパイル	520
	Db2 サブオプションの分離	521
	COBOL および Db2 CCSID の決定	522
	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	523

	SQLCCSID または NOSQLCCSID オプションを使用したプログラミング	523
	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	524
	EXEC SQL INCLUDE ステートメントの最後のピリオド	525
	EXEC SQL と REPLACE または COPY REPLACING	525
	END-EXEC ステートメントの後のソース・コード	525
	ホスト変数の複数定義	525
	EXEC SQL ステートメントの継続行	526
	ビット・データ・ホスト変数	526
	SQL-INIT-FLAG	526
	DYNAM または NODYNAM コンパイラー・オプションの選択	527

第 22 章 COBOL プログラムの開発 (IMS の場合) 529

	IMS SQL コプロセッサ	529
	SQLIMS ステートメントのコーディング	530
	IMS SQL コプロセッサで SQLIMS INCLUDE を使用	531
	SQLIMS ステートメントでの文字データの使用	531
	SQLIMS ステートメントでのバイナリー項目の使用	531
	SQLIMS ステートメントの成否の判断	532
	SQLIMS オプションを使用したコンパイル	532
	IMS サブオプションの分離	533
	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	534
	IMS のもとでのオブジェクト指向 COBOL と Java の使用	535
	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	535
	COBOL で開始する COBOL と Java の混合アプリケーションの作成	536
	混合言語 IMS アプリケーションの作成	537

第 23 章 z/OS UNIX のもとでの COBOL プログラムの実行 541

	z/OS UNIX 環境での実行	542
	環境変数の設定およびアクセス	542
	実行に影響を与える環境変数の設定	543
	ランタイム環境変数	543
	例: 環境変数の設定とアクセス	544
	UNIX/POSIX API の呼び出し	545
	z/OS UNIX 下でのメインプログラム・パラメーターへのアクセス	547
	例: z/OS UNIX 下でのメインプログラム・パラメーターへのアクセス	547

第 4 部 複雑なアプリケーションの 構造化 549

第 24 章 サブプログラムの使用	551
メインプログラム、サブプログラム、および呼び出し	551
メインプログラムまたはサブプログラムの終了と再入	552
別のプログラムへの制御権移動	554
静的呼び出しの作成	555
動的呼び出しの作成	555
AMODE 切り替え	558
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	560
静的呼び出しと動的呼び出しの両方の作成	561
例: 静的および動的 CALL ステートメント	561
ネストされた COBOL プログラムの呼び出し	563
再帰呼び出しの実行	567
オブジェクト指向プログラムとの間での呼び出し	567
プロシージャ・ポインターと関数ポインターの使用	568
使用するポインター・タイプの決定	569
代替入り口点の呼び出し	570
プログラムを再入可能にする	570
第 25 章 データの共用	573
データの受け渡し	573
呼び出し側プログラムの中での引数の記述	575
呼び出し先プログラムの中でのパラメーターの記述	576
OMITTED 引数に関するテスト	577
LINKAGE SECTION のコーディング	578
引数を受け渡すための PROCEDURE DIVISION のコーディング	578
受け渡されるデータのグループ化	579
ヌル終了ストリングの処理	579
ポインターによるチェーン・リストの処理	580
戻りコード情報の引き渡し	583
RETURN-CODE 特殊レジスターの使用	583
PROCEDURE DIVISION RETURNING . . . の使用	583
CALL . . . RETURNING の指定	584
EXTERNAL 節によるデータの共用	584
プログラム間でのファイルの共用 (外部ファイル)	585
例: 外部ファイルの使用	585
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	588
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	589
第 26 章 DLL または DLL アプリケーションの作成	591
ダイナミック・リンク・ライブラリー (DLL)	591
DLL を作成するためのプログラムのコンパイル	592
DLL のリンク	593
例: プロシージャ型 DLL アプリケーションのサンプル JCL	594
DLL での CALL ID の使用	595

z/OS UNIX ファイル・システム での DLL の探索順序	596
DLL リンケージと動的呼び出しの併用	596
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	597
非 DLL からの DLL の呼び出し	598
例: 非 DLL からの DLL の呼び出し	599
C/C++ プログラムでの COBOL DLL の使用	600
OO COBOL アプリケーションでの DLL の使用	601

第 27 章 マルチスレッド化のための COBOL プログラムの準備	603
マルチスレッド化	604
マルチスレッド化サポートのための THREAD の選択	605
マルチスレッド化されたプログラムへの制御権移動	606
マルチスレッド化されたプログラムの終了	606
マルチスレッド化によるファイルの処理	607
ファイル定義 (FD) ストレージ	608
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	608
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	609
マルチスレッド化による COBOL 制限の処理	609

第 5 部 Web サービスに COBOL を使用 613

第 28 章 Web サービス・インターフェース	615
---	------------

第 29 章 JSON 入力の処理	617
JSON 文書の構文解析	617
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	618
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	619
JSON 配列の処理	619
JSON PARSE の例	621

第 30 章 JSON 出力の生成	625
------------------------------------	------------

第 31 章 XML 入力の処理	627
COBOL での XML パーサー	628
XML 文書へのアクセス	630
XML 文書の構文解析	630
XML を処理するためのプロシージャの作成	632
XML イベント	634
XML テキストの COBOL データ項目への変換	640
検証を伴う XML 文書の構文解析	641
XML 文書を 1 セグメントずつ構文解析	644
XML-INFORMATION 特殊レジスターを使用した分割の処理	647
XML 文書のエンコード方式	648
XML 入力文書エンコード	649

UTF-8 でエンコードされた XML 文書の構文解析	653
XML PARSE の例外処理	654
XML パーサーによるエラーの処理方法	656
エンコード競合の処理	658
XML 構文解析の終了	659
XML PARSE の例	659
例: 単純な文書の構文解析	660
例: XML の処理用プログラム	660
例: 名前空間を使用する XML 文書の構文解析	665
例: XML 文書を 1 セグメントずつ構文解析	669
例: 検証を伴う XML 文書の構文解析	671
第 32 章 XML 出力の生成	675
XML 出力の生成	675
生成される XML 出力のエンコードの制御	681
XML GENERATE 例外の処理	682
例: XML の生成	682
プログラム XGFX	683
プログラム Pretty	684
プログラム XGFX からの出力	687
XML 出力の拡張	687
例: XML 出力の拡張	688

第 6 部 オブジェクト指向プログラムの開発 691

第 33 章 オブジェクト指向プログラムの作成	693
例: 口座	694
サブクラス	695
クラスの定義	697
クラス定義用の CLASS-ID 段落	698
クラス定義用の REPOSITORY 段落	699
クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION	701
例: クラスの定義	701
クラス・インスタンス・メソッドの定義	702
クラス・インスタンス・メソッド定義用の METHOD-ID 段落	703
クラス・インスタンス・メソッド定義用の INPUT-OUTPUT SECTION	704
クラス・インスタンス・メソッド定義用の DATA DIVISION	704
クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION	705
インスタンス・メソッドのオーバーライド	706
インスタンス・メソッドの多重定義	707
属性 (get および set) メソッドのコーディング	708
例: メソッドの定義	709
クライアントの定義	711
クライアント定義用の REPOSITORY 段落	713
クライアント定義用の DATA DIVISION	714
オブジェクト参照の比較および設定	715
メソッドの呼び出し (INVOKE)	716

クラスのインスタンスの作成および初期化	721
クラスのインスタンスの解放	723
例: クライアントの定義	723
サブクラスの定義	724
サブクラス定義用の CLASS-ID 段落	725
サブクラス定義用の REPOSITORY 段落	726
サブクラス・インスタンス・データ定義用の WORKING-STORAGE SECTION	726
サブクラス・インスタンス・メソッドの定義	727
例: サブクラスの定義 (メソッドに関して)	727
ファクトリー・セクションの定義	728
ファクトリー・データ定義用の WORKING-STORAGE SECTION	729
ファクトリー・メソッドの定義	730
例: ファクトリーの定義 (メソッドに関して)	733
プロシージャ指向 COBOL プログラムのラッピング	738
OO アプリケーションの構造化	739
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	740

第 34 章 Java メソッドとの通信 743

JNI サービスへのアクセス	743
Java 例外の処理	745
ローカル参照とグローバル参照の管理	746
Java アクセス制御	748
Java とのデータ共有	748
COBOL および Java での相互運用可能なデータ型のコーディング	749
Java 用の配列およびストリングの宣言	749
Java 配列の取り扱い	751
Java ストリングの取り扱い	753
例: COBOL で書かれた J2EE クライアント	756
COBOL クライアント (ConverterClient.cbl)	756
Java クライアント (ConverterClient.java)	759
例: バッチ COBOL プログラムから Java を呼び出す	760

第 7 部 特殊処理 765

第 35 章 割り込みおよびチェックポイント・リスタート 767

チェックポイントの設定	767
チェックポイントの設計	768
チェックポイントが成功したかどうかのテスト	769
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	769
チェックポイント時に生成されるメッセージ	770
プログラムの再始動	771
自動再始動の要求	772
据え置き再始動の要求	772
据え置き再始動の要求用の形式	773
再始動用のジョブの再実行依頼	774
例: 特定チェックポイント・ステップでのジョブの再始動	774

例: ステップ再始動の要求	774
例: ステップ再始動のためのジョブの再実行依頼	775
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	775

第 8 部 パフォーマンスおよび生産性の向上 777

第 36 章 プログラムのチューニング 779

最適なプログラミング・スタイルの使用	780
構造化プログラミングの使用	780
一括表示表現	780
シンボリック定数の使用	781
効率的なデータ型の選択	781
効率的な計算データ項目の選択	781
一貫性のあるデータ型の使用	782
算術式の効率化	782
指数計算の効率化	782
効率的な VOLATILE 節の使用	783
テーブルの効率的処理	783
テーブル参照の最適化	785
コードの最適化	786
最適化	786
パフォーマンスを向上させるコンパイラー機能の選択	787
パフォーマンスに関連するコンパイラー・オプション	788
パフォーマンスの評価	792
CICS、IMS、または VSAM での効率的な実行	793
静的呼び出しまたは動的呼び出しの選択	794

第 37 章 コーディングの単純化 795

反復コーディングの除去	795
例: COPY ステートメントの使用	796
言語環境プログラム呼び出し可能サービスの使用	797
言語環境プログラムの呼び出し可能サービスのサンプル・リスト	799
言語環境プログラム・サービスの呼び出し	800
例: 言語環境プログラムの呼び出し可能サービス形式 2 SORT ステートメントを使用したテーブルのソート	801

第 9 部 付録 803

付録 A. 中間結果および算術精度 805

中間結果用の用語	806
例: 中間結果の計算	807
固定小数点データと中間結果	807
加算、減算、乗算、および除算	807
指数	808
例: 固定小数点の算術での指数	810
中間結果での切り捨て	810
バイナリー・データと中間結果	811
固定小数点算術で評価される組み込み関数	811
整数関数	811

混合関数	812
浮動小数点データと中間結果	813
浮動小数点演算で評価される指数	813
浮動小数点演算で評価される組み込み関数	814
非算術ステートメントの算術式	814

付録 B. 2 バイト文字セット (DBCS) データの変換 817

DBCS の表記	817
英数字から DBCS データへの変換 (IGZCA2D)	817
IGZCA2D の構文	818
IGZCA2D の戻りコード	819
例: IGZCA2D	819
DBCS から英数字データへの変換 (IGZCD2A)	820
IGZCD2A の構文	820
IGZCD2A の戻りコード	821
例: IGZCD2A	821

付録 C. XML 参照資料 823

XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外	823
XMLPARSE(COMPAT) が有効な場合の XML PARSE 例外	825
継続を許可する XML PARSE 例外	825
継続を許可しない XML PARSE 例外	829
XML GENERATE 例外	832

付録 D. JSON 参照資料 835

JSON GENERATE 例外	835
JSON PARSE 状態、関連コード、およびランタイム・メッセージ	835
非例外状態とそれに対応する JSON-STATUS 値	836
例外状態とそれに対応する JSON-CODE 値	836
非例外状態ランタイム・メッセージ	837
例外状態ランタイム・メッセージ	838

付録 E. EXIT コンパイラー・オプション 841

ユーザー出口作業域の使用	841
出口モジュールからの呼び出し	842
INEXIT の処理	842
INEXIT パラメーター	843
LIBEXIT の処理	844
ネストされた COPY ステートメントによる LIBEXIT の処理	844
LIBEXIT パラメーター	845
PRTEXIT の処理	847
PRTEXIT パラメーター	847
ADEXIT の処理	848
ADEXIT パラメーター	849
MSGEXIT の処理	850
MSGEXIT パラメーター	851
コンパイラー・メッセージの重大度のカスタマイズ	852
例: MSGEXIT ユーザー出口	855
出口モジュールでのエラー処理	861

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 . . . 862

付録 F. JNL.cpy コピーブック 865

付録 G. COBOL SYSADATA ファイル の内容. 871

SYSADATA ファイルに影響するコンパイラー・オ
プション 871
SYSADATA レコード・タイプ 872
例: SYSADATA 873
SYSADATA レコード記述 874
共通ヘッダー・セクション 875
ジョブ識別レコード: X'0000' 877
ADATA 識別レコード: X'0001' 878
コンパイル単位の開始|終了レコード: X'0002' . . . 878
オプション・レコード: X'0010' 878
外部シンボル・レコード: X'0020' 890
構文解析ツリー・レコード: X'0024' 891
トークン・レコード: X'0030' 906
ソース・エラー・レコード: X'0032' 920
ソース・レコード: X'0038' 920
COPY REPLACING レコード: X'0039' 921
記号レコード: X'0042' 922
記号相互参照レコード: X'0044' 933
ネストされたプログラム・レコード: X'0046' . . . 934
ライブラリー・レコード: X'0060' 935
統計レコード: X'0090' 936
EVENTS レコード: X'0120' 936

付録 H. サンプル・プログラムの使用 941

IGYTCARA: バッチ・アプリケーション 941
IGYTCARA の入力データ 942
IGYTCARA によって作成される報告書 943
IGYTCARA の実行準備 944
IGYTCARB: 対話式プログラム 945
IGYTCARB の実行準備 946
IGYTSALE: ネストされたプログラム・アプリケー
ション 948
IGYTSALE の入力データ 950
IGYTSALE によって作成される報告書 952
IGYTSALE の実行準備 955
示されている言語エレメントおよび概念 956

付録 I. Enterprise COBOL for z/OS のアクセシビリティ機能 961

特記事項. 963
商標 965

用語集. 967

リソース・リスト 1007

Enterprise COBOL for z/OS 1007
関連資料 1007

索引 1011

表

1. FILE-CONTROL 記入項目	7
2. FILE SECTION 記入項目	14
3. プログラム内でのデータ項目の割り当て	29
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	45
5. COMP-5 データ項目の値の範囲	56
6. 数値項目の内部表現	57
7. NUMCLS(PRIM) および有効な符号	62
8. NUMCLS(ALT) および有効な符号	62
9. 算術演算子の評価の順序	64
10. 数字組み込み関数	65
11. 演算組み込み関数と呼び出し可能サービスの互換性	66
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	67
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	68
14. ユーロ記号の 16 進値	73
15. COBOL ステートメントと国別データ	142
16. 組み込み関数と国別文字データ	145
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	154
18. エンコード方式と英数字、 DBCS 、および国別データのサイズ	155
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	174
20. QSAM ファイル割り振り	198
21. QSAM ファイルの最大レコード長	201
22. VSAM 、 COBOL 、および非 VSAM 用語の比較	209
23. VSAM データ・セット・タイプの比較	211
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	212
25. VSAM 固定長レコードの定義	217
26. VSAM 可変長レコードの定義	217
27. VSAM 順次ファイル用入出力ステートメント	218
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	219
29. 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	223
30. VSAM ファイルのレコードを更新するためのステートメント	225
31. VSAM パフォーマンスを改善する方法	237
32. NOFASTSORT によるソート・エラーの検査のメソッド	264
33. ソート動作を制御する方法	265
34. コンパイラー・データ・セット	300
35. 固定長コンパイラー・データ・セットのブロック・サイズ	302
36. 可変長コンパイラー・データ・セットのブロック・サイズ	303
37. z/OS のもとでのコンパイラー出力のタイプ	310
38. コンパイラー診断メッセージの重大度コード	318
39. cob2 コマンドへの入力ファイル	329
40. cob2 コマンドからの出力ファイル	330
41. クラス定義のコンパイルおよびリンクのコマンド	335
42. JVM をカスタマイズするための Java コマンド・オプション	337
43. コンパイラー・オプション	345
44. 相互に排他的なコンパイラー・オプション	349
45. EBCDIC マルチバイト・コード化文字セット ID	361
46. DISPSIGN (COMPAT) オプションまたは DISPSIGN (SEP) を指定した DISPLAY 出力	371
47. LANGUAGE コンパイラー・オプションの値	386
48. 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	403
49. コンパイラー・メッセージの重大度レベル	460
50. コンパイラー・オプションとリストの対応	464
51. MAP 出力で使用する用語	472
52. LIST および MAP 出力で使用する記号	473
53. INFO BYTE セクションでのコンパイラー・オプション	476
54. シグニチャー情報バイト	477
55. CICS のもとでの COBOL およびアセンブラー間の呼び出し	505
56. 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	507
57. 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	510
58. 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	510
59. POSIX 関数呼び出しを使用したサンプル	546
60. 終了ステートメントの影響	552
61. CALL ステートメントでデータを渡す方法	574
62. DLL アプリケーションのコンパイラー・オプション	593
63. DLL アプリケーションのバインダー・オプション	593
64. XML パーサーが使用する特殊レジスター	633
65. 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	636
66. XMLPARSE (COMPAT) 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	636
67. XML 文書のコード化文字セット	649
68. 空白文字 (white-space characters) の 16 進値	650
69. XML エンコード宣言の別名	652
70. さまざまな EBCDIC CCSID 用特殊文字の 16 進値	652
71. XML イベントおよび特殊レジスター	660

72. XML イベントおよび特殊レジスター	666	106. ネストなしの COPY ステートメントによる LIBEXIT の処理	844
73. 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	668	107. ネストされた COPY ステートメントによる LIBEXIT の処理	845
74. ENCODING 句を省略した場合に生成される XML のエンコード	681	108. LIBEXIT パラメーター	846
75. クラス定義の構成	697	109. PRTEXTIT 処理	847
76. インスタンス・メソッド定義の構成	702	110. PRTEXTIT パラメーター	847
77. COBOL クライアントの構成	712	111. ADEXIT 処理	849
78. COBOL クライアントでの引数の合致	718	112. ADEXIT パラメーター	849
79. COBOL クライアントでの戻されるデータ項 目の合致	720	113. MSGEXIT の処理	850
80. ファクトリー定義の構成	729	114. MSGEXIT パラメーター	851
81. ファクトリー・メソッド定義の構成	731	115. FIPS (FLAGSTD) メッセージのカテゴリ	854
82. ローカルおよびグローバル参照の JNI サービス	748	116. 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	863
83. COBOL および Java で相互運用可能なデー タ型	749	117. SYSADATA レコード・タイプ	872
84. COBOL および Java で相互運用可能な配列 およびストリング	750	118. SYSADATA 共通ヘッダー・セクション	875
85. COBOL および Java で相互運用可能でない 配列型	751	119. SYSADATA ジョブ識別レコード	877
86. JNI 配列サービス	751	120. ADATA 識別レコード	878
87. jstring 参照と国別データ間の変換サービス	754	121. SYSADATA コンパイル単位の開始・終了レコ ード	878
88. jstring 参照と英数字データ間の変換サービス	754	122. SYSADATA オプション・レコード	878
89. パフォーマンスに関連するコンパイラー・オ プション	788	123. SYSADATA 外部シンボル・レコード	890
90. パフォーマンス調整のワークシート	792	124. SYSADATA 構文解析ツリー・レコード	891
91. 言語環境プログラム呼び出し可能サービス	799	125. SYSADATA トークン・レコード	906
92. 形式 1 と形式 2 の SORT ステートメントの 比較	801	126. SYSADATA ソース・エラー・レコード	920
93. IGZCA2D の戻りコード	819	127. SYSADATA ソース・レコード	920
94. IGZCD2A の戻りコード	821	128. SYSADATA COPY REPLACING レコード	921
95. Enterprise COBOLに固有な XML PARSE 例外の理由コード	824	129. SYSADATA 記号レコード	922
96. 続行可能な XML PARSE 例外	825	130. SYSADATA 記号相互参照レコード	933
97. 継続を許可しない XML PARSE 例外 (XMLPARSE (COMPAT) の場合)	830	131. SYSADATA ネストされたプログラム・レコ ード	934
98. XML GENERATE 例外	833	132. SYSADATA ライブラリー・レコード	935
99. JSON GENERATE 例外	835	133. SYSADATA 統計レコード	936
100. JSON 非例外状態の理由コード	836	134. SYSADATA EVENTS TIMESTAMP レコー ドのレイアウト	937
101. JSON 例外状態の理由コード	836	135. SYSADATA EVENTS PROCESSOR レコー ドのレイアウト	937
102. ユーザー出口作業域のレイアウト	841	136. SYSADATA EVENTS FILE END レコード のレイアウト	937
103. INEXIT 処理	842	137. SYSADATA EVENTS PROGRAM レコード のレイアウト	938
104. INEXIT パラメーター	843	138. SYSADATA EVENTS FILE ID レコードの レイアウト	938
105. LIBEXIT 処理	844	139. SYSADATA EVENTS ERROR レコードの レイアウト	939

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になる場合があるオペレーティング・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。

重要: 本書全体を通して、Enterprise COBOL for z/OS は Enterprise COBOL と呼ばれています。

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。さらに、オブジェクト指向クラスおよびメソッドの定義、メソッドの呼び出し、およびプログラムでのオブジェクトの参照を行うために役立ちます。

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 言語環境プログラム プログラミング・ガイド」および「IBM z/OS 言語環境プログラム プログラミング・リファレンス」を参照してください。

よく使用される Enterprise COBOL 用語と 言語環境プログラム用語の比較については、xvi 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

使用される用語	長形式
デバッグ・ツール	IBM Debug for z Systems® (以前は IBM Debug Tool for z/OS) ¹
Enterprise COBOL	IBM Enterprise COBOL for z/OS
言語環境プログラム	IBM z/OS 言語環境プログラム
MVS™	MVS/ESA
z/OS UNIX	z/OS UNIX システム・サービス
<p>注:</p> <p>1. IBM Debug for z Systems は、IBM Debug Tool for z/OS に置き換わるものです。COBOL 文書ライブラリーで、IBM Debug Tool for z/OS への参照がすべて変更されているわけではありません。多くの COBOL アプリケーションのデバッグに、引き続き IBM Debug Tool for z/OS V13.1 を使用することができます。ただし、Enterprise COBOL V6 に用意されている新しいデバッグ機能を使用する場合は、最新バージョンの IBM Debug for z Systems が必要です。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/support/knowledgecenter/SSQ2R2_14.0.0/com.ibm.debugtool.doc/common/dcompo.html を参照してください。</p>	

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

- ISO 1989:1985、プログラム言語 - COBOL
- ISO/IEC 1989/AMD1:1992、プログラム言語 - COBOL - 組み込み関数モジュール
- ISO/IEC 1989/AMD2:1994、プログラム言語 - COBOL 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

用語「2002 COBOL 標準」は、次の標準を示します。

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

用語「2014 COBOL 標準」は、次の標準を示します。

- *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 言語環境プログラムおよび IBM Enterprise COBOL for z/OS の情報で使われる用語、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。

言語環境プログラムの用語	対応する Enterprise COBOL の用語
集合体	グループ項目
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。

>>--- 記号は、構文図の始まりを示します。

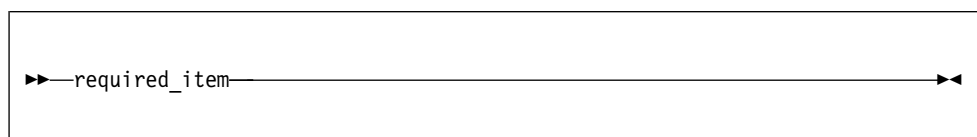
---> 記号は、構文図が次の行に続くことを示します。

>--- 記号は、構文図が前の行からの続きであることを示します。

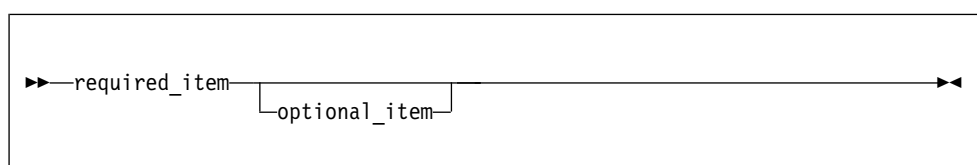
--->< 記号は、構文図の終わりを示します。

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

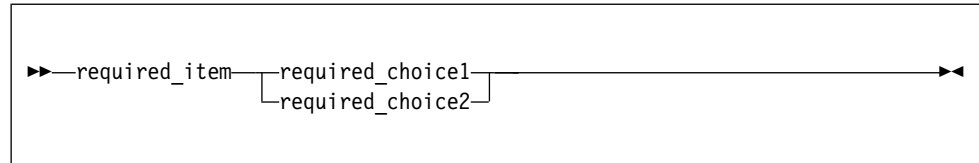
- ・ 必須項目は、水平線 (メインパス) と同じ高さに示されます。



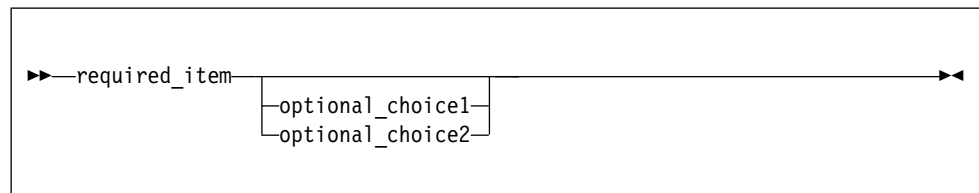
- ・ オプション項目は、メインパスの下側に示されます。



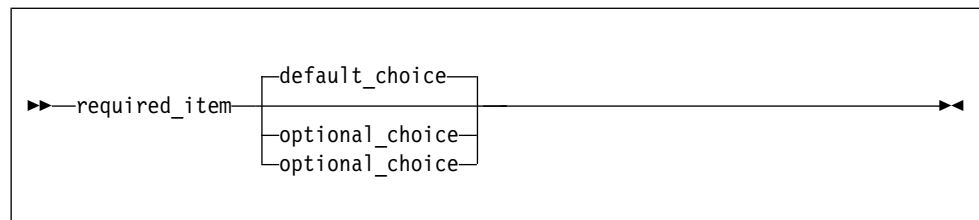
- 複数の項目から選択できる場合には、それらの項目は縦方向に重ねて示されます。それらの項目のうち 1 つを選択しなければならない場合には、スタックのうちの 1 つの項目がメインパス上に示されます。



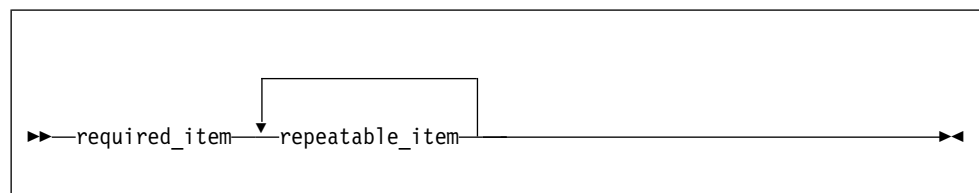
項目の選択が任意である場合には、スタック全体がメインパスより下に置かれます。



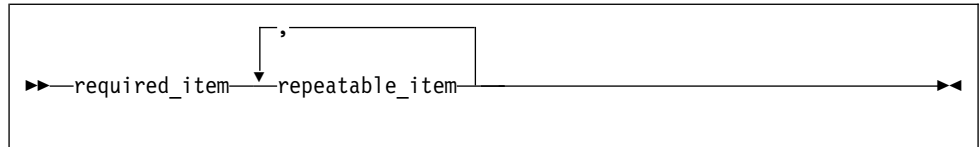
項目のうちの 1 つがデフォルトであれば、それがメインパスより上に示され、残りの選択項目はメインパスより下に示されます。



- メインパス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示されています (例えば、FROM)。これは表記どおり正確につづらなければなりません。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。

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

テキスト内の COBOL キーワードとコンパイラ・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。

追加の資料およびサポート

IBM Enterprise COBOL for z/OS は、このバージョンと前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 (<http://www.ibm.com/support/docview.wss?uid=swg27036733>) 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/home/product/B984385H82239E03/Enterprise_COBOL_for_z/OS に用意されています。

変更の要約

このセクションには、Enterprise COBOL バージョン 6 リリース 2 および PTF インストール済みバージョン 6 リリース 2 向けのこの資料に対して行われた主な変更がリストされています。本書で解説されている変更には、読者の便宜のため、参照ページが記載されています。最新の技術的な変更は、HTML 版では >| と |< の間に示され、PDF 版では左側の余白にある縦棒 (|) で示されています。

バージョン 6 リリース 2 (PTF インストール済み)

新しいコンパイラ・オプションと変更されたコンパイラ・オプション

- 新しいコンパイラ・オプションは以下のとおりです。

- PI91584: COPYLOC: 新しい COPYLOC コンパイラー・オプションは、PDSE (または PDS) データ・セット、あるいは z/OS UNIX ディレクトリーを、ライブラリー・フェーズ中に COPY メンバーを探索する場所として追加するために使用できます。 (362 ページの『COPYLOC』)
- PH05855: INITIAL: 新しい INITIAL コンパイラー・オプションによって、IS INITIAL 節を PROGRAM-ID 段落に追加することなく、また動的 CALL ステートメントも動的 CANCEL ステートメントも使用することなく、呼び出されるたびに初期値がデータ項目に入るプログラムを得ることができます。 (383 ページの『INITIAL』)
- 変更されたコンパイラー・オプション:
 - PI90571: ZONEDATA: ZONEDATA オプションが更新され、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のある USAGE DISPLAY データ項目または PACKED-DECIMAL データ項目に対する、MOVE ステートメントの動作、比較、および計算に作用するようになりました。 (440 ページの『ZONEDATA』)
 - PI91585: RULES: 新しいサブオプション OMITODOMIN | NOOMITODOMIN が RULES オプションに追加されました。このサブオプションは、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が警告メッセージを出すかどうかを制御します。 (411 ページの『RULES』)
 - PI91586: RULES: 新しいサブオプション UNREF | NOUNREFALL | NOUNREFSOURCE が RULES オプションに追加されました。このサブオプションは、コンパイラーが参照されていないデータ項目を報告するかどうかを制御します。さらに、COPY メンバーで宣言されていないデータ項目に関してのみ報告を行う (NOUNREFSOURCE) か、すべてのデータ項目に関して報告を行う (NOUNREFALL) かを制御します。 (411 ページの『RULES』)
 - PI96135: NUMCHECK(PAC): 偶数桁を持つパック 10 進数 (COMP-3) データ項目に対して、未使用のビットでゼロの有無が検査されます。 (395 ページの『NUMCHECK』)
 - PI98480: NUMCHECK(ZON): 新しいサブオプション ALPHNUM | NOALPHNUM が NUMCHECK(ZON) オプションに追加されました。このサブオプションは、コンパイラーが英数字データ項目、英数字リテラル、または英数字形象定数と比較されるゾーン 10 進数データ項目の暗黙数値クラス・テスト用のコードを生成するかどうかを制御します。 (395 ページの『NUMCHECK』)
 - PH04369: RULES(NO EVENPACK) は、名前が DFH、DSN、EYU、または SQL で始まる偶数桁 PACKED-DECIMAL データ項目 (これらは CICS および Db2 用に、または CICS および Db2 によって生成されたデータ項目です) についてメッセージを出しません。 (411 ページの『RULES』)
 - PH04485: TEST: 新しいサブオプション DSNAME | NODSNAME が TEST|NOTEST(SEPARATE) オプションに追加されました。このサブオプションは、外部ファイル名 (コンパイル中に使用される SYSDEBUG データ・セット名) がオブジェクト・プログラムに格納されるかどうかを制御します。 (423 ページの『TEST』)
 - PH08642: NUMCHECK: NUMCHECK オプションによって追加されていた冗長性検査が取り除かれ、パフォーマンスが向上します。いくつかの検査をコンパイル時に実行できます。NUMCHECK を指定することによって、コンパイラーは、

ランタイムではなくコンパイル時にいくつかのメッセージを生成できるようになります。(395 ページの『NUMCHECK』)

- PH09225: INITCHECK: INITCHECK オプションを OPTIMIZE(0) と一緒に指定できます。(382 ページの『INITCHECK』)
- PH11667: NUMCHECK(BIN): NUMCHECK(BIN) は、2 進数データ項目 (COMP、COMP-4、および USAGE BINARY) について、TRUNC(BIN) が有効な場合でも検査します。(395 ページの『NUMCHECK』)

IBM 提供 CICS 予約語テーブルの変更

- PI91589: 新しい COBOL ワードが IBM 提供の CICS 予約語テーブルに追加されました。(511 ページの『CICS 予約語テーブル』)

ステートメントの変更

- PI95081: 獲得する動的ストレージの場所を制御するために、新しい LOC(24|31) 句が ALLOCATE ステートメントに追加されました。これは DATA コンパイラー・オプションの影響をオーバーライドします。(43 ページの『ストレージとそのアドレス可能性』)

組み込み関数の強化

- PI97434: 以下の組み込み関数による国別データ項目の処理のサポートを追加します。
 - REVERSE
 - ULENGTH
 - UPOS
 - USUBSTR
 - UWIDTH
- (145 ページの『組み込み関数と国別データ』)

5 月のコンパイラー PTF (UI56120、UI56121、UI56122) の更新された組み込み関数 (REVERSE、ULENGTH、UPOS、USUBSTR、UWIDTH) を使用する場合は、これらのプログラムがリンクまたは実行されるすべてのシステム上の Language Environment に、5 月のランタイム PTF UI56043(V2R1)/UI56042(V2R2)/UI55861(V2R3) も適用する必要があります。

- PI99703:
 - 以下の組み込み関数が IBM 拡張として追加されました。
 - BIT-OF
 - HEX-OF

(133 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)』)

 - 以下の組み込み関数が 2014 COBOL 標準の一部として追加されました。
 - E
 - PI
 - TRIM

7 月のコンパイラー PTF (UI57342、UI57343、UI57344、UI57345) の新しい組み込み関数 (BIT-OF、HEX-OF、E、PI、TRIM) を使用する場合は、これらのプログラムがリンクまたは実行されるすべてのシステム上の Language Environment に、7 月のランタイム PTF UI57304(V2R1)/UI57303(V2R2)/UI57302(V2R3) も適用する必要があります。

- PH02183:

- 以下の組み込み関数が IBM 拡張として追加されました。

- BIT-TO-CHAR

- HEX-TO-CHAR

- (133 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)』)

- 以下の組み込み関数が 2014 COBOL 標準の一部として追加されました。

- ABS

- BYTE-LENGTH (138 ページの『データ項目の長さの検出』)

- EXP

- EXP10

- NUMVAL-F

- SIGN

- TEST-NUMVAL

- TEST-NUMVAL-C

- TEST-NUMVAL-F

9 月のコンパイラー PTF (UI58632、UI58633、UI58634) の新しい組み込み関数 (BIT-TO-CHAR、HEX-TO-CHAR、NUMVAL-F、TEST-NUMVAL、TEST-NUMVAL-C、TEST-NUMVAL-F) を使用する場合は、これらのプログラムがリンクまたは実行されるすべてのシステム上の Language Environment に、9 月のランタイム PTF UI58596(V2R1)/UI58595(V2R2)/UI58603(V2R3) も適用する必要があります。

バージョン 6 リリース 2

新規コンパイラー・オプション、変更されたコンパイラー・オプション、および削除されたコンパイラー・オプション

- 新しいコンパイラー・オプション:

- DEFINE (368 ページの『DEFINE』)

- INITCHECK (382 ページの『INITCHECK』)

- INLINE (384 ページの『INLINE』)

- NUMCHECK (395 ページの『NUMCHECK』)

- PARMCHECK (404 ページの『PARMCHECK』)

- 変更されたコンパイラー・オプション:

- AFP: デフォルト値が AFP(NOVOLATILE) に変更されました。 (351 ページの『AFP』)

- ARCH: 新しい上位レベルの ARCH(12) が受け入れられました。デフォルトは引き続き ARCH(7) です。 (352 ページの『ARCH』)

- MAXPCF: V6 コンパイラー・キャパシティーの増加を反映して、デフォルト値が MAXPCF(100000) に変更されました。(389 ページの『MAXPCF』)
- NOSTGOPT: 以前のバージョンでは、NOSTGOPT が有効になっていてもデータ項目を OPT(2) で最適化できました。このバージョンでは、OPT(2) が指定されていてもストレージやデータ項目が最適化されないように NOSTGOPT が変更されました。これは特に WORKING-STORAGE の目印に役立ちます。(421 ページの『STGOPT』)
- SSRANGE: コンパイラーによる参照変更長の検査方法を制御する新しいサブオプション ABD および SSRANGE が、SSRANGE コンパイラー・オプションに追加されました。(420 ページの『SSRANGE』)
- TEST: デバッグ能力を保持しながらディスク上のプログラム・オブジェクト・サイズを制御するために新規サブオプション SEPARATE および NOSEPARATE が TEST コンパイラー・オプションに追加されました。また、TEST(NODWARF)、TEST(SEPARATE)、NOTEST(DWARF,SOURCE) など、サブオプションの新しい組み合わせが TEST コンパイラー・オプションと NOTEST コンパイラー・オプションの両方でサポートされるようになりました。(423 ページの『TEST』)
- 以下のコンパイラー・オプションは削除されました。
 - ZONECHECK は非推奨ですが、互換性のために使用できるようにはなっています。これは NUMCHECK(ZON) で置き換えられます。(438 ページの『ZONECHECK』)

新規ステートメント

- 新規 JSON PARSE ステートメントは JSON テキストを COBOL データ・フォーマットに変換します。(617 ページの『第 29 章 JSON 入力処理』)

デバッグの変更

- TEST(SEPARATE) で、デバッグ能力を保持しながらモジュール・サイズを制御するためにデバッグ情報をサイド・ファイルに生成できるようになりました。(423 ページの『TEST』)

リストの変更

- Enterprise COBOL V5 より前の COBOL コンパイラーの場合と同様に、コンパイラー診断メッセージがリストの末尾に表示されるようになりました。
 - MD5 シグニチャーがプログラム・オブジェクトとデバッグ・データに追加されたため、プログラムが再コンパイルされた場合でもデバッグ・データを実行可能ファイルと突き合わせることができるようになりました。(484 ページの『例: MD5 シグニチャー』)
 - PPA4 の終わりに 3 つの新規フィールドが追加されました。
 - WORKING-STORAGE における最初のユーザー定義データ項目のオフセット。
 - WORKING-STORAGE におけるユーザー定義データ項目の合計長。
 - 外部データ項目が存在するかどうかを示すビット。
- (486 ページの『例: プログラム・プロログ域』)

使用可能度に関する機能強化

- cob2 コンパイラ呼び出しコマンドに関するヘルプ情報が追加されたため z/OS UNIX システム・サービス環境におけるコンパイラのユーザビリティが向上しました。(327 ページの『cob2 の構文およびオプション』)

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

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

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

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

関連タスク

『プログラムの識別』

6 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』

19 ページの『データの処理』

697 ページの『クラスの定義』

702 ページの『クラス・インスタンス・メソッドの定義』

739 ページの『OO アプリケーションの構造化』

プログラムの識別

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

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

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

```
IDENTIFICATION DIVISION.  
Program-ID.      Helloprog.  
Author.          A. Programmer.  
Installation.    Computing Laboratories.  
Date-Written.    09/04/2017.  
Date-Compiled.   09/08/2017.
```

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』
- 『プログラムを再帰的として識別する』
- 5 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。

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 オプションを指定してコンパイルしたプログラムには、RECURSIVE をコーディングする必要があります。

関連タスク

18 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』

567 ページの『再帰呼び出しの実行』

収容プログラムによってプログラムに呼び出し可能のマークを付ける

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

含まれているプログラムだけが COMMON 属性を持つことができます。

関連概念

564 ページの『ネストされたプログラム』

プログラムを初期状態に設定する

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

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

552 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』

555 ページの『静的呼び出しの作成』

555 ページの『動的呼び出しの作成』

関連参照

383 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、コンパイラおよび現行リリース・レベルの識別、コンパイルの日時、およびページ番号が入っています。

以下の例はこれら 5 つのエLEMENTを示しています。

PP 5655-EC6 IBM Enterprise COBOL for z/OS 6.2.0 P170724 Date 09/08/2017 Time 15:05:19
Page 1

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

注:

1. IBM Enterprise COBOL Value Unit Edition for 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。
2. IBM Enterprise COBOL Developer Trial for 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には、Developer Trial 製品の製品 ID と現行リリース・レベルが示されます。

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。

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

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

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

- プログラム内のファイルの特性を識別および記述する。
- ファイルを、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。

7 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』

9 ページの『シンボリック文字を定義する』

- 9 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 記入項目

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
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 ステートメントまたは環境変数を使用して定義することができます。 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 ファイルおよび行順次ファイルの場合、ACCESS MODE 節はオプションです。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

- 179 ページの『第 9 章 QSAM ファイルの処理』
- 209 ページの『第 10 章 VSAM ファイルの処理』
- 241 ページの『第 11 章 行順次ファイルの処理』
- 6 ページの『コンピューター環境の記述』

照合シーケンスの指定

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

これらの節では、英数字項目に対する以下の操作の照合シーケンスを指定します。

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

『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

259 ページの『代替照合シーケンスの選択』

165 ページの『国別 (UTF-16) データの比較』

例: 照合シーケンスの指定

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

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

```
IDENTIFICATION DIVISION.  
.  
.  
ENVIRONMENT DIVISION.  
  CONFIGURATION SECTION.  
    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。

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。

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。

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。

- MVS JCL の場合は DD ステートメント。
- TSO のもとでは、ALLOCATE コマンド。
- z/OS または z/OS UNIX 用の環境変数。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 項目と、システム・データ定義および FILE SECTION 内の FD 項目との関係を示しています。

- JCL DD ステートメント:

```
(1)
//OUTFILE DD DSNAME=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

14 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。

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

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

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 MASTER 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 DD 名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 名または環境変数名として 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 MASTER ASSIGN TO DA-3330-S-MASTERA
```

3 つのファイル、MASTER1、MASTER2、および MASTER3 が入力ファイルとなる可能性があるとして、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。

```
//MASTERA DD DSNAME=MY.MASTER1,.. .  
export MASTERA=DSN(MY.MASTER1),.. .
```

```
//MASTERA DD DSNAME=MY.MASTER2,.. .  
export MASTERA=DSN(MY.MASTER2),.. .
```

```
//MASTERA DD DSNAME=MY.MASTER3,.. .  
export MASTERA=DSN(MY.MASTER3),.. .
```

したがって、プログラムで MASTER を参照すると、それは現在 DD 名または環境変数名 MASTERA 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。

APPLY WRITE-ONLY を指定すると、バッファは、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のみ切り捨てられます。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。APPLY WRITE-ONLY 節が指定されているファイルに対しては、NOAWO コンパイラ・オプションの効力はありません。APPLY WRITE-ONLY 節の方が、NOAWO コンパイラ・オプションに優先します。

APPLY-WRITE ONLY 節を使用すると、入力ファイルがバッファ内のデータを処理せずに、レコード域を使用することがあります。これは、入力ファイルと出力ファイルの両方の処理に影響を与える可能性があります。

関連参照

355 ページの『AWO』

データの記述

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

これらのセクションは、以下のタイプのデータを定義するときに使用できます。

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

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

関連概念

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

関連タスク

『入出力操作でのデータの使用』

17 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。

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。レコードを WORKING-STORAGE に入れるには、READ-INTO ステートメントを使用します。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、RECORDING MODE 節でレコード形式を設定することができます。RECORDING MODE 節を省くと、コンパイラーは 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。BLOCK CONTAINS 節を省略すると、デフォルトとしてファイルはブロック化されません。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。BLOCK CONTAINS 1 RECORDS、または BLOCK CONTAINS n CHARACTERS (ここで、 n は 1 つの論理レコードの長さ (バイト単位) です) をコーディングすると、WRITE ステートメントにより、レコードは、バッファに入れられずに、即時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効です。

同じ実行単位内のプログラムは、共通ファイルを共有でき、また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

564 ページの『ネストされたプログラム』

関連タスク

585 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

14 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	注
FD	PROCEDURE DIVISION の入出力ステートメント (OPEN、CLOSE、READ、さらに、VSAM の場合は START および DELETE) 内で参照される <i>file-name</i>	SELECT 節内の <i>file-name</i> と一致しなければなりません。 <i>file-name</i> は、 <i>assignment-name</i> を介して <i>ddname</i> と関連付けられます。
BLOCK CONTAINS	物理レコードのサイズ	<p>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。</p> <p>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。 BLOCK CONTAINS 0 と指定した場合、または指定を行わなかった場合、ユーザーに代わってシステムが最適ブロック・サイズを決定します。</p> <p>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。</p> <p>VSAM: 構文が検査されますが、実行に影響はありません。</p>
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している必要があります。 <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している必要があります。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している必要があります。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

表 2. FILE SECTION 記入項目 (続き)

節	定義対象	注
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 節で識別されたとき、そのファイルが VS COBOL II、COBOL for OS/390® & VM、または IBM Enterprise COBOL for z/OS を使用して作成されたものでない場合は、対応する DD ステートメントで DCB=(OPTCD=Q. . .) または DCB=(RECFM=D. . .) をコーディングしなければならないことがあります。
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

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

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

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

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。

『例: ストレージ・セクション』

関連タスク

552 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』

603 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

701 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

15 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

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

関連タスク

18 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用』

18 ページの『ネストされたプログラムでのデータの共用』

18 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』

573 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。

呼び出し側プログラムでは、CALL . . . USING または INVOKE . . . USING ステートメントをコーディングしてデータを渡してください。

関連タスク

573 ページの『データの受け渡し』

578 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。GLOBAL 節は、データ名が、そのデータ名を定義するプログラム内に含まれているすべてのプログラムに使用可能であることを指定します (ただし、その中に含まれているプログラム自体がその名前を定義している場合を除きます)。

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

564 ページの『ネストされたプログラム』

再帰的またはマルチスレッド化されたプログラムでのデータの共用

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

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

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

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

関連概念

604 ページの『マルチスレッド化』

関連タスク

567 ページの『再帰呼び出しの実行』

607 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

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

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

PROCEDURE DIVISION.

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

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

PROCEDURE DIVISION USING *dataname*
PROCEDURE DIVISION USING BY REFERENCE *dataname*

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

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

PROCEDURE DIVISION USING BY VALUE *dataname*

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

PROCEDURE DIVISION RETURNING *dataname2*

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

PROCEDURE DIVISION USING *dataname* RETURNING *dataname2*

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

関連概念

20 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

578 ページの『LINKAGE SECTION のコーディング』

578 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

583 ページの『PROCEDURE DIVISION RETURNING . . . の使用』

795 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サブディビジョン。

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、プロシーチャー、またはプログラムのサブディビジョン。

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 処理の定義済み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、およびコンパイラー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

22 ページの『コンパイラー指示ステートメント』

22 ページの『範囲終了符号』

『命令ステートメント』

21 ページの『条件ステートメント』

24 ページの『宣言部分』

関連参照

PROCEDURE DIVISION の構成 (*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

『条件ステートメント』

22 ページの『範囲終了符号』

条件ステートメント

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

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

範囲区切りステートメントは、次のような方法で使用できます。

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

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

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

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

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

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

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

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

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

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

関連概念

20 ページの『命令ステートメント』
『範囲終了符号』

関連タスク

103 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)

コンパイラー指示ステートメント

コンパイラー指示ステートメントは、コンパイラーに、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。

コンパイラー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

445 ページの『第 18 章 コンパイラー指示ステートメント』
コンパイラー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

```

暗黙の範囲終了符号を使用すると、ステートメントの終わる場所が不明確になることがあります。その結果、ステートメントを意図せず終了し、プログラムのロジックを変えてしまうおそれがあります。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せず終了す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。

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

条件ステートメント内にネストされている命令ステートメントについての明示的範囲終了符号をコーディングする際には、注意が必要です。範囲終了符号が、その対象とするステートメントと対にな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

21 ページの『条件ステートメント』

20 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。プロシージャールの中に、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

453 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

43 ページの『ストレージとそのアドレス可能性』

関連タスク

『変数、構造、リテラル、および定数の使用』

29 ページの『データ項目への値の割り当て』

39 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

41 ページの『組み込み関数の使用』

42 ページの『テーブル (配列) とポインターの使用』

141 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ム言語では、変数、構造 (グループ項目)、リテラル、または定数としてデータを表すという概念は同じです。

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

関連タスク

『変数の使用』

26 ページの『データ項目とグループ項目の使用』

28 ページの『リテラルの使用』

28 ページの『定数の使用』

29 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。

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

```

Data Division.
01 Customer-Name           Pic X(20).
01 Original-Customer-Name Pic X(20).
. . .
Procedure Division.
    Move Customer-Name to Original-Customer-Name
. . .

```

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

関連概念

146 ページの『Unicode および言語文字のエンコード』

関連タスク

147 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

393 ページの『NSYMBOL』

155 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

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

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

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

```

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

```



```

Procedure Division.
    Move Customer-Name to Orig-Customer-Name
    Move Part-Name to Inventory-Part-Name
    . . .

```

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

```

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

```

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

関連概念

- 146 ページの『Unicode および言語文字のエンコード』
- 151 ページの『国別グループ』

関連タスク

- 147 ページの『COBOL での国別データ (Unicode) の使用』
- 152 ページの『国別グループの使用』

関連参照

- 14 ページの『FILE SECTION 記入項目』
 - 155 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。

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 は数値リテラルです。

```
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

146 ページの『Unicode および言語文字のエンコード』

関連タスク

149 ページの『国別リテラルの使用』

168 ページの『DBCS リテラルの使用』

関連参照

393 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、定数のための構造を定義しません。ただし、データ記述に VALUE 節をコーディングすることにより (INITIALIZE ステートメントをコーディングする代わりに)、初期値を使用してデータ項目を定義することができます。

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

上の例では、英数字データ項目と数値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

- 147 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。

以下に例を示します。

Move Spaces To Report-Header

関連タスク

- 150 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』

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。

表 3. プログラム内でのデータ項目の割り当て

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

表 3. プログラム内でのデータ項目の割り当て (続き)

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

『例: データ項目の初期化』

関連タスク

- 33 ページの『構造の初期化 (INITIALIZE)』
- 35 ページの『基本データ項目への値の割り当て (MOVE)』
- 36 ページの『グループ・データ項目への値の割り当て (MOVE)』
- 38 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)』
- 117 ページの『データ項目の結合 (STRING)』
- 120 ページの『データ項目の分割 (UNSTRING)』
- 37 ページの『算術結果への割り当て (MOVE または COMPUTE)』
- 128 ページの『データ項目の計算および置換 (INSPECT)』
- 141 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・カテゴリー内にあるすべての従属データ項目をどのように便利な方法で初期化できるかが分かります。

データ項目のブランクまたはゼロへの初期化:

INITIALIZE identifier-1

<i>identifier-1</i> PICTURE	<i>identifier-1</i> (初期化前)	<i>identifier-1</i> (初期化後)
9(5)	12345	00000
X(5)	AB123	bbbb ¹
N(3)	004100420031 ²	002000200020 ³
99XX9	12AB3	bbbb ¹

<i>identifier-1</i> PICTURE	<i>identifier-1</i> (初期化前)	<i>identifier-1</i> (初期化後)
XXBX/XX	ABbC/DE	<i>bbbb/bb</i> ¹
99.9CR	1234.5CR	<i>00.0bb</i> ¹
A(5)	ABCDE	<i>bbbbbb</i> ¹
+99.99E+99	+12.34E+02	+00.00E+00
1. 記号 <i>b</i> は、ブランク・スペースを表します。 2. 国別 (UTF-16) 文字「AB1」の 16 進表記。例では、 <i>identifier-1</i> が Usage National を持っている想定しています。 3. 国別 (UTF-16) 文字「 」(3 個のブランク・スペース) の 16 進表記。 <i>identifier-1</i> 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 <i>identifier-1</i> 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	<i>bbbbbbbbb</i> ¹	<i>bbbb</i> ABCD 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/ <i>bb</i> ¹	ABbC/DEF ¹	M/ <i>bb/bbb</i> 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		

数字編集 (USAGE DISPLAY) データ項目の初期化:

```

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

数字編集 (USAGE NATIONAL) データ項目の初期化:

```

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

『構造の初期化 (INITIALIZE)』

82 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

393 ページの『NSYMBOL』

構造の初期化 (INITIALIZE)

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

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

```

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

```

レコード	TRANSACTION-OUT (初期化前)	TRANSACTION-OUT (初期化後)
1	R00138300024000000000000000	b000000000000000000000000 ¹
2	R00139000048000000000000000	b000000000000000000000000 ¹
3	S00141000012000000000000000	b000000000000000000000000 ¹
4	C00138300000000042500000000	b000000000000000000000000 ¹
5	C00201000000000000010000000	b000000000000000000000000 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は、そのグループ項目に INITIALIZE ステートメントを適用することにより、リセットできます。以下の構造は、上記の構造と似ていますが、Unicode UTF-16 データを使用している点で異なります。

```

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

```

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

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

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

- SPACE は、英字、英数字、英数字編集、DBCS、カテゴリー国別、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

151 ページの『国別グループ』

関連タスク

82 ページの『テーブルの初期化 (INITIALIZE)』

152 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。

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

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

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

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

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

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

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

146 ページの『Unicode および言語文字のエンコード』

関連タスク

36 ページの『グループ・データ項目への値の割り当て (MOVE)』

156 ページの『国別 (Unicode) 表現との変換』

関連参照

359 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。

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

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

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

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

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

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

関連概念

146 ページの『Unicode および言語文字のエンコード』

151 ページの『国別グループ』

関連タスク

35 ページの『基本データ項目への値の割り当て (MOVE)』

152 ページの『国別グループの使用』

156 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。

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

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

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

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

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

言語環境プログラム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。言語環境プログラムのサービスは、標準の COBOL CALL ステートメントを介して使用することができ、それらが戻す値は CALL ステートメントのパラメーターに入れて渡されます。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プログラムのサービス 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

369 ページの『DIAGTRUNC』

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

言語環境プログラム 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。

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

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

画面ではなくファイルから読み取る場合は、次の変更を行います。

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

```
SYSIN is Names-Input
```

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 UNIX** 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 UNIX** 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。

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

値を **USAGE NATIONAL** データ項目に割り当てると、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

146 ページの『Unicode および言語文字のエンコード』

関連タスク

157 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

359 ページの『**CODEPAGE**』

画面上またはファイル内での値の表示 (DISPLAY)

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

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

上記の例で、データ項目 *Customer-Name* の内容が JOHNSON の場合、ステートメントは、次のメッセージをシステム論理出力装置に表示します。

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

データをシステム論理出力装置以外の宛先に書き込みたい場合には、SYSOUT 以外の宛先を指定した UPON 句を使用してください。例えば、次のステートメントは、SYSPUNCH DD ステートメントで指定されたファイルに書き込みます。

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

SYSPUNCH DD ステートメントを使用して、z/OS UNIX ファイル・システム 内のファイルを指定できます。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 下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

146 ページの『Unicode および言語文字のエンコード』

関連タスク

40 ページの『システム論理出力装置上でのデータの表示』

40 ページの『WITH NO ADVANCING の使用』

157 ページの『国別の英数字への変換 (DISPLAY-OF)』

501 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

359 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。

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。

OUTDD という DD 名が割り振られておらず、プログラムが z/OS UNIX 環境で実行されていない場合には、SYSOUT=* というデフォルトの DD が割り振られます。
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 UNIX 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)

DISPLAY ステートメントが失敗し、重大度 3 の言語環境プログラム条件が起きます。

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。

OUTDD と言語環境プログラムのランタイム・オプションの MSGFILE が同じ DD 名を指定していると、DISPLAY 出力と言語環境プログラム実行時診断の両方が言語環境プログラムのメッセージ・ファイルに経路指定されます。

関連タスク

542 ページの『環境変数の設定およびアクセス』

関連参照

403 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。

WITH NO ADVANCING 句を指定した場合に、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。

WITH NO ADVANCING を指定せず、出力が DD 名に送られる場合は、印刷制御文字 ' ' (スペース) が次の DISPLAY ステートメントの最初の出力位置に入れます (これはシングル・スペース出力を示します)。

```
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。これらの関数はストリングと数値を操作する機能を提供します。

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

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

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

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

COBOL ワード `function` は予約語ですが、関数名は予約されていません。関数名は、他のコンテキスト (データ項目の名前など) で使用できます。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返します。

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)` は数値を返します。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

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

テーブル (配列) とポインターの使用

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

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

ポインター・データ項目を使用して以下の操作を実行することができます。

- `CALL . . BY REFERENCE` ステートメントを使用してプログラム間でポインター・データ項目の受け渡しを行う。
- `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。
- `VALUE IS NULL` を使用して、それらが無効なアドレスを含むように初期化する。

ポインター・データ項目は、以下のことを行うために使用してください。

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

関連タスク

75 ページの『テーブルの定義 (OCCURS)』

568 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 境界より下でも 16 MB 境界より上でも構いませんが、2 GB の境界を超えないようにしてください。このストレージをアドレッシングするときは、24 ビットおよび 31 ビットの 2 つのアドレッシング・モードが使用可能です。

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・アドレッシングを使用して、無制限のストレージ をアドレッシングできます。したがって、16 MB 境界より上と 16 MB 境界より下の両方を含む、プログラムで使用可能なすべてのストレージをアドレッシングできます。

Enterprise COBOL は、z/OS の 64 ビットの仮想アドレッシング機能を直接には利用しませんが、31 ビット・アドレッシング・モードまたは 24 ビット・アドレッシング・モードで実行している COBOL アプリケーションは、64 ビット z/OS システムで完全にサポートされます。

アドレッシング・モード (AMODE) は、ユーザーのプログラムによってどのハードウェア・アドレッシング・モードがサポートされるかを示す属性です。24 ビット・アドレッシング、31 ビット・アドレッシング、24 ビットまたは 31 ビット・アドレッシングのいずれかを示す属性があります。これらの属性はそれぞれ AMODE 24、AMODE 31、および AMODE ANY になります。プログラム・オブジェクトおよび実行プログラムにはそれぞれ、AMODE 属性があります。Enterprise COBOL V5.1.1 オブジェクト・プログラムは、AMODE MIN (AMODE 24 が使用可能な場合) または AMODE 31 (それ以外の場合) です。44 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。

Enterprise COBOL は、言語環境プログラム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・オプションおよび言語環境プログラムのランタイム・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** プログラムが RENT を指定してコンパイルされている場合、WORKING-STORAGE データ、入出力バッファ、パラメーター・リスト用ストレージのロケーションに影響を及ぼすコンパイラー・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。
- Enterprise COBOL V4.2 以前のリリースでコンパイルされた
 - DATA(24) コンパイラー・オプションを使用してコンパイルされた
 - CICS での実行
 - COBOL プログラム (COBOL 5.1.0 を除く) およびアセンブラーのみを含むプログラム・オブジェクトにおける COBOL V5.1.1 以降のプログラム。そのプログラム・オブジェクト内には言語環境プログラムの言語間呼び出しがない。また、COBOL V5.1.0 プログラムもない。
 - メインエントリー・ポイントが COBOL V5 のプログラム・オブジェクト内にある、COBOL V5 プログラムである。この場合、COBOL が静的に C、C++、または PL/I にリンクしている状態で、プログラム・オブジェクトに言語環境プログラム言語間呼び出しが入ることができる。そのようなプログラム・オブジェクト内のすべての COBOL V5 プログラム (メイン・エントリー・ポイントではない場合も含め) では、WORKING-STORAGE はヒープ・ストレージから割り振られる。
 - COBOL V6.1 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 で実行されなければなりません。これは、COBOL V3 および V4 と同じ AMODE 24 制限セットです。

- 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
- INVOKE ステートメントなどのオブジェクト指向言語構文や、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 オプションを使用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、実行先は複数のスレッドや複数の PL/I タスクを持たないアプリケーションに限られます。

- 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 プログラムを含むエンクレーブ、または XPLINK ランタイム・オプションを使用して実行されるエンクレーブが含まれます。

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 V5.1.1 以降のバージョン、または Enterprise COBOL V4.2 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 V5.1.0 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 を使用して実行される COBOL プログラムは、バインダー・オプション RMODE(24) を使用してリンクされなければなりません。

RMODE の設定

RMODE オプションと RENT オプションは、プログラムの RMODE 属性を決定します。

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 オプションを指定した場合は、RMODE(24) または RMODE(AUTO) のいずれかのコンパイラー・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・コードに属性 RMODE ANY がある場合には、RMODE ANY または RMODE 24 でオブジェクト・コードをリンク・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で割り振られたパラメーターを AMODE 24 サブプログラムに渡してはなりません。31 ビット・アドレッシング・モードで実行しているプログラムの場合は、WORKING-STORAGE データおよびパラメーター・リストを強制的に境界より下に配置して、AMODE 24 で実行しているプログラムにデータを渡してください。

- コンパイラー・オプション RENT および DATA(24) を使用してコンパイルを行います。または、WORKING-STORAGE がヒープ上にある場合 (既出の HEAP オプションの説明を参照) は、HEAP(,,BELOW) ランタイム・オプションを使用して実行します。
- NORENT コンパイラー・オプションを使用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は、DATA コンパイラー・オプションおよび HEAP ランタイム・オプションによって、データ域のストレージ (WORKING-STORAGE SECTION や FD レコード域など) を 16MB 境界より下から獲得するのか、制限のないストレージから獲得するのかを制御します。プログラムが 16 MB 境界より上の仮想記憶域アドレスで 31 ビット・アドレス方式で実行される場合は、プログラムを RENT および RMODE(ANY) または RMODE(AUTO) でコンパイルしなければなりません。DATA オプションは、NORENT を指定してコンパイルされたプログラムに影響を与えません。

LOCAL-STORAGE データのストレージ

LOCAL-STORAGE データ項目のロケーションは、STACK ランタイム・オプションおよびプログラムの AMODE によって制御されます。LOCAL-STORAGE データ項目は、STACK(,,ANYWHERE) ランタイム・オプションが有効で、プログラムが AMODE 31 で実行されているときは、制限のない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 データのロケーションに影響を与えません。

外部データに対するストレージ

DATA コンパイラー・オプションは、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) のストレージ獲得方法に影響を与えるだけでなく、EXTERNAL データ用のストレージをどこから獲得するかにも影響を与えます。EXTERNAL データに必要なストレージは、次の条件が満たされる場合には、制限のないストレージから獲得されます。

- プログラムが、DATA(31) および RENT コンパイラー・オプションでコンパイルされている。
- 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。

これ以外の場合はすべて、EXTERNAL データ用のストレージは 16 MB 境界より下から獲得されます。ALL31(ON) ランタイム・オプションを指定する場合、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。

QSAM 入出力バッファ用ストレージ

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をどこに獲得するかにも影響する場合があります。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、以下の関連参照情報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 でストレージを獲得する方法に影響します。

- DATA(24) が有効で、ALLOCATE ステートメントの LOC 31 句が指定されていない場合、ALLOCATE は 16 MB 境界より下からストレージを獲得します。
- DATA(31) が有効で、ALLOCATE ステートメントの LOC 24 句が指定されていない場合、ALLOCATE は 16 MB 境界より上からストレージを獲得しようとします。

関連概念

558 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
(言語環境プログラム プログラミング・ガイド)

関連タスク

551 ページの『第 24 章 サブプログラムの使用』

573 ページの『第 25 章 データの共用』

関連参照

204 ページの『QSAM ファイル用のバッファの割り振り』

236 ページの『VSAM ファイル用のレコード域の割り振り』

366 ページの『DATA』

409 ページの『RENT』

410 ページの『RMODE』

788 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 (言語環境プログラム 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

第 3 章 数値および算術演算

一般的に、COBOL 数値データは、一連の 10 進数字の桁位置と見なすことができます。ただし、数値項目は、算術符号や通貨記号などの特殊な特性を持つこともできます。

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

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

関連タスク

『数値データの定義』

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『非互換データの検査 (数値のクラス・テスト)』

62 ページの『算術の実行』

72 ページの『通貨記号の使用』

数値データの定義

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

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

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

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

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

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

それ以外にコーディングできる特殊な意味を持つ文字は、次のとおりです。

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

次の例の s は、値が符号付きであることを意味します。

```
05 Price Pic s99v99.
```

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

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

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

このようにすれば、現在使用中のマシンとは非分離符号を格納するための規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。

57 ページの『例: 数値データおよび内部表現』

関連概念

54 ページの『数値データの形式』

805 ページの『付録 A. 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』
- 62 ページの『算術の実行』
- 151 ページの『国別数値データ項目の定義』

関連参照

- 60 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』
- 155 ページの『文字データの保管』
- 354 ページの『ARITH』
- 398 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。

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

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

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

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

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、上記のコードの Edited-price が USAGE NATIONAL を持っているときに、上記の最後のステートメントが以下のものである場合、プログラムを実行すると \$ 1,500.99 が表示されます。

Display Edited-price Upon Console

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

```
05 Price          Pic      9(5)v99.
05 Edited-price-D Pic    $99,999.99
Blank When Zero.
```

```
05 Edited-price-N Pic $99,999.99 Usage National  
Blank When Zero.
```

```
. . .  
Move 0 to Price  
Move Price to Edited-price-D  
Move Price to Edited-price-N  
Display Edited-price-D  
Display Edited-price-N 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。以下の例では、数字編集項目の値 (USAGE DISPLAY を持っているか USAGE NATIONAL を持っているかにかかわらず) は数値項目に移動します。

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

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

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

57 ページの『例: 数値データおよび内部表現』

関連タスク

39 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

『数値データの保管方法の制御』

49 ページの『数値データの定義』

62 ページの『算術の実行』

151 ページの『国別数値データ項目の定義』

156 ページの『国別 (Unicode) 表現との変換』

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。

次のような理由から、形式を制御する場合があります。

- 計算データ型を使用して実行する算術の方が、USAGE DISPLAY や USAGE NATIONAL データ型を使用して実行する算術より効率的である。
- パック 10 進数形式の方が、USAGE DISPLAY または USAGE NATIONAL データ型に比べ、1 桁当たりのストレージが少なく済む。
- パック 10 進数形式の方が 2 進数形式の場合よりも、DISPLAY または NATIONAL 形式との変換が効率的である。
- 浮動小数点形式は、位取りが大きく変化するような算術オペランドおよび結果を格納するのに最適であり、最大数の有効数字が維持される。
- データをあるマシンから別のマシンに移すときに、データ形式を保持する必要がある。

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

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

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

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

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

どの USAGE 節を使用して値の内部表現を制御するかにかかわらず、使用する PICTURE 節の規約および 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)。

57 ページの『例: 数値データおよび内部表現』

関連概念

54 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』

805 ページの『付録 A. 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』

62 ページの『算術の実行』

関連参照

59 ページの『変換および精度』

60 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

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

カテゴリー数値データ項目で USAGE NATIONAL が有効である場合、それぞれの 10 進数字ごとに 2 バイトのストレージが必要です。項目は UTF-16 形式で保管されます。USAGE NATIONAL 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中的に実行し、効率を優先させるのであれば、算術計算で使用するデータ項目には、COBOL の計算数値タイプを使用した方がよい場合もあります。

外部浮動小数点 (DISPLAY および NATIONAL) 項目

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

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

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

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

浮動小数点データ項目で USAGE NATIONAL が有効である場合、それぞれの PICTURE 文字位置 (使用されている場合、*v* を除く) は 2 バイトのストレージを占有しま

す。項目は国別文字 (UTF-16) として保管されます。 USAGE NATIONAL を持つ外部浮動小数点項目は、国別浮動小数点 データ項目と呼ばれます。

表示浮動小数点項目の既存の規則は、国別浮動小数点項目に適用されます。

以下の例では、Compute-Result-N は国別浮動小数点項目です。

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

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

2 進数は、付随する PICTURE 記述が 4 個以下の 10 進数字であれば 2 バイトを占め、5 個から 9 個の 10 進数字であれば 4 バイトを占め、10 個から 18 個の 10 進数字であれば 8 バイトを占めます。9 桁以上の 2 進数項目には、コンパイラによる余分な処理が必要となります。それらを SIZE ERROR 条件についてテストしたり、丸めたりするのは、他のタイプに比べて非効率的です。

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。しかし、これらは、USAGE COMP 項目とは異なり、PICTURE 節の 9 の数で暗黙指定される値に制限されるのではなく、固有 2 進数表現 (2、4、または 8 バイト) の容量までの大きさの値を含むことができます。

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

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

次の表は、COMP-5 データ項目に指定可能な値の範囲を示しています。

表 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 として記述したデータ項目は、ストレージ内では 2 進数ハーフワードとして表され、-327.68 から +327.67 までの範囲の値をサポートします。

VALUE 節のラージ・リテラル: COMP-5 項目の VALUE 節に指定されたりテラルは、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 コンパイラー・オプションの設定に関係なく、COMP-5 データ項目は、TRUNC(BIN) でコンパイルされたプログラムでは、2 進データのように動作します。

パック 10 進数 (COMP-3) 項目

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

内部浮動小数点 (COMP-1 および COMP-2) 項目

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

146 ページの『Unicode および言語文字のエンコード』

805 ページの『付録 A. 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』

151 ページの『国別数値データ項目の定義』

関連参照

155 ページの『文字データの保管』

430 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

次の表は、数値項目の内部表現を示しています。

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
2 進数	PIC S9999 BINARY	+ 1234	04 D2
	PIC S9999 COMP	- 1234	FB 2E
	PIC S9999 COMP-4		
	PIC S9999 COMP-5	+ 12345 ¹	30 39
		- 12345 ¹	CF C7
	PIC 9999 BINARY PIC 9999 COMP PIC 9999 COMP-4	1234	04 D2
	PIC 9999 COMP-5	60000 ¹	EA 60

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
内部 10 進数	PIC S9999 PACKED-DECIMAL	+ 1234	01 23 4C
	PIC S9999 COMP-3	- 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

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け取り側への割り当ての場合) 変換します。

変換とは、実際には、値を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。

データは演算の実行前に内部作業域に移動されて変換されるため、変換には通常、追加のストレージおよび処理時間が必要になります。さらに、結果を作業域に戻し、再度変換することが必要な場合もあります。

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

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

関連参照

『変換および精度』

60 ページの『ゾーンおよびパック 10 進数データのサイン表記』

変換および精度

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

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

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

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

精度が低下する変換

USAGE COMP-1 データ項目が 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。

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

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

関連概念

805 ページの『付録 A. 中間結果および算術精度』

ゾーンおよびパック 10 進数データのサイン表記

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

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

正: C、A、E、および F

負: D および 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

398 ページの『NUMPROC』

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

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

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

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

数値のクラス・テストでは、データ項目の内容が、そのデータ項目の PICTURE および USAGE にとって有効な値のセットと照合されます。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 のある外部 10 進数データ項目は、数字位置 (各バイトの下位 4 ビット) に 16 進値 X'0' から X'9' があり、各バイトの上位 4 ビットに有効なゾーン・コードがあり、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。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 が指定された場合は別個のバイトにあります。SIGN IS SEPARATE 節が使用されている場合、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: ZONEDATA(MIG|NOPFD) 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・コードが許容されますが、ゾーン 10 進数データ項目内の無効なデータ項目は数値のクラス・テストによって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. NUMCLS(PRIM) および有効な符号

	NUMPROC(NOPFD)	NUMPROC(PFD)
符号付き	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. NUMCLS(ALT) および有効な符号

	NUMPROC(NOPFD)	NUMPROC(PFD)
符号付き	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)

NUMCHECK(ZON,PAC) オプションを使用すれば、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) データ項目およびパック 10 進数 (COMP-3) データ項目に対して暗黙的な数値クラス・テストをコンパイラーで生成することもできます。この数値クラス・テストでは、データが検証されるだけでなく、NUMPROC コンパイラー・オプションに対して符号付きフィールドが検証され、NUMPROC(PFD) を使用できるかどうかを判別しやすくなります。詳細については、395 ページの『NUMCHECK』を参照してください。

関連参照

- 395 ページの『NUMCHECK』
- 398 ページの『NUMPROC』
- 440 ページの『ZONEDATA』

算術の実行

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

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

- 63 ページの『COMPUTE およびその他の算術ステートメントの使用』
- 63 ページの『算術式の使用』
- 64 ページの『数値組み込み関数の使用』

66 ページの『数学用の呼び出し可能サービスの使用』

67 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。

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

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

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

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

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

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

関連概念

70 ページの『固定小数点演算と浮動小数点演算の対比』

805 ページの『付録 A. 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)。

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、単一の数値リテラル、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

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

優先順位が同じレベルの演算子は、左から右へと評価されます。ただし、演算子とともに括弧を使用して、それらが評価される順序を変更することができます。括弧内の式は、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

70 ページの『固定小数点演算と浮動小数点演算の対比』

805 ページの『付録 A. 中間結果および算術精度』

数字組み込み関数の使用

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

数字組み込み関数は符号付き数値を戻し、一時数値データ項目として扱われます。

数字関数は、以下のカテゴリーに分類されます。

整数 整数を戻すもの。

浮動小数点

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

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

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

表 10. 数字組み込み関数

数値処理	日付および時刻	金融	数学	統計
LENGTH	CURRENT-DATE	ANNUITY	ABS	MEAN
MAX	DATE-OF-INTEGERS	PRESENT-VALUE	ACOS	MEDIAN
MIN	DATE-TO-YYYYMMDD		ASIN	MIDRANGE
NUMVAL	DAY-OF-INTEGERS		ATAN	RANDOM
NUMVAL-C	DAY-TO-YYYYDDD		COS	RANGE
NUMVAL-F	INTEGER-OF-DATE		E	STANDARD-DEVIATION
SIGN	INTEGER-OF-DAY		EXP	VARIANCE
TEST-NUMVAL	WHEN-COMPILED		EXP10	
TEST-NUMVAL-C	YEAR-TO-YYYY		FACTORIAL	
TEST-NUMVAL-F			INTEGER	
ORD-MAX			INTEGER-PART	
ORD-MIN			LOG	
			LOG10	
			MOD	
			PI	
			REM	
			SIN	
			SQRT	
			SUM	
			TAN	

68 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て評価すべきかを判別するときは例外です)。

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

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

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

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

数字組み込み関数の機能の多くは、言語環境プログラム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

70 ページの『固定小数点演算と浮動小数点演算の対比』

805 ページの『付録 A. 中間結果および算術精度』

関連参照

354 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。

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) の結果を返します。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) の結果を返します。

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 と同じ結果を返します。ARITH(EXTEND) を使用してコンパイルした場合は、CEESQACS が ACOS と同じ結果を返します。

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プログラム呼び出し可能サービス	対応する拡張精度言語環境プログラム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	CEERAN0	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、ARITH(EXTEND) を使用してコンパイルし、31 ビットの引数を渡した場合でも、長精度 (64 ビット) 浮動小数点結果を返します。			

RANDOM 組み込み関数と CEERAN0 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 CEERAN0 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プログラム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

使用して言語環境プログラム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プログラム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。言語環境プログラム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、言語環境プログラムが自動的にエラーを通知します。

関連概念

- 70 ページの『固定小数点演算と浮動小数点演算の対比』
- 805 ページの『付録 A. 中間結果および算術精度』

関連タスク

- 797 ページの『言語環境プログラム呼び出し可能サービスの使用』

関連参照

- 354 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プログラム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 コンパイラー・オプションの設定値によって異なることがあります。

INTDATE(LILIAN) が有効な場合、COBOL は第 1 日として 1582 年 10 月 15 日を使用します。言語環境プログラムは常に、第 1 日として 1582 年 10 月 15 日を使用します。INTDATE(LILIAN) を使用すると、COBOL 組み込み関数と言語環境プログラム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	言語環境プログラムの呼び出し可能サービス	結果
DATE-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYYMDD 付き)	互換性あり
DAY-OF-INTEGER	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	言語環境プログラムの呼び出し可能サービス	結果
INTEGER-OF-DATE	CEECBLDY	互換性あり
DATE-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYMMDD 付き)	非互換
DAY-OF-INTEGER	CEEDATE (ピクチャー・ストリング YYYYDDD 付き)	非互換
INTEGER-OF-DATE	CEEDAYS	非互換

関連タスク

797 ページの『言語環境プログラム呼び出し可能サービスの使用』

関連参照

385 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
```

日付および時刻

次の例は、今日から 90 日後の期限を計算する方法を示しています。CURRENT-DATE 関数から戻される最初の 8 文字は、日付を 4 桁の年、2 桁の月、および 2 桁の日

という形式 (YYYYMMDD) で表します。この日付がその整数値に変換されます。そのあと、この値に 90 が追加され、整数が YYYYMMDD 形式に再度変換されます。

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

金融

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

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

```
01 Series-Amt1     Pic 9(9)V99      Value 100.
01 Series-Amt2     Pic 9(9)V99      Value 200.
01 Series-Amt3     Pic 9(9)V99      Value 300.
01 Discount-Rate   Pic S9(2)V9(6)   Value .10.
01 Todays-Value    Pic 9(9)V99.
...
    Compute Todays-Value =
        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、各期間の長さと、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

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

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

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

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、算術演算に浮動小数点に関する上記のどの特性も含まれていない場合、コンパイラーはそれを固定小数点演算で評価します。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。以下に例を示します。

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

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

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

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

暗黙的な比較 (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
```

『例: 固定小数点計算および浮動小数点計算』

関連参照

814 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。

以下の記号の 1 つ以上を使用できます。

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

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

```

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

```

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

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

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

16 進リテラルを使用して通貨記号の値を表すことができます。ソース・プログラムのデータ入力方法では、対象とする文字を簡単に入力できない場合、16 進数リテラルを使用すると役立つことがあります。次の例は、通貨記号として使用される 16 進値 X'9F' を示しています。

```

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

```

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

365 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 "U".
Data Division.
WORKING-STORAGE SECTION.
01 Deposit-in-Euro      Pic S9999V99 Value 8000.00.
01 Deposit-in-CHF      Pic S99999V99.
01 Deposit-Report.
    02 Report-in-Franc Pic -FFFFF9.99.
    02 Report-in-Euro  Pic -UUUUU9.99.
01 EUR-to-CHF-Conv-Rate Pic 9V99999 Value 1.53893.
. . .
PROCEDURE DIVISION.
Report-Deposit-in-CHF-and-EUR.
    Move Deposit-in-Euro to Report-in-Euro
    Compute Deposit-in-CHF Rounded
        = Deposit-in-Euro * EUR-to-CHF-Conv-Rate
    On Size Error
        Perform Conversion-Error
    Not On Size Error
        Move Deposit-in-CHF to Report-in-Franc
        Display "Deposit in euro = " Report-in-Euro
        Display "Deposit in franc = " Report-in-Franc
    End-Compute
    Goback.
Conversion-Error.
    Display "Conversion error from EUR to CHF"
    Display "Euro value: " Report-in-Euro.
```

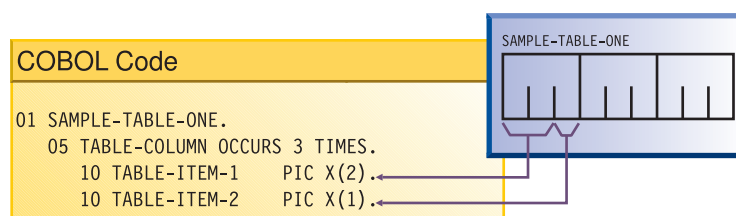
上記の例は、次のような表示出力を作成します。

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

この例で使用されている交換レートは例として使われているだけです。

第 4 章 テーブルの処理

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



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

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

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

テーブルは、プログラムの処理速度、特にレコードを探索する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

91 ページの『複合 OCCURS DEPENDING ON』

関連タスク

『テーブルの定義 (OCCURS)』

77 ページの『テーブルのネスト』

79 ページの『テーブル内の項目の参照』

81 ページの『テーブルに値を入れる方法』

87 ページの『可変長テーブルの作成 (DEPENDING ON)』

95 ページの『テーブルの探索』

98 ページの『テーブルのソート』

98 ページの『組み込み関数を使用したテーブル項目の処理』

100 ページの『無制限テーブルおよび無制限グループの処理』

783 ページの『テーブルの効率的処理』

テーブルの定義 (OCCURS)

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

```

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

```

上記の例では、table-name は、英数字グループ項目の名前です。テーブル・エレメント定義 (OCCURS 節が組み込まれている) は、テーブルを含むグループ項目に従属しています。OCCURS 節をレベル 01 記述で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 USAGE NATIONAL として記述する必要があります。また符合付きの従属数値データ項目は、暗黙的または明示的に SIGN IS SEPARATE 節で記述されている必要があります。

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

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

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

テーブルの二分探索 (SEARCH ALL) を行うには、OCCURS 節の ASCENDING または DESCENDING KEY 句をコーディングする必要があります。形式 2 SORT ステートメントを使用して、定義済みキーに従ってテーブルを順序付けし、それにより SEARCH ALL ステートメントでテーブルを検索可能にすることができます。テーブルがキーに従って順序付けられていない場合、SEARCH ALL は予測不能な結果を返すことに注意してください。

97 ページの『例: 二分探索』

関連概念

151 ページの『国別グループ』

関連タスク

77 ページの『テーブルのネスト』

79 ページの『テーブル内の項目の参照』

81 ページの『テーブルに値を入れる方法』

87 ページの『可変長テーブルの作成 (DEPENDING ON)』

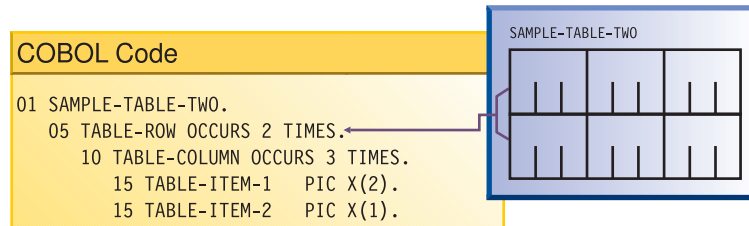
152 ページの『国別グループの使用』
97 ページの『二分探索 (SEARCH ALL)』
49 ページの『数値データの定義』

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*)

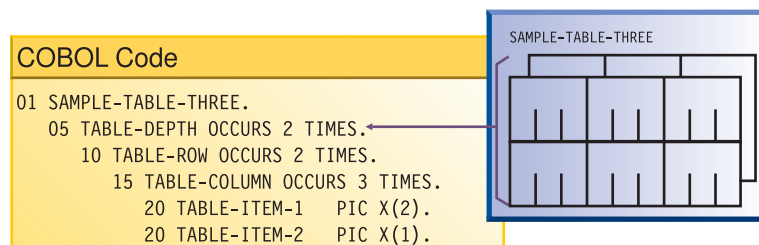
テーブルのネスト

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。



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

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



SAMPLE-TABLE-THREE の TABLE-DEPTH は 1 次元テーブルのエレメントで、2 回出現します。TABLE-ROW は 2 次元テーブルのエレメントで、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 次元テーブルのエレメントで、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。

78 ページの『例: 添え字付け』
78 ページの『例: 指標付け』

関連タスク

- 75 ページの『テーブルの定義 (OCCURS)』
- 79 ページの『テーブル内の項目の参照』
- 81 ページの『テーブルに値を入れる方法』
- 87 ページの『可変長テーブルの作成 (DEPENDING ON)』
- 95 ページの『テーブルの探索』
- 98 ページの『組み込み関数を使用したテーブル項目の処理』
- 783 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

- 79 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

80 ページの『索引付け』

テーブル内の項目の参照

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

項目を参照するには、次の 3 つの方法のいずれかを使用できます。

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

関連タスク

『添え字付け』

80 ページの『索引付け』

添え字付け

設定可能な最も小さい添え字値は 1 であり、これはテーブル・エレメントの最初のオカレンスを指します。1 次元テーブルでは、添え字は行番号に対応します。

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

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

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

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

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

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

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

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

78 ページの『例: 添え字付け』

関連タスク

『索引付け』

81 ページの『テーブルに値を入れる方法』

95 ページの『テーブルの探索』

783 ページの『テーブルの効率的処理』

索引付け

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

例えば、以下のコードにおける INX-A は指標名です。

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

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

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

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

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

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

SET ステートメントを使用すれば、上のステートメント SET INX-B TO INX-A のように、指標名に保管した値を指標データ項目に割り当てることができます。例え

ば、レコードを可変長テーブルにロードする際に、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。その後で、指標名を SEARCH ステートメントまたは関係条件ステートメントで使用できます。値を変更するには、PERFORM、SEARCH、または SET ステートメントを使用してください。

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

78 ページの『例: 指標付け』

関連タスク

79 ページの『添え字付け』

『テーブルに値を入れる方法』

95 ページの『テーブルの探索』

98 ページの『組み込み関数を使用したテーブル項目の処理』

783 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。

関連タスク

82 ページの『テーブルの動的なロード』

89 ページの『可変長テーブルのロード』

82 ページの『テーブルの初期化 (INITIALIZE)』

83 ページの『テーブルの定義時の値の割り当て (VALUE)』

90 ページの『可変長テーブルへの値の割り当て』

テーブルの動的なロード

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

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

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

85 ページの『例: PERFORM と添え字付け』

86 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。

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

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

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

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

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

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

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

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

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

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

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

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

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

30 ページの『例: データ項目の初期化』

関連タスク

33 ページの『構造の初期化 (INITIALIZE)』

『テーブルの定義時の値の割り当て (VALUE)』

90 ページの『可変長テーブルへの値の割り当て』

114 ページの『テーブルのループ処理』

26 ページの『データ項目とグループ項目の使用』

152 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。

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

84 ページの『それぞれのテーブル項目の個別の初期化』

84 ページの『グループ・レベルでのテーブルの初期化』

85 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』

33 ページの『構造の初期化 (INITIALIZE)』

それぞれのテーブル項目の個別の初期化

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

以下のコード例に示されている手法を使用します。

1. テーブルに入れられる項目を含むレコード (以下の Error-Flag-Table など) を定義します。
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。

```
*****
***          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

33 ページの『構造の初期化 (INITIALIZE)』

90 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。

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

```

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

```

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

```

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

```

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

関連参照

OCCURS 節 (*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。

```

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

90 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と、対応するエラー・メッセージが報告書印刷フィールドに移動されます。

```

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

```

```

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

. . .
PROCEDURE DIVISION.
. . .
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 (DEPENDING ON)

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

X OCCURS 1 TO 10 TIMES DEPENDING ON Y

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

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、DEPENDING ON 句のオブジェクトと OCCURS 節のサブジェクトの長さとの積です。

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

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

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

WORKING-STORAGE SECTION.

01 MAIN-AREA.

03 REC-1.

05 FIELD-1 PIC 9.

05 FIELD-2 OCCURS 1 TO 5 TIMES
DEPENDING ON FIELD-1 PIC X(05).

01 REC-2.

03 REC-2-DATA PIC X(50).

REC-1 (この場合は送り出し項目) を REC-2 に移動したい場合、REC-1 の長さは、FIELD-1 の現行値を使用して、移動の直前に決定されます。FIELD-1 の内容がその PICTURE 節と一致している場合 (すなわち、FIELD-1 がゾーン 10 進数項目を含んでいる場合)、REC-1 の実際の長さに基づいて移動は続行可能です。それ以外の場合は、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、最大のオカレンス回数を使用して決定されます。この例では、FIELD-2 の 5 回のオカレンスと FIELD-1 の合計で 26 バイトの長さになります。この場合、REC-1 を受け取り項目として参照する前に、ODO オブジェクト (FIELD-1) を設定する必要はありません。ただし、移動によって受信フィールドの ODO オブジェクトを有効に設定するために、送信フィールドの ODO オブジェクト (非表示) を 1 から 5 の間の有効な数値に設定しなければなりません。

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

01 MAIN-AREA

03 REC-1.

05 FIELD-1 PIC 9.

05 FIELD-3 PIC 9.

05 FIELD-2 OCCURS 1 TO 5 TIMES
DEPENDING ON FIELD-1 PIC X(05).

03 REC-2.

05 FIELD-4 OCCURS 1 TO 5 TIMES
DEPENDING ON FIELD-3 PIC X(05).

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

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

```
DATA DIVISION.
FILE SECTION.
FD  LOCATION-FILE
   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

91 ページの『複合 OCCURS DEPENDING ON』

関連タスク

90 ページの『可変長テーブルへの値の割り当て』

『可変長テーブルのロード』

94 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』

138 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

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

```
DATA DIVISION.
FILE SECTION.
FD  LOCATION-FILE
   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

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

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

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

```

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

```

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

関連タスク

83 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

複合 OCCURS DEPENDING ON (複合 ODO) にはいくつかのタイプがあります。複合 ODO は、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。
- 可変長エレメントを持つテーブル: OCCURS 節によって記述されたデータ項目に、OCCURS 節に DEPENDING ON 句を指定して記述された従属データ項目が含まれています。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

『例: 複合 ODO』

関連タスク

93 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』

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

関連参照

92 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.
```

定義: この例では、COUNTER-1 は ODO オブジェクト です。つまり、RECORD-1 の DEPENDING ON 節のオブジェクトです。RECORD-1 は ODO サブジェクト であると言われます。同様に、COUNTER-2 は、対応する ODO サブジェクトである RECORD-2 の ODO オブジェクトです。

上記の例で現れている複合 ODO のタイプは次のとおりです。

- (1) 可変位置項目: EMPLOYEE-NUMBER は、同じレベル 01 レコード内の可変長テーブルに続いている (ただし、従属してはいない) データ項目です。

- (2) 可変位置テーブル: TABLE-2 は、同じレベル 01 レコード内の可変長テーブルに続いている (ただし、従属してはいない) テーブルです。
- (3) 可変長エレメントを持つテーブル: TABLE-2 は、従属データ項目 RECORD-2 を含んでいるテーブルであり、この従属データ項目の出現回数は ODO オブジェクトの内容によって異なります。
- (4) 可変長エレメントを持つテーブルの指標名 INDX。
- (5) 可変長エレメントを持つテーブルのエレメント TABLE-ITEM。

長さの計算方法

各レコードの可変部分の長さは、その ODO オブジェクトと、その ODO サブジェクトの長さの積です。例えば、上記の複合 ODO 項目の 1 つに対して参照が行われるたびに、実際の長さ (使用する場合) は以下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 句を指定した OCCURS 節で記述されたデータ項目の後に、同じグループ内で 1 つ以上の非従属データ項目 (複合 ODO 形式) が続いている場合、ODO オブジェクトの値を変更すると、レコード内の複合 ODO 項目への後続の参照が影響を受けます。

以下に例を示します。

- 関係のある ODO 節を含んでいるグループのサイズは、ODO オブジェクトの新しい値を反映します。
- ODO オブジェクトの新しい値に基づいて、ODO サブジェクトを含んでいるグループへの移動 (MOVE) が行われます。
- ODO 節で記述された項目に続いている非従属項目の位置は、ODO オブジェクトの新しい値の影響を受けます。(非従属項目の内容を保持するためには、ODO オブジェクトの値が変更される前に、非従属項目を作業域に移動しておき、後でそれらを戻してください。)

ODO オブジェクトの値は、データをその ODO オブジェクトに移動するか、その ODO オブジェクトが含まれているグループに移動すると、変更される可能性があります。また、ODO オブジェクトが READ ステートメントのターゲットであるレコードに含まれている場合にも、その値が変更されることがあります。

関連タスク

『ODO オブジェクト値を変更する際の指標エラーを防止する』

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

ODO オブジェクト値を変更する際の指標エラーを防止する

テーブル内の従属データ項目の ODO オブジェクトの値を変更した後に、複合 ODO 指標名 (つまり、可変長エレメントを持つテーブルの指標名) を参照する場合には、注意してください。

ODO オブジェクトの値を変更すると、テーブルの長さが変わるため、関連する複合 ODO 指標のバイト・オフセットはもう有効ではありません。ですから次のような指標名への参照をコーディングした場合、予防措置を講じなければ、予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)
- SET *index-name* UP|DOWN BY *integer* 形式の SET ステートメント (形式 2)

この種のエラーを回避するためには、次の手順を行ってください。

1. 指標を整数データ項目に保存する。(これを行うと、暗黙の変換が行われます。整数項目は、指標のオフセットに対応するテーブル・エレメント出現番号を受け取ります。)
2. ODO オブジェクトの値を変更する。
3. ただちに整数データ項目から指標を復元する。(これを行うと、暗黙の変換が行われます。指標名は、整数項目でのテーブル・エレメント出現番号に対応するオフセットを受け取ります。オフセットは、その時点で有効なテーブルの長さに従って計算されます。)

次のコードは、ODO オブジェクト COUNTER-2 が変更される場合の指標名の保存方法と復元方法を示しています (91 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く可変オカレンス・テーブル内のエレメント数を増やす場合は、注意が必要です。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。

この種のエラーを回避するためには、次の手順を行ってください。

1. テーブルの後に続く可変位置データ項目を別のデータ域に保管する。
2. ODO オブジェクトの値を増分する。
3. データを新しいテーブル・エレメントに移動する (必要な場合)。
4. 可変位置データ項目を、それらを保管したデータ域から復元する。

次の例では、テーブル VARY-FIELD-1 にエレメントを追加しますが、このテーブルのエレメントの数は ODO オブジェクト CONTROL-1 に左右されます。

VARY-FIELD-1 の後には、非従属可変位置データ項目である GROUP-ITEM-1 が続いており、このエレメントがオーバーレイされる可能性があります。

WORKING-STORAGE SECTION.

```
01 VARIABLE-REC.
  05 FIELD-1                      PIC X(10).
  05 CONTROL-1                    PIC 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)									

VARY-FIELD-1 に 4 番目のエレメントを追加する場合は、次のようにコーディングすれば、VARY-FIELD-2 の最初の 5 バイトのオーバーレイを回避することができます。(GROUP-ITEM-2 は、可変位置 GROUP-ITEM-1 の一時記憶域として働きます。)

```
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(1)									
VARY-FIELD-1(2)									
VARY-FIELD-1(3)									
VARY-FIELD-1(4)									
VARY-FIELD-2(1)									
VARY-FIELD-2(2)									
VARY-FIELD-2(3)									

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。

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 の次数、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) の次数にすぎません。ただし、二分探索を行うには、テーブル項目をあらかじめソートしておく必要があります。

関連タスク

『逐次探索 (SEARCH)』

97 ページの『二分探索 (SEARCH ALL)』

逐次探索 (SEARCH)

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

WHEN 句の条件は、それらが指定された順番に評価されます。

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

それぞれの SEARCH ステートメントでは、テーブルの 1 つのレベル (1 つのテーブル・エレメント) だけを参照することができます。テーブルの複数のレベルを探索

するには、ネストされた SEARCH ステートメントを使用してください。ネストされたそれぞれの SEARCH ステートメントは、END-SEARCH で区切らなければなりません。

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

『例: 逐次探索』

関連参照

SEARCH ステートメント (*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あります (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 "AAAAAAA00" 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)
      = "A1234AAAAAAA00"
      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 "A1234AAAAAAA00"
RETURN-CODE = 0
```

二分探索 (SEARCH ALL)

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

SEARCH ALL ステートメントを使用してテーブルを探索するには、テーブルで OCCURS 節の ASCENDING または DESCENDING KEY 句 (あるいはその両方) を指定する必要があります。また ASCENDING および DESCENDING KEY 句で指定されたキーに基づいて既に順序付けられている必要があります。形式 2 SORT ステートメントを使用して、定義済みキーに従ってテーブルを順序付けし、それにより SEARCH ALL ステートメントでテーブルを検索可能にすることができます。テーブルがキーに従って順序付けられていない場合、SEARCH ALL は予測不能な結果を返すことに注意してください。

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

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

『例: 二分探索』

関連タスク

75 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。

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

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

このテーブルは、次のステートメントを使用して探索することができます。

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

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

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。

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*)

801 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。

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ます。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))
```

『例: 組み込み関数を使用したテーブルの処理』

関連タスク

41 ページの『組み込み関数の使用』

130 ページの『データ項目の変換 (組み込み関数)』

134 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。

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. WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION で、OCCURS DEPENDING ON オブジェクトを定義します。
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を呼び出します。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* のように指定します。
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、無制限グループを組み込み関数に対する引数として指定することはできません。

関連参照

『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: アンバインドされた表のストレージを割り振る/解放する

(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。以前の経験に基づいて OCCURS UNBOUNDED DEPENDING ON オブジェクトに任意の制限を設定し、文書の内容を処理するために文書を直接構文解析すると、より効率的な場合があります。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ジャは、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 "CEEGETST" 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

100 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

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

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

関連タスク

『プログラム・アクションの選択』

112 ページの『プログラム・アクションの繰り返し』

プログラム・アクションの選択

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

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

関連タスク

『アクションの選択項目のコーディング』

108 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

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

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

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

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

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

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

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
```

NEXT SENTENCE について詳しくは、「Enterprise COBOL for z/OS 言語解説書」の『IF ステートメント』を参照してください。

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

関連タスク

『ネストされた IF ステートメントの使用』

105 ページの『EVALUATE ステートメントの使用』

108 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 といいます。論理上は、ネストされた IF ステートメントの深さに制限はありません。

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

以下に、ネストされた IF ステートメントの疑似コードを示します。

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

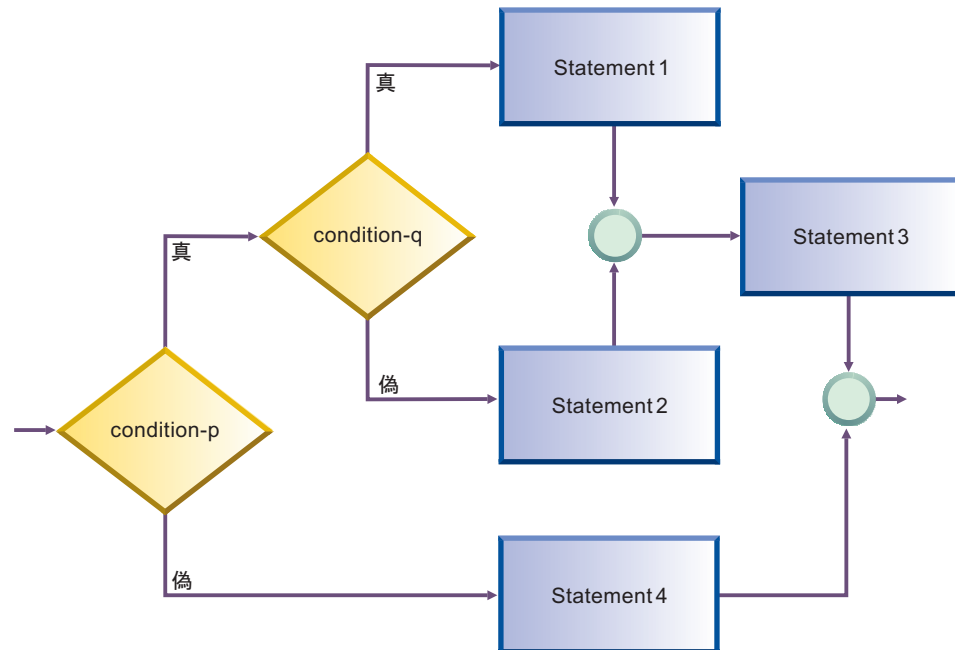
```

statement-3
ELSE
statement-4
END-IF

```

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

次の図は、上記の疑似コードの論理構造を示しています。



関連タスク

103 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

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

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

106 ページの『例: THRU 句を使用した EVALUATE』

107 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。

107 ページの『例: 複数の条件をテストする EVALUATE』

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

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

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

WHEN 句は、ソース・プログラムに現れている順序どおりにテストされます。そのため、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

103 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 は選択サブジェクトであり、1、2、および 3 THRU 6 は選択オブジェクトです。

```
EVALUATE CARPOOL-SIZE
  WHEN 1
    MOVE "SINGLE" TO PRINT-CARPOOL-STATUS
  WHEN 2
    MOVE "COUPLE" TO PRINT-CARPOOL-STATUS
  WHEN 3 THRU 6
```



```

        MOVE "SMALL GROUP" TO PRINT-CARPOOL STATUS
    WHEN OTHER
        MOVE "BIG GROUP" TO PRINT-CARPOOL STATUS
END-EVALUATE

```

次のネストされた IF ステートメントも同じロジックを表します。

```

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

```

例: 複数の **WHEN** 句を使用する **EVALUATE**:

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

```

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

```

次のネストされた IF ステートメントも同じロジックを表します。

```

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

```

例: 複数の条件をテストする **EVALUATE**:

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

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

```

Identification Division.
    Program-ID. MiniEval.
Environment Division.
    Configuration Section.

```

```

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。

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 の条件に当てはまるかどうかのテスト)
- 切り替え状況条件 (SPECIAL-NAMES 段落で名前を付けた UPSI スイッチの状況のテスト)
- 次のような複合条件
 - 否定条件。NOT (A IS EQUAL TO B) など
 - 複合条件 (論理演算子 AND または OR を組み合わせた条件)

関連概念

『スイッチおよびフラグ』

関連タスク

- 110 ページの『スイッチおよびフラグの定義』
- 111 ページの『スイッチとフラグのリセット』
- 61 ページの『非互換データの検査 (数値のクラス・テスト)』
- 165 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

- 110 ページの『スイッチおよびフラグの定義』
- 111 ページの『スイッチとフラグのリセット』

スイッチおよびフラグの定義

DATA DIVISION では、スイッチまたはフラグとして機能するレベル 88 項目を定義して、それらに分かりやすい名前を与えます。

フラグを持つ 2 つを超える値をテストするには、複数のレベル 88 項目を使用することによって、フィールドに複数の条件名を割り当ててください。

意味のある条件名が選択されており、かつそれらに割り当てられた値が論理値に関連付けられているならば、プログラムを読むときコードを追跡するのが容易になります。

『例: スイッチ』

『例: フラグ』

例: スイッチ

以下の例は、レベル 88 項目を使用してプログラム内のさまざまな 2 進値 (オン/オフ) 条件をテストする方法を示しています。

例えば、Transaction-File という名前の入力ファイルのファイル終了 (EOF) 条件をテストするには、データ定義を以下のように記述できます。

```
WORKING-STORAGE SECTION.  
01 Switches.  
    05 Transaction-EOF-Switch Pic X value space.  
        88 Transaction-EOF      value "y".
```

レベル 88 記述では、Transaction-EOF-Switch の値が「y」の場合に Transaction-EOF という名前の条件が有効になることが指定されています。PROCEDURE DIVISION 内で Transaction-EOF を参照することは、Transaction-EOF-Switch = "y" をテストすることと同じ条件を表します。例えば、次のステートメントによって報告書が印刷されるのは、Transaction-EOF-Switch が 'y' に設定されている場合に限られます。

```
If Transaction-EOF Then  
    Perform Print-Report-Summary-Lines  
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、マスター・ファイルを更新するプログラムを考えてみましょう。更新内容は、トランザクション・ファイルから読み取られます。ファイル内のレコードには、3 つの機能 (追加、変更、または削除) のうちどれを実行するかを指示するフィールドが含まれています。入力ファイルのレコード記述で、レベル 88 項目を使用して機能コード用のフィールドをコーディングします。

```
01 Transaction-Input Record  
    05 Transaction-Type          Pic X.  
        88 Add-Transaction       Value "A".  
        88 Change-Transaction    Value "C".  
        88 Delete-Transaction    Value "D".
```

これらの条件名をテストしてどの機能が実行されるかを判別するための、
PROCEDURE DIVISION 内のコードは、次のようになります。

```
Evaluate True
  When Add-Transaction
    Perform Add-Master-Record-Paragraph
  When Change-Transaction
    Perform Update-Existing-Record-Paragraph
  When Delete-Transaction
    Perform Delete-Master-Record-Paragraph
End-Evaluate
```

スイッチとフラグのリセット

プログラムの随所で、スイッチまたはフラグを、それらのデータ記述における元の値にリセットすることが必要になる場合があります。 そのためには、SET ステートメントを使用するか、データ項目をスイッチまたはフラグに移動するように定義します。

SET *condition-name* TO TRUE ステートメントを使用すると、スイッチまたはフラグは、データ記述の中で割り当てられた元の値に設定されます。 複数の値を持つレベル 88 項目の場合、SET *condition-name* TO TRUE は、最初の値 (次の例では A) を割り当てます。

```
88 Record-is-Active Value "A" "0" "S"
```

SET ステートメントと意味のある条件名を使用すれば、他のプログラマーにもコードが容易に追跡できるようになります。

『例: スイッチをオンに設定する』

112 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、次の例にある SET ステートメントは、ステートメント Move "y" to Transaction-EOF-Switch をコーディングした場合と同じ働きをします。

```
01 Switches
  05 Transaction-EOF-Switch Pic X Value space.
  88 Transaction-EOF      Value "y".
. . .
Procedure Division.
000-Do-Main-Logic.
  Perform 100-Initialize-Paragraph
  Read Update-Transaction-File
  At End Set Transaction-EOF to True
End-Read
```

次の例では、入力レコードのトランザクション・コードに基づいて、出力レコード内のフィールドに値を割り当てる方法を示します。

```
01 Input-Record.
  05 Transaction-Type      Pic X(9).
01 Data-Record-Out.
  05 Data-Record-Type      Pic X.
  88 Record-Is-Active      Value "A".
  88 Record-Is-Suspended   Value "S".
  88 Record-Is-Deleted     Value "D".
```

```

05 Key-Field                      Pic X(5).
. . .
Procedure Division.
  Evaluate Transaction-Type of Input-Record
    When "ACTIVE"
      Set Record-Is-Active to TRUE
    When "SUSPENDED"
      Set Record-Is-Suspended to TRUE
    When "DELETED"
      Set Record-Is-Deleted to TRUE
  End-Evaluate

```

例: スイッチをオフに設定する

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。

また、PERFORM ステートメントを使用すると、段落を実行し、その後で次の実行可能ステートメントに暗黙的に制御権を戻すようにすることもできます。実際には、この PERFORM ステートメントは、プログラムの異なる多くの部分から入ることができる閉じたサブルーチンをコーディングするための手段です。

PERFORM ステートメントはインラインまたはライン外にすることができます。

関連タスク

- 『インラインまたはライン外 PERFORM の選択』
- 114 ページの『ループのコーディング』
- 114 ページの『テーブルのループ処理』
- 115 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。

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。

『例: インライン PERFORM ステートメント』

例: インライン PERFORM ステートメント

この例は、必須の範囲終了符号と必須の END-PERFORM 句を持つインライン PERFORM ステートメントの構造を示しています。

```
Perform 100-Initialize-Paragraph
* The following statement is an inline PERFORM:
  Perform Until Transaction-EOF
    Read Update-Transaction-File Into WS-Transaction-Record
    At End
      Set Transaction-EOF To True
    Not At End
      Perform 200-Edit-Update-Transaction
      If No-Errors
        Perform 300-Update-Commuter-Record
      Else
        Perform 400-Print-Transaction-Errors
  * End-If is a required scope terminator
  End-If
  Perform 410-Re-Initialize-Fields
* End-Read is a required scope terminator
End-Read
End-Perform
```

ループのコーディング

PERFORM . . . TIMES ステートメントは、手順を指定された回数だけ実行する場合に使用します。

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

上記の例では、制御が PERFORM ステートメントに達すると、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。

PERFORM . . . UNTIL ステートメントは、選択した条件が満たされるまで手順を実行する場合に使用します。以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER . . . . UNTIL . . .  
PERFORM . . . [WITH TEST BEFORE] . . . UNTIL . . .
```

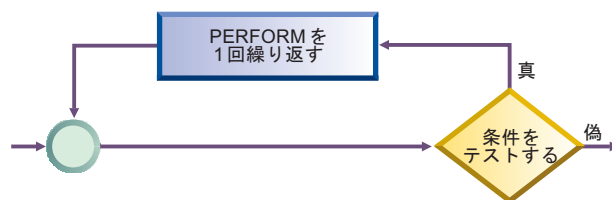
PERFORM . . . WITH TEST AFTER . . . UNTIL ステートメントは、手順を少なくとも 1 回実行し、その後テストしてからそれ以降を実行したい場合に使用します。このステートメントは、do-until 構造と同等です。



次の例では、暗黙の WITH TEST BEFORE 句によって do-while 構造が提供されます。

```
PERFORM 010-PROCESS-ONE-MONTH  
UNTIL MONTH GREATER THAN 12  
INSPECT . . .
```

制御が PERFORM ステートメントに達すると、条件 MONTH GREATER THAN 12 がテストされます。条件が満たされると、制御が INSPECT ステートメントに移ります。条件が満たされない場合には、010-PROCESS-ONE-MONTH が実行され、条件が再度テストされます。このサイクルは、条件が真になるまで継続されます。(プログラムを読みやすくするために、WITH TEST BEFORE 節をコーディングすることが必要な場合もあります。)



テーブルのループ処理

PERFORM . . . VARYING ステートメントを使用してテーブルを初期化することができます。この形式の PERFORM ステートメントでは、条件が満たされるまで変数が増加または減少され、テストされます。

そのあと、PERFORM ステートメントを使用して、テーブルを操作するループを制御することができます。以下のいずれかの形式を使用することができます。

```
PERFORM . . . WITH TEST AFTER . . . . VARYING . . . . UNTIL . . .  
PERFORM . . . [WITH TEST BEFORE] . . . . VARYING . . . . UNTIL . . .
```

以下のコードのセクションは、テーブル全体をループ処理して無効データがないか検査する例を示しています。

```
PERFORM TEST AFTER VARYING WS-DATA-IX  
    FROM 1 BY 1 UNTIL WS-DATA-IX = 12  
    IF WS-DATA (WS-DATA-IX) EQUALS SPACES  
        SET SERIOUS-ERROR TO TRUE  
        DISPLAY ELEMENT-NUM-MSG5  
    END-IF  
END-PERFORM  
INSPECT . . .
```

上記の PERFORM ステートメントに制御が達すると、WS-DATA-IX は 1 に設定され、PERFORM ステートメントが実行されます。その後、条件 WS-DATA-IX = 12 がテストされます。条件が真である場合には、制御が INSPECT ステートメントに渡ります。条件が偽である場合、WS-DATA-IX が 1 だけ増やされて、PERFORM ステートメントが実行され、条件が再度テストされます。この実行とテストのサイクルは、WS-DATA-IX が 12 になるまで継続されます。

上記のループは、項目 WS-DATA の 12 個のフィールドに関する入力検査を制御します。アプリケーションでは空のフィールドは許可されません。ですから、コードのセクションはループし、必要に応じてエラー・メッセージを発行します。

複数の段落またはセクションの実行

構造化プログラミングでは、通常、単一の段落を実行します。ただし、PERFORM . . . THRU ステートメントをコーディングすれば、段落グループ、単一セクション、またはセクション・グループを実行できます。

PERFORM . . . THRU ステートメントを使用するときには、段落 EXIT ステートメントをコーディングして、一連の段落のエンドポイントを明確に示してください。

関連タスク

98 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

COBOL は、String・データ項目に対して多種類の操作を実行するための言語構造体を提供します。

例えば、次のようなことが可能です。

- データ項目の結合または分割。
- 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

『データ項目の結合 (STRING)』

120 ページの『データ項目の分割 (UNSTRING)』

123 ページの『スル終了Stringの取り扱い』

124 ページの『データ項目のサブStringの参照』

128 ページの『データ項目の計算および置換 (INSPECT)』

130 ページの『データ項目の変換 (組み込み関数)』

134 ページの『データ項目の評価 (組み込み関数)』

141 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。

STRING ステートメントは、示された順序でデータを受信データ項目に転送します。STRING ステートメントでは、以下のものも指定します。

- 送信フィールド・セットごとの区切り文字。検出されると、これにより送信フィールドの転送は停止されます (DELIMITED BY 句)
- (オプション) すべての送信データが処理される前に受信フィールドが満杯になっている場合に実行する処置 (ON OVERFLOW 句)
- (オプション) データの転送先である受信フィールド内の左端文字位置を示す、整数データ項目 (WITH POINTER 句)

受信データ項目を編集項目にしてはなりません。また表示浮動小数点項目や国別浮動小数点項目にしてもなりません。受信データ項目が何を持っているかによって次のような違いが生じます。

- USAGE DISPLAY を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE DISPLAY を持っている必要があり、ステートメント内のそれぞれのリテラルは英数字でなければなりません。
- USAGE NATIONAL を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE NATIONAL を持っている必要があり、ステートメント内のそれぞれのリテラルは国別でなければなりません。
- USAGE DISPLAY-1 を持っている場合、ステートメント内のそれぞれの ID は (POINTER ID を除いて) USAGE DISPLAY-1 を持っている必要があり、ステートメント内のそれぞれのリテラルは DBCS でなければなりません。

STRING ステートメントによってデータが書き込まれる、受信フィールドの特定部分のみが変更されます。

『例: STRING ステートメント』

関連タスク

272 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.
```

STRING ステートメントが実行される前、POINTER フィールドの LINE-POS は値 4 を持っているので、データは受信フィールド RPT-LINE に移動されるとき、文字位置 4 から開始されます。位置 1 から 3 の文字は未変更のままです。

DELIMITED BY SIZE を指定している送信項目は、その全体が受信フィールドに移動されます。BAL-DUE は DEC-POINT で区切られているので、受信フィールドへの BAL-DUE の移動は、小数点 (DEC-POINT の値) が検出されると停止します。

STRING の結果

STRING ステートメントが実行されると、次の表に示されるように、項目は RPT-LINE に移動されます。

項目	位置
LINE-NO	4 - 8
スペース	9
CUST-INFO	10 - 59
INV-NO	60 - 65
スペース	66
DATE-DUE	67 - 74
スペース	75
BAL-DUE の小数点より前の部分	76 - 81

STRING ステートメントの実行後、LINE-POS の値は 82 であり、RPT-LINE は以下に示す値を持ちます。

Column					
4	10		60	67	76
↓	↓		↓	↓	↓
00001	J.B. SMITH	444 SPRING ST., CHICAGO, ILL.	A14275	10/22/76	\$2,336

データ項目の分割 (UNSTRING)

UNSTRING ステートメントは、送信フィールドを複数の受信フィールドに分割するために使用します。1 つの UNSTRING ステートメントを複数の MOVE ステートメントの代わりに使用できます。

UNSTRING ステートメントでは、以下のものを指定することができます。

- 区切り文字。区切り文字の 1 つが送信フィールドで検出されると、現行受信フィールドは受け取りを停止し、次のフィールド (ある場合) が受け取りを開始します (DELIMITED BY 句)
- 区切り文字用のフィールド。送信フィールドで区切り文字が検出されると、現行受信フィールドは受け取りを停止します (DELIMITER IN 句)
- 現行受信フィールドに入れられた文字の数を保管する整数データ項目 (COUNT IN 句)
- UNSTRING 処理が開始される送信フィールド内の左端文字位置を示す、整数データ項目 (WITH POINTER 句)
- 操作対象の受信フィールドの数の計算値を保管する、整数データ項目 (TALLYING IN 句)
- 送信データ項目の最後に達する前にすべての受信フィールドが満杯になった場合に実行する処置 (ON OVERFLOW 句)

送信データ項目および DELIMITED BY 句の区切り文字は、カテゴリ英字、英数字、英数字編集、DBCS、国別、または国別編集にする必要があります。

受信データ項目は、カテゴリ英字、英数字、数値、DBCS、または国別にすることができます。数値の受信データ項目は、ゾーン 10 進数または国別 10 進数にする必要があります。受信データ項目が何を持っているかによって次のような違いが生じます。

- USAGE DISPLAY を持っている場合、送信項目およびステートメント内のそれぞれの区切り文字項目は USAGE DISPLAY を持っている必要があり、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、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、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

121 ページの『例: UNSTRING ステートメント』

関連概念

146 ページの『Unicode および言語文字のエンコード』

関連タスク

272 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ー (*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 に入れられます。このレコードのフォーマットを以下のとおりです。

Column						
1	10	20	30	40	50	60
↓	↓	↓	↓	↓	↓	↓
ZYFOUR-PENNY-NAILS			707890/BBA	475120	00122	000379.50

UNSTRING ステートメントを以下に示します。

```
* Move subfields of INV-RCD to the subfields of DISPLAY-REC
* and WORK-REC:
  UNSTRING INV-RCD
    DELIMITED BY ALL SPACES OR "/" OR DBY-1
    INTO ITEM-NAME      COUNT IN CTR-1
      INV-NO            DELIMITER IN DLTR-1  COUNT IN CTR-2
      INV-CLASS
      M-UNITS          COUNT IN CTR-3
      FIELD-A
      DISPLAY-DOLS DELIMITER IN DLTR-2  COUNT IN CTR-4
    WITH POINTER CHAR-CT
    TALLYING IN  FLDS-FILLED
    ON OVERFLOW GO TO UNSTRING-COMPLETE.
```

UNSTRING ステートメントの実行前には POINTER フィールドである CHAR-CT の値は 3 であるので、INV-RCD 内の CONTROL-CHARS フィールドの 2 つの文字位置は無視されます。

UNSTRING の結果

この UNSTRING ステートメントを実行すると、以下のステップで処理が行われます。

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 (BBA) が INV-CLASS に入れられます。区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 34 のスペースは迂回されます。
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。
6. 桁 42 から 46 (00122) が FIELD-A に入れられ、領域内で右寄せされます。高位桁位置にはゼロ (0) が埋め込まれます。区切り文字は SPACE ですが、区切り文字の受取域としてフィールドが定義されていないので、桁 47 のスペースは迂回されます。
7. 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。
8. すべての受信フィールドに対して操作が行われたが、INV-RCD の 2 文字が検査されなかったため、ON OVERFLOW ステートメントが実行されます。UNSTRING ステートメントの実行は完了です。

UNSTRING ステートメントの実行後、フィールドには以下の値が入っています。

フィールド	値
DISPLAY-REC	707890 FOUR-PENNY-NAILS 000379
WORK-REC	475120000122BBA
CHAR-CT (POINTER フィールド)	55
FLDS-FILLED (TALLYING フィールド)	6

ヌル終了ストリングの取り扱い

さまざまな仕組みを使用して、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。

例えば、次のようなことが可能です。

- ヌル終了リテラル定数 (Z". . . ")。
- INSPECT ステートメントを使用して、ヌル終了ストリング内の文字数をカウントする。

```
MOVE 0 TO char-count
INSPECT source-field TALLYING char-count
                        FOR CHARACTERS
                        BEFORE X"00"
```

- UNSTRING ステートメントを使用して、ヌル終了ストリング内の文字をターゲット・フィールドに移動し、文字カウントを得る。

```
WORKING-STORAGE SECTION.
01 source-field          PIC X(1001).
01 char-count            COMP-5 PIC 9(4).
01 target-area.
   02 individual-char OCCURS 1 TO 1000 TIMES DEPENDING ON char-count
                        PIC X.
```

. . .

```
PROCEDURE DIVISION.
   UNSTRING source-field DELIMITED BY X"00"
                        INTO target-area
                        COUNT IN char-count

   ON OVERFLOW
      DISPLAY "source not null terminated or target too short"
END-UNSTRING
```

- SEARCH ステートメントを使用して、後続ヌルまたはスペース文字を見つける。検査するストリングを、単一文字からなるテーブルとして定義してください。
- ループのフィールドの各文字を検査する (PERFORM)。フィールドの各文字は、source-field (I:1) のような参照修飾子を使用して検査することができます。

124 ページの『例: ヌル終了ストリング』

関連タスク

579 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。

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、160 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、コロンの分離された 2 つの値を含むことができます。

1. サブストリングを開始したい文字の序数位置 (左からの)
2. (オプション) 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。

参照修飾子を使用すると、国別グループ、国別編集データ項目、数字編集データ項目、国別浮動小数点データ項目、および国別 10 進数データ項目を含め、USAGE NATIONAL データ項目のサブストリングを参照することができます。これらのデータ項目のいずれかを参照修飾した場合、結果はカテゴリ国別になります。例えば、次のように国別 10 進数データ項目を定義するとしましょう。

```
01 NATL-DEC-ITEM Usage National Pic 999 Value 123.
```

NATL-DEC-ITEM はカテゴリ数値なので、NATL-DEC-ITEM を算術式で使用できます。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。

127 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。

127 ページの『例: 参照修飾子としての組み込み関数』

参照修飾子の中のそれぞれの数値は少なくとも 1 の値でなければなりません。サブストリングの最後を越えて参照することがないよう、2 つの数値の合計が、データ項目の全長を 2 文字位置以上超えるようなことがあってはなりません。

左端の文字位置または長さ値が固定小数点の非整数の場合には、整数を作成するために切り捨てが行われます。浮動小数点の非整数の場合には、整数を作成するための丸めが行われます。

SSRANGE コンパイラー・オプションにより、範囲外の参照修飾子が検出され、実行時メッセージによって違反が示されます。

関連概念

『参照修飾子』

146 ページの『Unicode および言語文字のエンコード』

関連タスク

79 ページの『テーブル内の項目の参照』

関連参照

420 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、システムから返される時刻用のデータ定義のみです。以下に例を示します。

```
01 REFMOD-TIME-ITEM    PIC X(8).
```

次のコードは、時刻値を取り出して、拡張します。

```
ACCEPT REFMOD-TIME-ITEM FROM TIME.
DISPLAY "CURRENT TIME IS: "
* Retrieve the portion of the time value that corresponds to
* the number of hours:
REFMOD-TIME-ITEM (1:2)
" ."
* Retrieve the portion of the time value that corresponds to
* the number of minutes:
REFMOD-TIME-ITEM (3:2)
" ."
* Retrieve the portion of the time value that corresponds to
* the number of seconds:
REFMOD-TIME-ITEM (5:2)
```

127 ページの『例: 参照修飾子としての演算式』

127 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

38 ページの『画面またはファイルからの入力割り当て (ACCEPT)』

124 ページの『データ項目のサブストリングの参照』
147 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、以下のコード・フラグメントにより、Customer-Record のサブストリングがデータ項目 WS-name に移動されます。サブストリングは、実行時に決定されます。

```
05 WS-name      Pic x(20).  
05 Left-posn    Pic 99.  
05 I            Pic 99.  
...  
Move Customer-Record(Function Min(Left-posn I):Function Length(WS-name)) to WS-name
```

整数関数を使わなければならない位置で非整数関数を使用したい場合、INTEGER または INTEGER-PART 関数を使用して結果を整数に変換できます。以下に例を示します。

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

関連参照

INTEGER (*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。

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。

『例: INSPECT ステートメント』

関連概念

146 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。データ項目内に現れる先行ゼロ (0) の数は、累算されて COUNTR に入れられます。文字 C の最初のインスタンスの後に続く文字 A の最初のインスタンスは、文字 2 に置き換えられます。

```

77 COUNTR          PIC 9   VALUE ZERO.
01 DATA-2         PIC X(11).
. . .
  INSPECT DATA-2
    TALLYING COUNTR FOR LEADING "0"
    REPLACING FIRST "A" BY "2" AFTER INITIAL "C"

```

DATA-2 (実行前)	COUNTR (実行後)	DATA-2 (実行後)
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATHAM0000	0	CH2THAM0000

次の例では、INSPECT ステートメントは、データ項目 DATA-3 内の文字を検査して置き換えます。引用符 (") の最初のインスタンスより前にある各文字は、文字 0 で置き換えられます。

```

77 COUNTR          PIC 9   VALUE ZERO.
01 DATA-3         PIC X(8).
. . .
  INSPECT DATA-3
    REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE

```

DATA-3 (実行前)	COUNTR (実行後)	DATA-3 (実行後)
456"ABEL	0	000"ABEL
ANDES"12	0	00000"12
"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 ? (もしあれば) の最初のインスタンスより前にあるすべての文字が、小文字から大文字に変換されます。

```

01 DATA-4         PIC X(11).
. . .
  INSPECT DATA-4
    CONVERTING
      "abcdefghijklmnopqrstuvwxyz" TO
      "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。

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。

128 ページの『例: INSPECT ステートメント』

関連タスク 『大/小文字の変更 (UPPER-CASE、LOWER-CASE)』

131 ページの『逆順への変換 (REVERSE)』

131 ページの『数値への変換 (NUMVAL、NUMVAL-C、NUMVAL-F)』

132 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』

133 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)』

133 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。

```
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

38 ページの『画面またはファイルからの入力の割り当て (ACCEPT)』

39 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

逆順への変換 (REVERSE)

REVERSE 組み込み関数を使用して、ストリング内の文字の順序を反転させることができます。

Move Function Reverse(Orig-cust-name) To Orig-cust-name

例えば、上記ステートメントは、Orig-cust-name 中の文字の順序を逆にします。開始値が JOHNSONbbb であれば、ステートメントの実行後の値は bbbNOSNHQJ になります (b はブランク・スペースを表します)。

関連概念

146 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。

```
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、および NUMVAL-F は、互換モードでは長精度 (64 ビット) 浮動小数点値を戻し、拡張モードでは拡張精度 (128 ビット) 浮動小数点値を戻します。これらの関数への参照は、数値データ項目への参照を表します。

最大でも、正確に長精度浮動小数点に変換できるのは 15 桁の 10 進数字までです (変換および精度に関する以下の関連参照で説明されています)。NUMVAL、NUMVAL-C、または NUMVAL-F の引数が 15 桁を超える場合、ARITH(EXTEND) コンパイラー・オプションを指定して、引数の値を正確に表現できる拡張精度関数結果が戻されるようにすることをお勧めします。

NUMVAL、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。以下に例を示します。

```
+001.23
-300.00
```

しかし、NUMVAL 関数を使用する場合は、次のようにコーディングすることができます。

```
01 A   Pic x(10).
01 B   Pic S999V99.
...
      Accept A from Console
      Compute B = Function Numval(A)
```

入力は次のようにすることができます。

```
1.23
-300
```

関連概念

- 54 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』
- 146 ページの『Unicode および言語文字のエンコード』

関連タスク

- 156 ページの『国別 (Unicode) 表現との間の変換』

関連参照

- 59 ページの『変換および精度』
- 354 ページの『ARITH』

あるコード・ページから別のコード・ページへの変換

DISPLAY-OF 組み込み関数と NATIONAL-OF 組み込み関数をネストすると、任意のコード・ページを別の任意のコード・ページに簡単に変換できます。

例えば、次のコードでは、EBCDIC のストリングを ASCII のストリングに変換しています。

```
77 EBCDIC-CCSID PIC 9(4) BINARY VALUE 1140.
77 ASCII-CCSID  PIC 9(4) BINARY VALUE 819.
77 Input-EBCDIC PIC X(80).
77 ASCII-Output PIC X(80).
...
* Convert EBCDIC to ASCII
  Move Function Display-of
    (Function National-of (Input-EBCDIC EBCDIC-CCSID),
      ASCII-CCSID)
    to ASCII-output
```

関連概念

- 146 ページの『Unicode および言語文字のエンコード』

関連タスク

- 156 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。

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 進数字のストリング (「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 進ストリングの長さ (バイト単位) は、入力引数ストリングの長さ (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・ストリング形式で表す、人間が読み取れる 2 進数字のストリング (「0」または「1」) に変換するために使用できます。出力ビット・ストリングの長さ (バイト単位) は、入力引数ストリングの長さ (バイト単位) の 8 倍です。

例えば、FUNCTION BIT-OF('Hello, world!') は
「11001000100001011001001110010011100101100110101101000000101001101001011010011001100100111000010001011010」を返します。

注: 出力ストリングの最初の 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」で構成されます) またはビット文字ストリング (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。

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。

次はその例です。

```
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。

次はその例です。

```
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。

以下の組み込み関数を使用します。

- CHAR および ORD。これらは、プログラム内で使用される照合シーケンスを基準にして、整数および単一の英字または英数字を評価するためのものです。
- MAX、MIN、ORD-MAX、および ORD-MIN。これらは、USAGE NATIONAL データ項目を含む一連のデータ項目から最大項目または最小項目を検出するためのものです。
- LENGTH。これはデータ項目 (USAGE NATIONAL データ項目など) の長さを調べるためのものです。および BYTE-LENGTH。これは DBCS データ項目などのデータ項目のバイト単位の長さを調べるためのものです。

- WHEN-COMPILED。これは、プログラムがコンパイルされた日時を調べるためのものです。

関連概念

146 ページの『Unicode および言語文字のエンコード』

関連タスク

『照合シーケンスに関する単一文字の評価』

『最大または最小データ項目の検出』

138 ページの『データ項目の長さの検出』

139 ページの『コンパイルの日付の検出』

照合シーケンスに関する単一文字の評価

照合シーケンス内のある特定の文字 (英字または英数字) の序数位置を検出するには、引数として文字を持つ ORD 関数を使用します。ORD はその序数位置を表す整数を返します。

以下のように、データ項目の 1 文字のサブストリングを ORD への引数として使用することができます。

```
IF Function Ord(Customer-record(1:1)) IS > 194 THEN . . .
```

ある文字の照合シーケンスにおける序数位置がわかっていて、それに対応する文字を検索する場合には、CHAR でその整数の序数位置を引数として使用することができます。CHAR は、要求された文字を返します。以下に例を示します。

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

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。

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、引数の代数値が比較されます。

MAX および MIN 関数は、指定された引数の 1 つの内容を返します。例えば、プログラムに以下のデータ定義が含まれているとします。

```
05 Arg1 Pic x(10) Value "THOMASSON ".
05 Arg2 Pic x(10) Value "THOMAS    ".
05 Arg3 Pic x(10) Value "VALLEJO   ".
```

次のステートメントは、VALLEJObbb を Customer-record の最初の 10 文字位置に割り当てます (ここで b はブランク・スペースを表します)。

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

代わりに MIN を使用すると、THOMASbbbb が割り当てられます。

ORD-MAX および ORD-MIN 関数は、提供した引数のリストの中で最大値または最小値を持つ引数の (左からカウントした) 序数位置を表す整数を返します。上の例で ORD-MAX 関数を使用した場合、数字関数への参照が有効な場所がないため、コンパイラーはエラー・メッセージを出します。上の例と同じ引数を使用して、次のように ORD-MAX を使用できます。

```
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された場合、整数 3 を x に割り当てます。代わりに ORD-MIN を使用した場合には、整数 2 が返されます。Arg1、Arg2、および Arg3 が配列 (テーブル) の連続エレメントであったなら、上の例はもっと現実的なものになると思われます。

任意の引数に国別項目を指定する場合、すべての引数をクラス国別と指定する必要があります。

関連タスク

62 ページの『算術の実行』

98 ページの『組み込み関数を使用したテーブル項目の処理』

『英数字または国別関数によって戻される可変結果』

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。

次の例では、R3 に移動されるデータの量および COMPUTE ステートメントの結果は、R1 および R2 の値とサイズによって異なります。

```
01 R1 Pic x(10) value "e".
01 R2 Pic x(05) value "f".
01 R3 Pic x(20) value spaces.
01 L Pic 99.
. . .
Move Function Max(R1 R2) to R3
Compute L = Function Length(Function Max(R1 R2))
```

このコードの結果は次のようになります。

- R2 は R1 より大きいものと評価されます。

- スtring 'fbbbb' が R3 に移動されます (ここで *b* はブランク・スペースを表します)。 (R3 内の残りの文字位置にはスペースが埋められます。)
- L は値 5 と評価されます。

R1 が「e」ではなく「g」を含んでいたなら、コードの結果は以下のようになります。

- R1 は、R2 より大きいと評価されます。
- String 'gbbbbbbbb' が R3 に移動されます。 (R3 内の残りの文字位置にはスペースが埋められます。)
- 値 10 が L に割り当てられます。

プログラムが関数引数として国別データを使用する場合、同様に関数結果の長さおよび値が変化します。例えば、以下のコードは上のフラグメントと等しいですが、英数字データではなく国別データを使用します。

```
01 R1    Pic n(10) national value "e".
01 R2    Pic n(05) national value "f".
01 R3    Pic n(20) national value spaces.
01 L     Pic 99    national.
. . .
      Move Function Max(R1 R2) to R3
      Compute L = Function Length(Function Max(R1 R2))
```

このコードの結果は以下のようになります。これらは、国別文字についてのものであることを除けば、最初の結果のセットと似ています。

- R2 は R1 より大きいものと評価されます。
- String NX"0066 0020 0020 0020 0020" (国別文字 'fbbbb' と同等のもの。ここで、*b* はブランク・スペース) は、ここでは読みやすくするためスペースが挿入された 16 進表記で示されており、R3 に移動されます。R3 内の空白文字位置には、国別スペースが埋め込まれます。
- L は値 5 (R2 の国別文字位置の長さ) と評価されます。

英数字または国別の関数からの可変長出力を取り扱うことがあります。それに応じてプログラムを設計しなければなりません。例えば、書き込むレコードによって長さが異なる可能性があるときには、可変長レコード・ファイルの使用を考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

135 ページの『最大または最小データ項目の検出』

62 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。

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、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戻します。Function Length(Customer-name) または LENGTH OF Customer-name のどちらをコーディングしても、英数字項目の場合は同じ結果 (Customer-name のバイト単位の長さ) が戻されます。

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。

LENGTH および BYTE-LENGTH 関数は、算術式が許可される場所でのみ使用できます。ただし、LENGTH OF 特殊レジスターはさまざまなコンテキストで使用することができます。例えば、整数引数を受け入れる組み込み関数に対する引数として LENGTH OF 特殊レジスターを使用できます。(組み込み関数を LENGTH OF 特殊レジスターに対するオペランドとして使用することはできません。) LENGTH OF 特殊レジスターは、CALL ステートメントのパラメーターとして使用することもできます。

関連タスク

62 ページの『算術の実行』

87 ページの『可変長テーブルの作成 (DEPENDING ON)』

98 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。

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 は、実行時に国別文字データとして Unicode UTF-16 をサポートします。UTF-16 は、プレーン・テキストをエンコードするための一貫性のある効率的な方法を提供します。UTF-16 を使用すると、さまざまな国の言語で動作するソフトウェアを開発できます。

以下の COBOL 機能を使用して、国別データを処理するプログラムのコーディングおよびコンパイルを行います。

- データ型およびリテラル:
 - 文字データ型。USAGE NATIONAL 節や、カテゴリー国別、国別編集、または数字編集のデータを定義する PICTURE 節で定義します。
 - 数値データ型。USAGE NATIONAL 節や、数値データ項目 (国別 10 進数項目) または外部浮動小数点データ項目 (国別浮動小数点項目) を定義する PICTURE 節で定義します。
 - リテラル接頭部 N または NX で指定される国別リテラル
 - 形象定数 ALL *national-literal*
 - 形象定数 QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO。これらは、国別文字コンテキストで使用されるときには国別文字 (UTF-16) 値を持ちます。
- COBOL ステートメント。COBOL ステートメントおよび国別データに関する以下の関連参照に示されています。
- 組み込み関数
 - NATIONAL-OF は、英数字または 2 バイト文字セット (DBCS) 文字ストリングを USAGE NATIONAL (UTF-16) に変換します。
 - DISPLAY-OF は、国別文字ストリングを選択されたコード・ページ (EBCDIC、ASCII、EUC、または UTF-8) の USAGE DISPLAY に変換します。
 - その他の組み込み関数は、組み込み関数および国別データに関する以下の関連参照に示されています。
- GROUP-USAGE NATIONAL 節。USAGE NATIONAL データ項目のみを含み、ほとんどの操作でカテゴリ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。

関連概念

- 146 ページの『Unicode および言語文字のエンコード』
- 151 ページの『国別グループ』

関連タスク

- 147 ページの『COBOL での国別データ (Unicode) の使用』
- 156 ページの『国別 (Unicode) 表現と間の変換』
- 159 ページの『UTF-8 データの処理』
- 164 ページの『中国語 GB 18030 データの処理』
- 165 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』
- 817 ページの『付録 B. 2 バイト文字セット (DBCS) データの変換』

関連参照

- 『COBOL ステートメントと国別データ』
- 145 ページの『組み込み関数と国別データ』
- 359 ページの『CODEPAGE』
- 393 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・カテゴリーおよび PICTURE 規則 (*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)

COBOL ステートメントと国別データ

PROCEDURE DIVISION および以下の表に示すコンパイラ指示ステートメントで、国別データを使用できます。

表 15. COBOL ステートメントと国別データ

COBOL ステートメント	国別にできるもの	コメント	詳細
ACCEPT	ID-1、 ID-2	<i>identifier-1</i> が CODEPAGE コンパイラ・オプションで指定されたネイティブ・コード・ページから変換されるのは、入力 が CONSOLE からのものである場合のみです。	38 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。		63 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</i> 、 <i>identifier-3</i> 、 <i>identifier-4</i> 、 <i>identifier-5</i> ; <i>literal-2</i> 、 <i>literal-3</i>		573 ページの『データの受け渡し』

表 15. COBOL ステートメントと国別データ (続き)

COBOL ステートメント	国別にできるもの	コメント	詳細
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。		63 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>		445 ページの『第 18 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。	39 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。		63 ページの『COMPUTE およびその他の算術ステートメントの使用』
INITIALIZE	<i>identifier-1</i> . REPLACING 句の <i>identifier-2</i> または <i>literal-1</i> 。	REPLACING NATIONAL または REPLACING NATIONAL-EDITED を指定する場合、 <i>identifier-2</i> または <i>literal-1</i> は、 <i>identifier-1</i> への移動における送信オペランドとして有効でなければなりません。	30 ページの『例: データ項目の初期化』
INSPECT	ID はすべてリテラルです。(TALLYING 整数データ項目である <i>identifier-2</i> は USAGE NATIONAL を持つことができます。)	これらのいずれか (TALLYING ID である <i>identifier-2</i> を除く) が USAGE NATIONAL を持っている場合、すべてが国別でなければなりません。	128 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> 。		716 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。	617 ページの『第 29 章 JSON 入力の処理』
MERGE	マージ・キー	COLLATING SEQUENCE 句は適用されません。	257 ページの『ソートまたはマージ基準の設定』

表 15. COBOL ステートメントと国別データ (続き)

COBOL ステートメント	国別にできるもの	コメント	詳細
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。	35 ページの『基本データ項目への値の割り当て (MOVE)』 36 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。		63 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。	97 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。	257 ページの『ソートまたはマージ基準の設定』
STRING	ID はすべてリテラルです。(POINTER 整数データ項目である <i>identifier-4</i> は USAGE NATIONAL を持つことができます。)	<i>identifier-3</i> (受信データ項目) が国別である場合、すべての ID およびリテラル (POINTER ID である <i>identifier-4</i> を除く) は国別でなければなりません。	117 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。		63 ページの『COMPUTE およびその他の算術ステートメントの使用』
UNSTRING	ID はすべてリテラルです。 (<i>identifier-6</i> および <i>identifier-7</i> (それぞれ COUNT および TALLYING 整数データ項目) は USAGE NATIONAL を持つことができます。)	<i>identifier-4</i> (受信データ項目) が USAGE NATIONAL を持っている場合、送信データ項目およびそれぞれの区切り文字は USAGE NATIONAL を持っている必要があり、それぞれのリテラルは国別でなければなりません。	120 ページの『データ項目の分割 (UNSTRING)』

表 15. COBOL ステートメントと国別データ (続き)

COBOL ステートメント	国別にできるもの	コメント	詳細
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)		675 ページの『第 32 章 XML 出力の生成』
XML PARSE	<i>identifier-1</i> (XML 文書)	XML-NTEXT 特殊レジスターには、構文解析時に国別文字文書フラグメントが入ります。XML-NNAMESPACE および XML-NNAMESPACE-PREFIX 特殊レジスターには、国別文字に関連した名前空間 ID および名前空間接頭部 (存在する場合) が含まれています。	627 ページの『第 31 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』

147 ページの『COBOL での国別データ (Unicode) の使用』

165 ページの『国別 (UTF-16) データの比較』

関連参照

359 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. 組み込み関数と国別文字データ

	組み込み関数	関数のタイプ	詳細の参照先
I	BIT-OF	英数字	BIT-OF (<i>Enterprise COBOL for z/OS</i> 言語解説書)
I	BYTE-LENGTH	整数	138 ページの『データ項目の長さの検出』
	DISPLAY-OF	英数字	157 ページの『国別の英数字への変換 (DISPLAY-OF)』
I	HEX-OF	英数字	HEX-OF (<i>Enterprise COBOL for z/OS</i> 言語解説書)
	LENGTH	整数	138 ページの『データ項目の長さの検出』
	LOWER-CASE、UPPER-CASE	国別	130 ページの『大/小文字の変換 (UPPER-CASE、LOWER-CASE)』
I	NUMVAL、NUMVAL-C、NUMVAL-F	数字	131 ページの『数値への変換 (NUMVAL、NUMVAL-C、NUMVAL-F)』
	MAX、MIN	国別	135 ページの『最大または最小データ項目の検出』

表 16. 組み込み関数と国別文字データ (続き)

組み込み関数	関数のタイプ	詳細の参照先
ORD-MAX、ORD-MIN	整数	135 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 言語解説書</i>)
TEST-NUMVAL、TEST-NUMVAL-C、 TEST-NUMVAL-F	整数	<ul style="list-style-type: none"> TEST-NUMVAL (<i>Enterprise COBOL for z/OS 言語解説書</i>) TEST-NUMVAL-C (<i>Enterprise COBOL for z/OS 言語解説書</i>) TEST-NUMVAL-F (<i>Enterprise COBOL for z/OS 言語解説書</i>)
TRIM	英数字または国別文字	TRIM (<i>Enterprise COBOL for z/OS 言語解説書</i>)
ULENGTH	整数	ULENGTH (<i>Enterprise COBOL for z/OS 言語解説書</i>)
UPOS	整数	UPOS (<i>Enterprise COBOL for z/OS 言語解説書</i>)
USUBSTR	英数字または国別文字	USUBSTR (<i>Enterprise COBOL for z/OS 言語解説書</i>)
USUPPLEMENTARY	整数	USUPPLEMENTARY (<i>Enterprise COBOL for z/OS 言語解説書</i>)
UVALID	整数	UVALID (<i>Enterprise COBOL for z/OS 言語解説書</i>)
UWIDTH	整数	UWIDTH (<i>Enterprise COBOL for z/OS 言語解説書</i>)

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』

147 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*)

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関係付ける明確な規則セットです。各コード・ページには名前があり、文字セットを表現するための記号を設定した一種のテーブルとなっています。それぞれの記号は、固有のビット・パターン、すなわちコード・ポイ

ントを持ちます。コード・ページにはそれぞれ、コード化文字セット ID (CCSID) があり、1 から 65,536 までの値をとります。

Unicode には、*Unicode Transformation Format (UTF)* と呼ばれるいくつかのエンコード・スキーム (UTF-8、UTF-16、UTF-32 など) があります。Enterprise COBOL では、国別リテラルおよび USAGE NATIONAL を持つデータ項目の表現として、ビッグ・エンディアン・フォーマットの UTF-16 (CCSID 1200) を使用します。

UTF-8 は、ASCII インバリアント文字 a から z、A から Z、0 から 9、および特殊文字 (' @ , . UTF-16 は、これらの文字を NX'00nn' として表します (ここで、X'nn' は ASCII での文字表現です)。

例えば、ストリング「ABC」は、UTF-16 では NX'004100420043' として表されます。UTF-8 では、「ABC」は X'414243' として表されます。

1 つ以上のエンコード・ユニットを使用して、コード化文字セットから文字を表します。UTF-16 の場合、エンコード・ユニットは 2 バイトのストレージを使用します。任意の EBCDIC、ASCII、または EUC コード・ページで定義された文字はいずれも、国別データ表現に変換されたときに 1 つの UTF-16 エンコード・ユニットで表現されます。

クロスプラットフォームに関する考慮事項: Enterprise COBOL 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

156 ページの『国別 (Unicode) 表現との間の変換』

関連参照

155 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。

次の国別データ型が使用可能です。

- 国別データ項目 (カテゴリー国別、国別編集、および数字編集)
- 国別リテラル
- 国別文字としての形象定数
- 数値データ項目 (国別 10 進数および国別浮動小数点)

加えて、明示的または暗黙的に USAGE NATIONAL を持つデータ項目のみを含んでおり、ほとんどの操作でカテゴリー国別基本項目と同様に振る舞う、国別グループを定義できます。

これらの宣言は、必要なストレージ量に影響します。

関連概念

146 ページの『Unicode および言語文字のエンコード』

151 ページの『国別グループ』

関連タスク

『国別データ項目の定義』

149 ページの『国別リテラルの使用』

150 ページの『国別文字形象定数の使用』

151 ページの『国別数値データ項目の定義』

152 ページの『国別グループの使用』

156 ページの『国別 (Unicode) 表現と間の変換』

165 ページの『国別 (UTF-16) データの比較』

関連参照

155 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。

以下のカテゴリーの国別データ項目を定義できます。

- 国別
- 国別編集
- 数字編集

カテゴリー国別データ項目を定義するには、1 つ以上の PICTURE 記号 N のみを含む PICTURE 節をコーディングしてください。

国別編集データ項目を定義するには、以下のそれぞれの記号の少なくとも 1 つを含む PICTURE 節をコーディングしてください。

- 記号 N
- 単純追加編集記号 B、0、または /

クラス国別の数字編集データ項目を定義するには、数字編集項目を定義する PICTURE 節 (例えば、-*\$999.99*) をコーディングしてください。また、USAGE NATIONAL 節をコーディングしてください。USAGE NATIONAL を持つ数字編集データ項目は、USAGE DISPLAY を持つ数字編集項目を使用するのと同様に使用できます。

また、PICTURE 節により数値として定義された基本項目に BLANK WHEN ZERO 節をコーディングすれば、データ項目を数字編集として定義することもできます。

PICTURE 節をコーディングしたが、1 つ以上の PICTURE 記号 N のみを含むデータ項目用に USAGE 節をコーディングしなかった場合、コンパイラ・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

393 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。

次のいずれかの表記を使用できます。

- **N"character-data"**
- **N'character-data'**

オプション **NSYMBOL(DBCS)** を指定してコンパイルする場合、リテラル接頭部文字 **N** は国別リテラルではなく **DBCS** リテラルを指定します。

国別リテラルを 16 進値として指定するには、接頭部 **NX** を使用します。 次のいずれかの表記を使用できます。

- **NX"hexadecimal-digits"**
- **NX'hexadecimal-digits'**

次の **MOVE** ステートメントのそれぞれは、国別データ項目 **Y** を文字「AB」の UTF-16 値に設定します。

```
01 Y pic NN usage national.  
...  
  Move NX"00410042" to Y  
  Move N"AB"         to Y  
  Move "AB"          to Y
```

国別リテラルを必要とするコンテキストで英数字 16 進数リテラルを使用しないでください。そのような使用法は誤解を招きやすくなります。例えば、次のステートメントの場合も、UTF-16 文字「AB」(16 進ビット・パターン C1C2 ではない) が、**Y** に移動されます (**Y** は、**USAGE NATIONAL** で定義されます)。

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

国別リテラルは、**SPECIAL-NAMES** 段落で使用したり、プログラム名として使用したりすることはできません。国別リテラルは、**METHOD-ID** 段落のオブジェクト指向メソッドを指定したり、**INVOKE** ステートメント内のメソッド名を指定したりするのに使用できます。

関連タスク

28 ページの『リテラルの使用』

関連参照

393 ページの『NSYMBOL』

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

国別文字形象定数の使用

国別文字を必要とするコンテキストでは、形象定数の *ALL national-literal* を使用できます。ALL 国別リテラル は、国別リテラルを構成する連続したエンコード・ユニットの連結によって生成される、ストリングの全部または一部を表します。

国別文字を必要とするコンテキスト (MOVE ステートメント、暗黙移動、または、国別オペランドを持つ比較条件など) では、形象定数 QUOTE、SPACE、HIGH-VALUE、LOW-VALUE、または ZERO を使用できます。こうしたコンテキストでは、形象定数は国別文字 (UTF-16) 値を表します。

国別文字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・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・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。

制約事項: HIGH-VALUE または HIGH-VALUE から割り当てられた値は、あるデータ表現から別のデータ表現への値の変換 (例えば、USAGE DISPLAY と USAGE NATIONAL の間の変換) が起こるような仕方では使用してはなりません。X'FF' (EBCDIC 照合シーケンスが使用されているときの、英数字コンテキストでの HIGH-VALUE の値) は有効な EBCDIC 文字を表しませんし、NX'FFFF' は有効な国別文字を表しません。このような値を別の表現に変換すると、置換文字が使用されることとなります (X'FF' でも NX'FFFF' でもなくなります)。次の例を見てください。

```
01 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

- 156 ページの『国別 (Unicode) 表現と間の変換』
- 165 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

国別文字 (UTF-16) で表される数値データを保持するデータ項目を、USAGE NATIONAL 節で定義します。国別 10 進数項目および国別浮動小数点項目を定義できます。

国別 10 進数項目を定義するには、記号 9、P、S、および V のみを含む PICTURE 節をコーディングしてください。PICTURE 節が S を含んでいる場合、その項目で SIGN IS SEPARATE 節が有効でなければなりません。

国別浮動小数点項目を定義するには、浮動小数点項目を定義する PICTURE 節をコーディングしてください (例えば、+99999.9E-99)。

国別 10 進数項目は、ゾーン 10 進数項目と同じように使用できます。国別浮動小数点項目は、表示浮動小数点項目と同じように使用できます。

関連タスク

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、PIC N(*m*) (ここで、*m* はグループ内の国別 (UTF-16) 文字の数です) として記述されたカテゴリー国別基本項目として再定義されているかのように処理されます。

ただし、国別グループに対する操作の中には (英数字グループに対する一部の操作の場合と同様に)、グループ・セマンティクスが適用されるものがあります。そのような操作 (例えば、MOVE CORRESPONDING や INITIALIZE) は、国別グループ内の基本項目を認識または処理します。

可能な場合、USAGE NATIONAL 項目を含んでいる英数字グループではなく、国別グループを使用してください。国別グループでの国別データの処理の場合、英数字グループ内の国別データの処理と比較して、いくつかの利点があります。

- 国別グループを、USAGE NATIONAL を持つもっと長いデータ項目に移動させると、受信項目に国別文字が埋め込まれます。これに対して、国別文字を含む英数字グループを、国別文字を含むもっと長い英数字グループに移動させると、埋め込みには英数字スペースが使用されます。その結果、データ項目の取り扱いを誤ることがあります。
- 国別グループを、USAGE NATIONAL を持つもっと短いデータ項目に移動させると、国別グループは国別文字境界で切り捨てられます。これに対して、国別文字を含む英数字グループを、国別文字を含むもっと短い英数字グループに移動させると、国別文字の 2 バイト間で切り捨てが起こります。
- 国別グループを国別編集または数字編集項目に移動させると、グループの内容が編集されます。これに対し、英数字グループを編集項目に移動させた場合、編集は行われません。

- 国別グループを、STRING、UNSTRING、または INSPECT ステートメントのオペランドとして使用した場合、次のようになります。
 - グループの内容は、1 バイト文字としてではなく、国別文字として処理されます。
 - TALLYING および POINTER オペランドは、国別文字の論理レベルで作動します。
 - 国別グループ・オペランドは、他の国別オペランド・タイプの混じり合ったものと一緒にサポートされます。

これに対し、これらのコンテキストで国別文字を含む英数字グループを使用した場合、文字はバイトごとに処理されます。結果として、取り扱いが無効になったり、データの破壊が起こることがあります。

USAGE NATIONAL グループ: グループ項目では、グループ内のそれぞれの基本データ項目の USAGE の便利な省略表現として、グループ・レベルで USAGE NATIONAL 節を指定できます。ただし、このようなグループは国別グループではなく、英数字グループであり、多数の操作 (移動や比較など) において USAGE DISPLAY の基本データ項目のように振る舞います (ただし、データの編集や変換は行われません)。

関連タスク

36 ページの『グループ・データ項目への値の割り当て (MOVE)』
 117 ページの『データ項目の結合 (STRING)』
 120 ページの『データ項目の分割 (UNSTRING)』
 128 ページの『データ項目の計算および置換 (INSPECT)』
 『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*)

国別グループの使用

グループ・データ項目を国別グループとして定義するには、グループ・レベルで項目の GROUP-USAGE NATIONAL 節をコーディングしてください。グループは、明示的または暗黙的に USAGE NATIONAL を持つデータ項目のみを含むことができます。

以下のデータ記述項目は、レベル 01 グループとその従属グループが国別グループ項目であることを指定します。

```
01 Nat-Group-1    GROUP-USAGE NATIONAL.
  02 Group-1.
    04 Month      PIC 99.
    04 DayOf      PIC 99.
    04 Year        PIC 9999.
  02 Group-2      GROUP-USAGE NATIONAL.
    04 Amount     PIC 9(4).99  USAGE NATIONAL.
```

上の例で、Nat-Group-1 は国別グループであり、その従属グループ Group-1 および Group-2 も国別グループです。Group-1 に関して GROUP-USAGE NATIONAL 節が暗黙指定され、Group-1 の従属項目に関して USAGE NATIONAL が暗黙指定されています。Month、DayOf、および Year は国別 10 進数項目であり、Amount は USAGE NATIONAL を持つ数字編集項目です。

英数字グループ内の国別グループは、次の例のようにして従属させることができます。

```
01 Alpha-Group-1.
  02 Group-1.
    04 Month PIC 99.
    04 DayOf PIC 99.
    04 Year PIC 9999.
  02 Group-2 GROUP-USAGE NATIONAL.
    04 Amount PIC 9(4).99.
```

上の例で、Alpha-Group-1 および Group-1 は英数字グループであり、Group-1 内の従属項目に関して USAGE DISPLAY が暗黙指定されています。(Alpha-Group-1 が USAGE NATIONAL をグループ・レベルで指定した場合、Group-1 の従属項目のそれぞれについて USAGE NATIONAL が暗黙指定されることになります。しかし、Alpha-Group-1 および Group-1 は (国別グループではなく) 英数字グループになり、移動や比較などの操作時に英数字グループと同様の振る舞いを示します。) Group-2 は国別グループであり、数字編集項目 Amount に関して USAGE NATIONAL が暗黙指定されています。

国別グループ内で英数字グループを従属させることはできません。国別グループ内の基本項目はすべて明示的または暗黙的に USAGE NATIONAL として記述されている必要があり、国別グループ内のグループ項目はすべて明示的または暗黙的に GROUP-USAGE NATIONAL として記述されている必要があります。

関連概念

151 ページの『国別グループ』

関連タスク

『国別グループを基本項目として使用』

154 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。

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

- 36 ページの『グループ・データ項目への値の割り当て (MOVE)』
- 167 ページの『国別データ・オペランドと英数字グループ・オペランドの比較』
- 『国別グループをグループ項目として使用』

関連参照

MOVE ステートメント (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。

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-quanN   Pic 99  Value 10.  
   . . .  
   Initialize Group-OneN Replacing Numeric Data By 15
```

上の Group-OneN の Trans-quanN のみが数値なので、Trans-quanN 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、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に処理されます。
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。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に初期化されます。
名前の修飾	国別グループ項目の名前を使用して、国別グループ内の基本データ項目の名前および従属グループ項目の名前を修飾してください。	英数字グループの場合と同じ修飾の規則に従ってください。

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 (続き)

言語機能	国別グループ項目の使用法	コメント
RENAMES 節の THROUGH 句	THROUGH 句で国別グループ項目を指定するには、英数字グループ項目の場合と同じ規則を使用してください。	結果は英数字グループ項目です。
XML GENERATE ステートメントの FROM 句	XML GENERATE ステートメントの規則に従って、グループとして処理する国別グループ項目を FROM 句で指定してください。	国別グループ内の基本項目は、英数字グループ内の USAGE NATIONAL を持つ基本項目と同様に処理されます。

関連タスク

- 33 ページの『構造の初期化 (INITIALIZE)』
- 82 ページの『テーブルの初期化 (INITIALIZE)』
- 35 ページの『基本データ項目への値の割り当て (MOVE)』
- 36 ページの『グループ・データ項目への値の割り当て (MOVE)』
- 138 ページの『データ項目の長さの検出』
- 675 ページの『XML 出力の生成』
- 519 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. エンコード方式と英数字、DBCS、および国別データのサイズ

特性	DISPLAY	DISPLAY-1	NATIONAL
文字エンコード・ユニット	1 バイト	2 バイト	2 バイト
コード・ページ	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
<ol style="list-style-type: none"> 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。 <ul style="list-style-type: none"> • 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 			

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。

MOVE ステートメントを使用すれば、暗黙的に、英字、英数字、DBCS、または整数データを国別データに変換できます。暗黙変換は、英数字データ項目を USAGE NATIONAL 付きデータ項目と比較する IF ステートメントなど、他の COBOL ステートメントでも行われます。

組み込み関数 NATIONAL-OF および DISPLAY-OF をそれぞれ使用して、明示的に国別データ項目に変換したり、国別データ項目から変換したりすることができます。これらの組み込み関数を使用すると、CODEPAGE コンパイラー・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

『英数字、DBCS、および整数から国別への変換 (MOVE)』

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

157 ページの『国別の英数字への変換 (DISPLAY-OF)』

158 ページの『デフォルト・コード・ページのオーバーライド』

165 ページの『国別 (UTF-16) データの比較』

関連参照

359 ページの『CODEPAGE』

158 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)

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

次の種類のデータをカテゴリ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
- DBCS
- USAGE DISPLAY の整数
- USAGE DISPLAY の数字編集

同様に次の種類のデータを、USAGE NATIONAL を持つ数字編集データ項目に移動させることができます。

- 英数字
- 表示浮動小数点 (USAGE DISPLAY の浮動小数点)
- USAGE DISPLAY の数字編集
- USAGE DISPLAY の整数

国別データへの移動に関する完全な規則については、MOVE ステートメントに関する関連参照を参照してください。

例えば、以下の MOVE ステートメントは、英数字リテラル "AB" を国別データ項目 UTF16-Data に移動させます。

```
01  UTF16-Data  Pic N(2) Usage National.  
      . . .  
      Move "AB" to UTF16-Data
```

上記の MOVE ステートメントの実行後、UTF16-Data には、英数字「AB」の国別表現である NX'00410042' が入ります。

USAGE NATIONAL を持つ受信データ項目で埋め込みが必要な場合、デフォルトの UTF-16 スペース文字 (NX'0020') が使用されます。切り捨てが必要な場合、それは国別文字位置の境界で行われます。

関連タスク

- 35 ページの『基本データ項目への値の割り当て (MOVE)』
- 36 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、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。CODEPAGE コンパイラー・オプションを使用して有効になっているコード・ページとは異なるコード・ページでソースがエンコードされている場合は、ソース・コード・ページを 2 番目の引数として指定してください。

- 158 ページの『例: 国別データとの間の変換』

関連タスク

- 159 ページの『UTF-8 データの処理』
- 164 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

359 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**)

2 番目の引数として指定されたコード・ページで表現される英数字 (USAGE DISPLAY) 文字ストリングへ国別データを変換するには、DISPLAY-OF 組み込み関数を使用してください。

2 番目の引数を省略した場合、出力のコード・ページは、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 文字を結合した EBCDIC または ASCII のいずれかのコード・ページを指定すると、返されるストリングには SBCS 文字と DBCS 文字が混在することがあります。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。

『例: 国別データとの間の変換』

関連タスク

159 ページの『UTF-8 データの処理』

164 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。

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、暗黙的変換を伴う操作 (国別データ項目への割り当てまたは国別データ項目との比較など) で、関数結果を使用しないでください。このような操作は、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

359 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 の言語環境プログラム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 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

359 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、CCSID 00037 (米国英語) から UTF-16 への変換に基づいて、Data-in-Greek を Unicode 表現に変換します。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
```

UTF-8 データの処理

UTF-8 データを処理するには、最初に UTF-8 データを国別データ項目の UTF-16 に変換します。国別データを処理したあとで、データを出力のために再び UTF-8 に変換します。この変換には、それぞれ、組み込み関数 NATIONAL-OF および DISPLAY-OF を使用します。UTF-8 データにはコード・ページ 1208 を使用します。

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。このプロパティーでは、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、『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。

1. 関数 NATIONAL-OF を使用して、ASCII または EBCDIC スtring を国別 String (UTF-16) に変換します。
2. 関数 DISPLAY-OF を使用して、国別 String を UTF-8 に変換します。

次の例は、ギリシャ語の EBCDIC データを UTF-8 に変換しています。

```
01 Greek-EBCDIC pic X(10) value "αβγδεζηθ".
01 UnicodeString pic N(10).
01 UTF-8-String pic X(20).
   Move function National-of(Greek-EBCDIC, 00875) to UnicodeString
   Move function Display-of(UnicodeString, 01208) to UTF-8-String
```

使用上の注意: 参照変更を使用して UTF-8 でエンコードされたデータを参照する場合には注意してください。UTF-8 文字のエンコードでは、1 文字に使用されるバイト数が異なります。マルチバイト文字を分割する可能性がある処理は避けてください。

関連タスク

- 124 ページの『データ項目のサブStringの参照』
- 156 ページの『国別 (Unicode) 表現との間の変換』
- 653 ページの『UTF-8 でエンコードされた XML 文書の構文解析』
- 『組み込み関数を使用した UTF-8 エンコード・データの処理』

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。

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String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StringのサブString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-suppl.  
  Compute u = function USUPPLEMENTARY(UTF-8-testStr)  
  If u not = 0  
  Display ' checkUTF-8-suppl 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'  
    ', starting at byte ' u ','  
  Display ' is a Unicode supplementary code point, '  
    '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-suppl` からの出力は以下のとおりです。

```
checkUTF-8-suppl 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

359 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 'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'.
    2 pic x(32) value 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'.
    2 pic x(32) value 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'.
```

```

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。

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 ASCII マルチバイト・コード・ページで、文字あたり 1 バイト、2 バイト、または 4 バイトを使用します。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。以下のことが可能です。

- 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 を処理する必要がある場合、まず入力データを国別データ項目の UTF-16 に変換してください。国別データ項目を処理した後、出力用としてそれを中国語 GB 18030 に逆変換してください。この変換には、組み込

み関数 NATIONAL-OF および DISPLAY-OF を使用し、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

- 156 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

- 155 ページの『文字データの保管』

国別 (UTF-16) データの比較

国別 (UTF-16) データ、すなわち USAGE NATIONAL を持つデータ項目 (クラス国別かクラス数値かにかかわらず) および国別リテラルを、比較条件の他の種類のデータと明示的または暗黙的に比較することができます。

以下のステートメントで、国別データを使用する条件式をコード化できます。

- EVALUATE
- IF
- INSPECT
- PERFORM
- SEARCH
- STRING
- UNSTRING

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

- 166 ページの『2 つのクラス国別オペランドの比較』
- 166 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』
- 167 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』
- 167 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』
- 167 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。

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際に、対応する文字のすべてのペアが等しい場合、それらは等しいと判別されます。そうでない場合は、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

151 ページの『国別グループ』

関連タスク

152 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。

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。

USAGE とは無関係に、数値オペランドの代数値を比較できます。そのため、国別 10 進数項目または国別浮動小数点項目を、バイナリー項目、内部 10 進数項目、ゾーン 10 進数項目、表示浮動小数点項目、または他の任意の数値項目と比較できます。

関連タスク

151 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、DBCS、または数字編集) のいずれかの文字値と比較できます。

これらのオペランドは、基本国別データ項目へ移動されたかのように扱われます。文字は国別 (UTF-16) 表現へ変換され、2 つの国別文字オペランドの比較が進行します。

関連タスク

150 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。

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。

DBCS サポート用のコーディング

IBM Enterprise COBOL for 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。

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

『DBCS データの定義』

『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』

817 ページの『付録 B. 2 バイト文字セット (DBCS) データの変換』

関連参照

367 ページの『DBCS』

DBCS データの定義

DBCS データ項目を定義するには、PICTURE および USAGE 節を使用してください。DBCS データ項目では、PICTURE 記号の G、G と B、または N を使用できます。

DBCS データ項目は、USAGE DISPLAY-1 節を使用して指定できます。PICTURE 記号の G を使用する場合、USAGE DISPLAY-1 を指定する必要があります。PICTURE 記号の N を指定したが、USAGE 節を省略した場合、NSYMBOL コンパイラー・オプションの設定に応じて USAGE DISPLAY-1 または USAGE NATIONAL が暗黙指定されます。

DBCS 項目の定義で 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。

関連参照

393 ページの『NSYMBOL』

DBCS リテラルの使用

DBCS リテラルを表すには、接頭部 N または G を使用できます。

すなわち、次のいずれかの方法で DBCS リテラルを指定できます。

- N'DBCS 文字' (コンパイラー・オプション NSYMBOL(DBCS) が有効である場合)

- G'dbcs characters'

APOST または QUOTE コンパイラー・オプションの設定にかかわらず、引用符 (") またはアポストロフィ (') を DBCS リテラルの区切り文字として使用できます。DBCS リテラルに対して、同じ開始区切り文字と終了区切り文字をコーディングする必要があります。

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

29 ページの『形象定数の使用』

関連参照

352 ページの『APOST/QUOTE』

367 ページの『DBCS』

393 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。

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

108 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。

すなわち、以下のステップを実行してください。

1. MOVE ステートメントまたは NATIONAL-OF 組み込み関数を使用して、項目を国別データ項目の UTF-16 に変換します。
2. 必要に応じて国別データ項目を処理します。
3. DISPLAY-OF 組み込み関数を使用して、結果を英数字データ項目に逆変換します。

関連タスク

117 ページの『データ項目の結合 (STRING)』

120 ページの『データ項目の分割 (UNSTRING)』

124 ページの『データ項目のサブストリングの参照』

156 ページの『国別 (Unicode) 表現と間の変換』

第 8 章 ファイルの処理

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

『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』

176 ページの『ファイルの割り振り』

177 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 Enterprise COBOL for 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・ファイル进行处理するときは、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 UNIX)
- 索引付き (VSAM)
- 相対 (VSAM)

関連タスク

- 176 ページの『ファイルの割り振り』
- 179 ページの『第 9 章 QSAM ファイルの処理』
- 209 ページの『第 10 章 VSAM ファイルの処理』
- 241 ページの『第 11 章 行順次ファイルの処理』
- 『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。

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
『入出力コーディングの形式』
242 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 文字の長さにする事ができ、リテラルである場合には 1 から 160 文字の長さにする事ができます。*assignment-name* の *name* 部分は、DD ステートメントで、ALLOCATE コマンド (TSO) で、あるいは環境変数として (例えば、export コマンドで) (z/OS UNIX) コーディングします。

(3) *org*

編成は、SEQUENTIAL、LINE SEQUENTIAL、INDEXED、または RELATIVE にすることができます。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。ファイル状況キーは、2 文字のカテゴリ英数字またはカテゴリ国別項目として、あるいは 2 桁のゾーン 10 進数 (USAGE DISPLAY) または国別 10 進数 (USAGE NATIONAL) 項目として指定できます。

(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

179 ページの『第 9 章 QSAM ファイルの処理』

ファイルの割り振り

z/OS または z/OS UNIX 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。

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。

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。

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

- 542 ページの『環境変数の設定およびアクセス』
- 197 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 ファイルの動的作成』
- 233 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。

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チャーを呼び出してください。VSAM ファイルの場合、FILE STATUS 節の 2 番目のデータ項目を使用して、追加の VSAM 状況コード情報を取得できます。

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

- 273 ページの『入出力操作でのエラーの処理』
- 276 ページの『ERROR 宣言のコーディング』
- 277 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。

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
- PROCEDURE DIVISION 内でこれらのファイル内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。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、テープ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

- 『COBOL での QSAM ファイルおよびレコードの定義』
- 191 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』
- 196 ページの『QSAM ファイルのエラーの処理』
- 196 ページの『QSAM ファイルの操作』
- 205 ページの『QSAM を使用する z/OS UNIX ファイルへのアクセス』
- 206 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。

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。ORGANIZATION 節と ACCESS MODE 節は、ともに任意指定です。

関連タスク

『レコード形式の指定』

188 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。

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。レコードがブロック化されるようにしたい場合には、FD 記入項目に BLOCK CONTAINS 節をコーディングしてください。

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よう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

181 ページの『論理レコード』

関連タスク

181 ページの『固定長フォーマットの要求』

182 ページの『可変長フォーマットの要求』

185 ページの『スパン形式の要求』

187 ページの『不定形式の要求』

179 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

14 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

182 ページの『形式 F レコードのレイアウト』

183 ページの『形式 V レコードのレイアウト』

186 ページの『形式 S レコードのレイアウト』

188 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。

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 F と見なします。

- 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

181 ページの『論理レコード』

関連タスク

『可変長フォーマットの要求』

185 ページの『スパン形式の要求』

187 ページの『不定形式の要求』

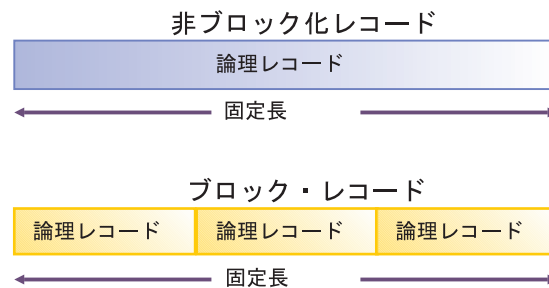
180 ページの『レコード形式の指定』

関連参照

『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

181 ページの『論理レコード』

関連タスク

181 ページの『固定長フォーマットの要求』

z/OS DFSMS: Using Data Sets (固定長レコード形式)

関連参照

183 ページの『形式 V レコードのレイアウト』

186 ページの『形式 S レコードのレイアウト』

188 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

可変長レコードのフォーマットは V または D にすることができます。フォーマット D のレコードは ASCII テープ・ファイル上では可変長のレコードです。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。

いずれの場合も、RECORDING MODE V を使用してください。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 *integer-1* および *integer-2* (RECORD IS VARYING FROM *integer-1* TO *integer-2*) に与えた場合、最大レコード長は、ファイルに関連したレベル 01 レコード記述

項目にコーディングされた長さにかかわらず、 *integer-2* にコーディングされた値です。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。

- RECORD CONTAINS *integer-1* TO *integer-2* 節 (形式 2 RECORD 節) を使用する。
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、その後にレベル 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

- 181 ページの『固定長フォーマットの要求』
- 185 ページの『スパン形式の要求』
- 187 ページの『不定形式の要求』
- 180 ページの『レコード形式の指定』

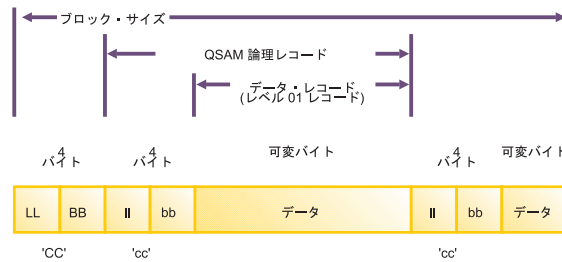
関連参照

- 14 ページの『FILE SECTION 記入項目』
- 『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。

II 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、ファイルの作成時に制御バイトを設定します。この制御バイト・フィールドは、プログラムの DATA DIVISION における論理レコードの記述には現れません。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

181 ページの『論理レコード』

関連タスク

182 ページの『可変長フォーマットの要求』

関連参照

182 ページの『形式 F レコードのレイアウト』

186 ページの『形式 S レコードのレイアウト』

188 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。

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。最大レコード長 (バイト数) に 4 を加えた値が、BLOCK CONTAINS 節で設定されたブロック・サイズより大きい場合、コンパイラは記録モードが S であると判定します。

プログラム内の形式 S のファイルの場合、コンパイラは最大レコード長を形式 V について使用されるのと同じ規則で判別します。この長さは、RECORD 節の使用法に基づいています。

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

181 ページの『論理レコード』

186 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

181 ページの『固定長フォーマットの要求』

- 182 ページの『可変長フォーマットの要求』
- 187 ページの『不定形式の要求』
- 180 ページの『レコード形式の指定』

関連参照

- 14 ページの『FILE SECTION 記入項目』
- 『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

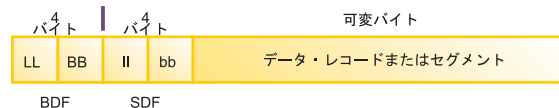
- 181 ページの『論理レコード』

関連タスク

- 185 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

185 ページの『スパン形式の要求』

関連参照

182 ページの『形式 F レコードのレイアウト』

183 ページの『形式 V レコードのレイアウト』

188 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。

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。コンパイラが記録モードを U と判別するのは、RECORDING MODE U をコーディングした場合だけです。

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。

- RECORD CONTAINS *integer* 節 (形式 1 RECORD 節) を使用する場合、ファイルに関連したレベル 01 レコード記述項目の長さにかかわらず、レコード長は *integer* 値になります。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。

integer-1 および *integer-2* (RECORD IS VARYING FROM *integer-1* TO *integer-2*) をコーディングする場合、ファイルに関連したレベル 01 レコード記述項目の長さにかかわらず、最大レコード長は *integer-2* 値です。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。

- RECORD CONTAINS *integer-1* TO *integer-2* 節 (形式 2 RECORD 節) を使用する場合に、*integer-1* および *integer-2* が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) と一致している場合、最大レコード長は *integer-2* 値です。

- 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。

形式 U ファイルおよび **READ INTO:** 形式 U ファイルに READ INTO ステートメントを指定した場合、コンパイラーは、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。

関連タスク

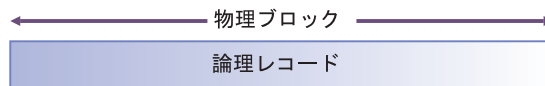
- 181 ページの『固定長フォーマットの要求』
- 182 ページの『可変長フォーマットの要求』
- 185 ページの『スパン形式の要求』
- 180 ページの『レコード形式の指定』

関連参照

- 14 ページの『FILE SECTION 記入項目』
- 『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

- 181 ページの『論理レコード』

関連タスク

- 187 ページの『不定形式の要求』

関連参照

- 182 ページの『形式 F レコードのレイアウト』
- 183 ページの『形式 V レコードのレイアウト』
- 186 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、コンパイラーはレコードがブロック化されないものと見なします。

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。

BLOCK CONTAINS 節でブロック・サイズを明示的に設定する場合には、装置についての最大ブロック・サイズよりも大きくしてはなりません。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。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングするか、BLOCK0 オプションを使用してコンパイルする。BLOCK0 について詳しくは、356 ページの『BLOCK0』を参照してください。
- ソース・プログラムに RECORD CONTAINS 0 をコーディングしない。
- JCL DD ステートメントで、BLKSIZE 値をコーディングしない。

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。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングするか、BLOCK0 オプションを使用してコンパイルする。
- 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。

BLOCK CONTAINS 節に直接、または BLOCK CONTAINS *n* RECORDS を使用して間接的に 32760 より大きなブロック・サイズを指定した場合には、データ・セットをテープに定義していない限り、データ・セットの OPEN は、ファイル状況コード 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。

- ソース・プログラムに BLOCK CONTAINS 0 をコーディングするか、BLOCK0 オプションを使用してコンパイルする。
- 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 節でブロック・サイズが明示的に定義されるファイルでも、使用されます。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。

- ファイルに対する DD ステートメントで、32760 より大きい BLKSZLIM 値を指定し、COBOL ソースで BLOCK CONTAINS 0 を使用する。
- 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・サイズおよび **DCB RECFM** サブパラメーター: z/OS のもとでは、DCB RECFM サブパラメーターに S または T オプションをコーディングすることができます。

- 標準ブロック (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または埋められていないトラックがないもの) だけを持つ形式 F レコードの場合は、DCB RECFM サブパラメーターで S (標準) オプションを使用してください。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

179 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

14 ページの『FILE SECTION 記入項目』
356 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

QSAM ファイル、または z/OS UNIX ファイル・システム 内のバイト・ストリーム・ファイルを QSAM (OPEN、READ、WRITE、REWRITE、および CLOSE) を使用して処理するには、次の入出力ステートメントをコーディングします。

- 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 UNIX 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。z/OS UNIX ファイル・システム 内のファイルは REWRITE を使用して更新することはできません。

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

『QSAM ファイルのオープン』

193 ページの『QSAM ファイルの動的作成』

194 ページの『QSAM ファイルへのレコードの追加』

194 ページの『QSAM ファイルの更新』

194 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』

195 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。

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、FD 項目に RECORD CONTAINS 0 をコーディングしても、レコード・サイズ属性は矛盾しません。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

『QSAM ファイルの動的作成』

201 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。

QSAM ファイルが z/OS 上で使用可能 であると見なされるのは、有効な DD ステートメント、環境変数用の export コマンド、または TSO ALLOCATE コマンドを使用してオペレーティング・システムに認識されている場合です。 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す。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す。

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。 INPUT、I-O、または EXTEND モードで開かれたファイルは、FILE-CONTROL 段落で SELECT OPTIONAL 節をコーディングすることによって、オプション・ファイルとして定義されます。

- 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。

この暗黙的な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同しないでください。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。

z/OS のもとでは、CBLQDA オプションを使用して作成されるファイルは一時データ・セットであり、プログラムの実行後は存在しません。

関連タスク

192 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。

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。

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。

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。)

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。

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。

LINAGE 節を WRITE BEFORE|AFTER ADVANCING *nn* LINES と組み合わせて使用する場合は、注意して値を設定してください。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 命令句が実行されるのは、書き込み操作が正常に完了した場合だけです。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。

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。
- 言語環境プログラム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せん。このためには、言語環境プログラムの 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。

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 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、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。

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 処理エラーが起これば、メッセージが出され、言語環境プログラム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。

FILE STATUS 節または EXCEPTION/ERROR 宣言を使用する場合は、そのファイルに対する DD ステートメントの DCB で EROPT=ACC をコーディングしてください。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。

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、予期したものではない結果になることがあります。

関連タスク

273 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。

関連タスク

197 ページの『QSAM ファイルの定義および割り振り』

- 200 ページの『QSAM ファイルの検索』
- 201 ページの『ファイル属性をプログラムと一致させる』
- 203 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

- 204 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

DD ステートメントまたは環境変数を使用して、QSAM ファイルまたは z/OS UNIX ファイル・システム のバイト・ストリーム・ファイルを定義できます。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。

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)

COBOL for OS/390 & VM バージョン 2 リリース 2 より前の COBOL のリリースとの互換性を与えるために、z/OS UNIX ファイルの DD 割り振りを使用するときに FILEDATA=TEXT を指定することもできますが、この使用はお勧めできません。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、LINE

SEQUENTIAL 編成を使用してください。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。

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	n/a
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	n/a

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。

- コンパイル時に BLOCK CONTAINS 0 RECORDS または BLOCK0 のいずれかのオプションが指定された場合のブロック・サイズ (BLKSIZE=) (推奨方法)。
- 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。

544 ページの『例: 環境変数の設定とアクセス』

関連タスク

- 188 ページの『ブロック・サイズの設定』
- 179 ページの『COBOL での QSAM ファイルおよびレコードの定義』
- 176 ページの『ファイルの割り振り』

関連参照

- 356 ページの『BLOCK0』
- 『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.proclstep.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

- 197 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。

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

『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 dataset-name(member-name) *.ddname *.stepname.ddname &&name &&name(member-name)
DSN=	

UNIT= (name[,unitcount])

VOLUME= (subparameter-list)

VOL=

LABEL= (subparameter-list)

DISP= (

OLD	[,DELETE ,KEEP ,PASS ,CATLG ,UNCATLG]	[,DELETE ,KEEP ,CATLG ,UNCATLG]
SHR		
MOD		

)

DCB= (subparameter-list)

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。

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、z/OS UNIX ファイル・システム では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。

DD または TSO ALLOCATE コマンドでファイルを使用可能にしていなかった場合でも、COBOL プログラムでファイルが作成されるように指定している場合、Enterprise COBOL はファイルを動的に割り振ります。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

『既存ファイルの処理』

202 ページの『新規ファイルの処理』

192 ページの『QSAM ファイルのオープン』

関連参照

14 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进行处理するときには、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。

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。レコードのデータ項目の基礎となる USAGE にかかわ

らず、整数サイズをバイト数で表してください。例えば、データ・セットの長さ属性を 104 バイト (LRECL=104) にすると想定しましょう。最大レコード長がレベル 01 レコード記述ではなく RECORD IS VARYING 節から判別されることを念頭に置いて、プログラム内で以下のコードを使用して形式 V 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 V  
    RECORD IS VARYING FROM 4 TO 100 CHARACTERS.  
01  COMMUTER-RECORD-A    PIC X(4).  
01  COMMUTER-RECORD-B    PIC X(75).
```

前の例では、既存のファイルがフォーマット V ではなくフォーマット U であると想定しています。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).
```

プログラムで固定長レコードを定義するには、RECORD CONTAINS *integer* 節をコーディングするか、またはこの節は省略し、すべてのレベル 01 レコード記述が同じ固定サイズになるようにコーディングしてください。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。同じプログラムを使用して実行時に種々のファイルを処理しようとする場合に、それらのファイルの固定長が異なっている場合、RECORD CONTAINS 0 をコーディングすることによりレコード長の矛盾を回避してください。

既存のファイルが ASCII データ・セット (DCB=(OPTCD=Q)) である場合には、ファイルについての FD 記入項目で CODE-SET 節を使用しなければなりません。

関連タスク

『新規ファイルの処理』

181 ページの『固定長フォーマットの要求』

182 ページの『可変長フォーマットの要求』

187 ページの『不定形式の要求』

192 ページの『QSAM ファイルのオープン』

関連参照

14 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。

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

- 201 ページの『既存ファイルの処理』
- 181 ページの『固定長フォーマットの要求』
- 182 ページの『可変長フォーマットの要求』
- 187 ページの『不定形式の要求』
- 192 ページの『QSAM ファイルのオープン』
- 193 ページの『QSAM ファイルの動的作成』

関連参照

- 14 ページの『FILE SECTION 記入項目』

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。

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。
- DATA(31) および RENT コンパイラ・オプションを指定してプログラムをコンパイルし、プログラム・オブジェクトを AMODE(31) にします。
- 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

43 ページの『ストレージとそのアドレス可能性』

関連タスク

203 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 UNIX ファイルへのアクセス

z/OS UNIX ファイル・システム 内のバイト・ストリーム・ファイルを、QSAM を使用して ORGANIZATION SEQUENTIAL ファイルとして処理できます。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。

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 および REWRITE はサポートされません。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。

COBOL for OS/390 & VM バージョン 2 リリース 2 より前の IBM COBOL のリリースとの互換性を与えるために、z/OS UNIX ファイルの DD 割り振りを使用するときに FILEDATA=TEXT を指定することもできますが、この使用はお勧めできません。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、LINE SEQUENTIAL 編成を使用してください。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

176 ページの『ファイルの割り振り』

197 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)

テープ上の **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には、ASCII アルファベットを要求し、レコード・フォーマットを定義し、(JCL で) DD 名を定義する必要があります。

さらに、プログラムが ASCII ファイルからの符号付き数値データ項目を処理する場合、別々の符号を持つゾーン 10 進数として数値データを定義してください。すなわち、USAGE DISPLAY として、また SIGN 節の SEPARATE 句で定義してください。

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 を明示的に指定することはできません。代わりに RECORDING MODE V を指定してください。この形式情報は、内部で 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。

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があります)。
- 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

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。

複雑な要件があるか、または VSAM を頻繁に使用する予定である場合には、使用するオペレーティング・システムの VSAM 資料を参照してください。

関連概念

210 ページの『VSAM ファイル』

関連タスク

- 212 ページの『VSAM ファイル編成およびレコードの定義』
- 218 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』
- 227 ページの『VSAM ファイルでのエラー処理』
- 228 ページの『パスワードによる VSAM ファイルの保護』
- 229 ページの『z/OS および z/OS UNIX のもとの VSAM データ・セットの操作』
- 237 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

- 236 ページの『VSAM ファイル用のレコード域の割り振り』
- 239 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。

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 (ESDS)	キー順データ・セット (KSDS)	相対レコード・データ・セット (RRDS)
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
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 うに処理される)
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ベ 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・セット内の空のスロットが 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ベ 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す。	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

『VSAM ファイルの順次編成の指定方法』

213 ページの『VSAM ファイルの索引編成の指定方法』

214 ページの『VSAM ファイルの相対編成の指定方法』

230 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。

DATA DIVISION の 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。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

『VSAM ファイルの順次編成の指定方法』

213 ページの『VSAM ファイルの索引編成の指定方法』

214 ページの『VSAM ファイルの相対編成の指定方法』

215 ページの『VSAM ファイルのアクセス・モードの指定』

216 ページの『VSAM ファイルのレコード長の定義』

277 ページの『ファイル状況キーの使用』

279 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』

230 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 プログラム内では、VSAM ESDS ファイルは、ORGANIZATION IS SEQUENTIAL 節で識別してください。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。

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) することは可能です。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

210 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。

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。基本キー・データ項目は、クラス英字、英数字、DBCS、数値、または国別にすることができます。USAGE NATIONAL を持っている場合、基本キーはカテゴリー国別にすることができます。あるいは国別編集、数字編集、国別 10 進数、または国別浮動小数点の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ーに関係なく、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

210 ページの『VSAM ファイル』

関連タスク

『代替キーの使用』

214 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ー (*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。

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 節を使用します。

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、基本キー・データ項目のように、クラス英字、英数字、DBCS、数値、または国別にすることができます。代替キーの照合は、キーのクラスやカテゴリーに関係なく、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。

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

231 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 プログラム内では、VSAM RRDS ファイルは、ORGANIZATION IS RELATIVE 節を使用して識別してください。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。

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

210 ページの『VSAM ファイル』

215 ページの『固定長および可変長 RRDS』

関連タスク

『可変長 RRDS の使用』

230 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。スロットの相対レコード番号に従って、レコードを保管および検索します。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 をロードする場合は、スロットをスキップするオプション、およびそれらを空にしておくオプションがあります。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

可変長レコードを含んだ相対レコード・データ・セット (RRDS) を使用する場合には、できる限り、VSAM 可変長 RRDS を使用する必要があります。・サポートを使用する場合、VSAM KSDS を使用して、可変長 RRDS をシミュレートします。

以下の手順を実行します。

1. ORGANIZATION IS RELATIVE 節でファイルを定義します。
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・プログラムを介して、VSAM ファイルを RRDS として定義します。

関連タスク

230 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。

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・アクセスの場合は、FILE-CONTROL 記入項目で ACCESS IS RANDOM 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、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。

『例: VSAM ファイルでの動的アクセスの使用』

関連タスク

223 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

223 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。

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

『固定長レコードの定義』

217 ページの『可変長レコードの定義』

関連参照

14 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。

表 25. VSAM 固定長レコードの定義

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。
RECORD 節を省略するが、ファイルに関連付けられたすべてのレベル 01 レコードを同じサイズとしてコーディングし、OCCURS DEPENDING ON 節には何もコーディングしません。		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。

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
RECORD IS VARYING FROM <i>integer-6</i> TO <i>integer-7</i> をコーディングする。	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。
RECORD IS VARYING をコーディングする。	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。
RECORD CONTAINS <i>integer-4</i> TO <i>integer-5</i> をコーディングする。	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。
RECORD 節を省略するが、サイズが異なるか OCCURS DEPENDING ON 節を含む、ファイルに関連付けられた複数のレベル 01 レコードをコーディングする。		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。

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

220 ページの『ファイル位置標識』

関連タスク

220 ページの『ファイルのオープン (ESDS、KSDS、または RRDS)』

223 ページの『VSAM ファイルからのレコードの読み取り』

224 ページの『VSAM ファイル内のレコードの更新』

225 ページの『VSAM ファイルへのレコードの追加』

226 ページの『VSAM ファイル内のレコードの置換』

226 ページの『VSAM ファイルからのレコードの削除』

227 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

223 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。

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、434 ページの『VSAMOPENFS』を参照してください。

関連タスク

『空のファイルのオープン』

223 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

222 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』

434 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。

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、以下のようにしてください。

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。

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。OPEN I-O または OPEN EXTEND を使用すると、プログラムのパフォーマンスに悪い影響があります。

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。

ACCESS IS SEQUENTIAL を使用する場合は、レコードを RECORD KEY の昇順に書き込まなければなりません。

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 がランダム処理または動的処理をサポートしていないので、COBOL が、OPEN OUTPUT または OPEN I-O とともに ACCESS IS RANDOM または ACCESS IS DYNAMIC を使用できるようにするために余分な処理を実行しなければなりません。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 VSAM データ・セットをロードする場合、z/OS DFSMS システム管理バッファリングによりローカル共用リソース (LSR) へのバッファリングが設定されていると、OPEN に対してファイル状況 30 が発生します。この場合に VSAM データ・セットを正常にロードするには、VSAM データ・セットがシステム管理バッファリングを迂回するよう DD AMP パラメーターで ACCBIAS=USER を指定してください。

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

223 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

『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に使用されるステートメント

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

- 221 ページの『空のファイルのオープン』
- 224 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

既にレコードが入っているファイルをオープンするには、OPEN INPUT、OPEN I-O、または OPEN EXTEND を使用してください。

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・キーを持っていなければなりません。

関連タスク

- 221 ページの『空のファイルのオープン』
- 229 ページの『z/OS および z/OS UNIX のもとでの VSAM データ・セットの操作』

関連参照

- 222 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。

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 索引付きファイルの場合は、ファイル位置標識のファイル位置標識から、使用中のキー (RECORD KEY または ALTERNATE RECORD KEY) の昇順に従い、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

ランダム

READ 要求の前に、RECORD KEY または ALTERNATE RECORD KEY または RELATIVE KEY をどのように設定するかによって、任意の順序で

動的 順次とランダムの混合で

動的アクセスの場合、順次検索には READ NEXT を使用し、ランダム検索 (キーによる) には READ を使用することによって、特定のレコードの直接読み取りと、いくつかのレコードの順次読み取りを切り替えることができます。

特定のレコードから順次に読み取りを行いたい場合には、READ NEXT 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。同じ代替キーを持つデータ・セット・レコードのそれぞれを検索するためには、一連の READ NEXT ステートメントが必要です。同じ代替キー値を持つ、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

220 ページの『ファイル位置標識』

関連タスク

215 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。

表 30. VSAM ファイル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
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

222 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。

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。SELECT 節に RELATIVE KEY データ項目を組み込むと、書き込まれるレコードの相対レコード番号がそのデータ項目に入れます。

ランダムまたは動的なレコード追加: レコードを索引付きデータ・セットに書き込むとき、ACCESS IS RANDOM または ACCESS IS DYNAMIC であれば、レコードは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。

順次ファイルの場合、置換レコードの長さは元のレコードの長さと同じでなければなりません。索引ファイルまたは可変長相対ファイルの場合、置換するレコードの長さを変更することができます。

レコードをランダムまたは動的に置換するには、まずレコードを READ する必要はありません。そうではなく、次のように置換対象のレコードを検索します。

- 索引付きファイルの場合、レコード・キーを RECORD KEY データ項目に移動してから、REWRITE を発行します。
- 相対ファイルの場合、相対レコード番号を RELATIVE KEY データ項目に移動してから、REWRITE を発行します。

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。

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 データ項目に移動してから、DELETE を出します。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。

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。
- 言語環境プログラム条件処理が完了し、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。

VSAM ファイルを扱う際の OPEN および CLOSE エラーはすべて、それがプログラム内の論理エラーであっても外部ストレージ・メディア上の入出力エラーであっても、また、たとえ DECLARATIVE も FILE STATUS 節もコーディングされていなくても、制御を COBOL プログラムに戻します。

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。

ファイル状況キーや宣言をコーディングしなかった場合、重大な VSAM 処理エラーが起こると、メッセージが出され、言語環境プログラム条件が信号で伝えられます。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。

関連タスク

273 ページの『入出力操作でのエラーの処理』

279 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。

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atalog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。

- Catalog記入項目に読み取りパスワードが入っている場合には、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、ファイルは COBOL プログラム内でオープンおよびアクセスできません。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必要があります。
- Catalog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すが、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。
- Catalog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 UNIX 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 UNIX のもとで使用する場合は、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。

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』
『VSAM ファイルの定義』
- 231 ページの『代替索引の作成』
- 233 ページの『VSAM ファイルの割り振り』
- 234 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。

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)
```

この方法でデータ・セットを定義すると、すべてのレコードが固定スロット・サイズ *n* まで埋め込まれます。RECORD IS VARYING ON *data-name* 形式の RECORD

節を使用すると、WRITE または REWRITE では、VSAM によって転送されるレコードの長さとして `DEPENDING ON data-name` で指定された長さが使用されます。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しなければなりません。

関連タスク

『代替索引の作成』

233 ページの『VSAM ファイルの割り振り』

214 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。

代替索引の使用を計画している場合には、以下のことを覚え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。

1. `DEFINE ALTERNATEINDEX` コマンドを使用して、代替索引を定義します。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、代替索引は、FILE-CONTROL 段落の ALTERNATE RECORD KEY 節によってのみ識別されます。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. DEFINE PATH コマンドを使用して、代替索引を基本クラスター (代替索引を介してアクセスするデータ・セット) に関連付けます。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。

『例: 代替索引の項目』

関連タスク

214 ページの『代替索引の使用』

関連参照

言語環境プログラム 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:

```
//MASTERA DD DSN=clustername,DISP=OLD (1)
//MASTERA1 DD DSN=path1,DISP=OLD (2)
//MASTERA2 DD DSN=path2,DISP=OLD (3)
```

環境変数を使用する場合:

```
export MASTERA=DSN(clustername),OLD (1)
export MASTERA=DSN(path1),OLD (2)
export MASTERA=DSN(path2),OLD (3)
. . .
```

```

FILE-CONTROL.
  SELECT MASTER-FILE ASSIGN TO MASTERA                (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

231 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。

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 ファイルを割り振る場合、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"
```

いずれの場合も、*dsname* は、アクセス方式サービス・プログラムの DEFINE CLUSTER コマンドまたは DEFINE PATH コマンドで使用した名前と同じ名前ではありません。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。次はその例です。

```
//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。次はその例です。

```
//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た最初の代替レコード (FILE-CONTROL 段落の ALTERNATE RECORD KEY 節) と関連付けられたパスに 1 を使用します。
- 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

176 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。

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、読み取り保水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

『RLS モードでの VSAM ファイルに関する更新問題の回避』

236 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

236 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。

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 を使用するためには、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。

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

279 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・レコード内のデータを CALL...USING パラメーターとして AMODE 24 サブプログラムに渡すプログラムは、影響を受けます。このようなプログラムは、DATA(24) コンパイラー・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。

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	
バッファリング	<p>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。ACCESS IS DYNAMIC の場合は、BUFND と BUFNI の両方を指定します。</p> <p>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。</p>	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	<p>以下の場合は、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。</p> <ul style="list-style-type: none"> 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。 <p>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。</p>	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。

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
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、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。	VSAM 固定長相対データ・セットは、VSAM 可変長相対データ・セットと比較して、スペース的には効率がよくありませんが、実行時の効率はよくなります。
制御インターバル・サイズ (CISZ)	<p>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。</p> <p>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。</p>	<p>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。</p> <p>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、大きな OS BLKSIZE と同様に、順次処理の場合に効率がよりよくなります。</p> <p>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。</p>

関連タスク

215 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・プールの作成、フリー・
スペースの最適なパーセントの選択)

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) 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 V1.3 の KSDS 用に導入されたものです。DFSMS/MVS V1.4 以降、拡張アドレッシング機能はレコード・レベル共有 (RLS) でサポートされています。DFSMS/MVS V1.5 では、拡張アドレッシング機能はその他すべての VSAM レコード編成へ拡張されました。

関連タスク

z/OS DFSMS: Using Data Sets

第 11 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)。

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られます。

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。

- ENVIRONMENT DIVISION および DATA DIVISION 内でファイルを識別および記述する
- 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

『COBOL での行順次ファイルおよびレコードの定義』

243 ページの『行順次ファイルの割り振り』

243 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』

246 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

COBOL プログラムのファイルを行順次ファイルとして定義し、そのファイルを対応する外部ファイル名 (DD 名または環境変数名) に関連付けるには、ENVIRONMENT DIVISION の FILE-CONTROL 段落を使用します。

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。ACCESS 句と FILE STATUS 節はオプションです。

関連タスク

242 ページの『行順次ファイルの構造の記述』

243 ページの『行順次ファイルの割り振り』

243 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

『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。

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).
```

同一の固定サイズをコーディングし、どのレベル 01 レコード記述項目の OCCURS DEPENDING ON 節もファイルと関連付けられていない場合には、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

243 ページの『行順次ファイルの割り振り』

243 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

z/OS UNIX ファイル・システムでは、DD ステートメントまたは環境変数のいずれかを使用して行順次ファイルを割り振ることができます。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。

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 UNIX 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

176 ページの『ファイルの割り振り』

241 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。

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

- 241 ページの『COBOL での行順次ファイルおよびレコードの定義』
- 242 ページの『行順次ファイルの構造の記述』
- 『行順次ファイルのオープン』
- 『行順次ファイルからのレコードの読み取り』
- 245 ページの『行順次ファイルへのレコードの追加』
- 245 ページの『行順次ファイルのクローズ』
- 246 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。

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

- 『行順次ファイルからのレコードの読み取り』
- 245 ページの『行順次ファイルへのレコードの追加』
- 245 ページの『行順次ファイルのクローズ』
- 243 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。

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。

レコード域の残りは、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。)

関連タスク

244 ページの『行順次ファイルのオープン』

『行順次ファイルへのレコードの追加』

『行順次ファイルのクローズ』

243 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。

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 10 進数データ項目は、符号なしにするか、または符号付きの場合には SIGN 節の SEPARATE 句で宣言する必要があります。

関連タスク

244 ページの『行順次ファイルのオープン』

244 ページの『行順次ファイルからのレコードの読み取り』

『行順次ファイルのクローズ』

243 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。

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。
- 言語環境プログラム条件処理が完了し、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、言語環境プログラムの 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

- 244 ページの『行順次ファイルのオープン』
- 244 ページの『行順次ファイルからのレコードの読み取り』
- 245 ページの『行順次ファイルへのレコードの追加』
- 243 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。

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、言語環境プログラム条件が発生します。

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、予期したものではない結果になることがあります。

関連タスク

- 243 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』
- 273 ページの『入出力操作でのエラーの処理』

第 12 章 ファイルのソートおよびマージ

SORT または MERGE ステートメントを使用すると、レコードを特定のシーケンスで並べることができます。同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。

注: このトピックで説明されている SORT ステートメント、ソート・プロセス、およびソート制約事項は、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、98 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 境界より上または下のいずれにでも常駐させることができます。

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。

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。

制約事項:

- z/OS UNIX のもとでは、SORT 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したプログラムで SORT ステートメントまたは MERGE ステートメントを使用することはできません。これには、オブジェクト指向構文を使用するプログラムとマルチスレッド・アプリケーションも含まれます。これらはいずれも THREAD オプションを必要とします。

関連概念

『ソートおよびマージ・プロセス』

関連タスク

98 ページの『テーブルのソート』

249 ページの『ソートまたはマージ・ファイルの記述』

249 ページの『ソートまたはマージへの入力の記述』

252 ページの『ソートまたはマージからの出力の記述』

256 ページの『ソートまたはマージの要求』

260 ページの『ソートまたはマージの成否の判断』

260 ページの『ソートまたはマージ操作の途中停止』

261 ページの『FASTSORT によるソート・パフォーマンスの向上』

264 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

268 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。

ファイルをソートするには、形式 1 SORT ステートメントを使用します。

複数のファイルのマージ時には (これらのファイルはソート済みでなければなりません)、レコードは、各レコード内の 1 つ以上のキーの内容に従って結合され、順序付けされます。レコードは、各キーの昇順または降順に順序付けすることができます。ソートの場合と同様、レコードはまず最初の (基本) キーの内容に従って順序付けされ、次に 2 番目のキーの内容に従って順序付けされる、というようになります。

MERGE . . . USING を使用して、順序付けられた 1 つのファイルとして結合したい複数のファイルの名前を指定します。マージ操作では、入力ファイルのレコード内のキーを比較し、順序付けられたレコードを 1 つずつ、出力プロシージャの RETURN ステートメントに、または GIVING 句で指定されたファイルに渡します。

関連タスク

257 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。

次のようにコーディングします。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つ以上の SELECT 節をコーディングし、ソート・ファイルの名前を指定します。以下に例を示します。

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT Sort-Work-1 ASSIGN TO SortFile.
```

Sort-Work-1 は、プログラムで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. DATA DIVISION の FILE SECTION の SD 項目でソート・ファイルを記述します。それぞれの SD 記入項目がレコード記述を含んでいなければなりません。以下に例を示します。

```
DATA DIVISION.  
FILE SECTION.  
SD Sort-Work-1  
    RECORD CONTAINS 100 CHARACTERS.  
01 SORT-WORK-1-AREA.  
    05 SORT-KEY-1    PIC X(10).  
    05 SORT-KEY-2    PIC X(10).  
    05 FILLER        PIC X(80).
```

SD 記入項目で記述するファイルは、ソートまたはマージ操作に使用される作業ファイルです。このファイルに入力または出力操作を実行することはできません。また、このファイルに DD 名定義を提供する必要もありません。

関連参照

14 ページの『FILE SECTION 記入項目』

ソートまたはマージへの入力の記述

ソートまたはマージ用の入力ファイルは、以下の手順に従って記述してください。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に 1 つ以上の SELECT 節をコーディングし、入力ファイルの名前を指定します。以下に例を示します。

```
ENVIRONMENT DIVISION.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT Input-File ASSIGN TO InFile.
```

Input-File は、プログラム内のファイルの名前です。 この名前を使用してファイルを参照してください。

2. その入力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。 以下に例を示します。

```
DATA DIVISION.  
FILE SECTION.  
FD Input-File  
  LABEL RECORDS ARE STANDARD  
  BLOCK CONTAINS 0 CHARACTERS  
  RECORDING MODE IS F  
  RECORD CONTAINS 100 CHARACTERS.  
01 Input-Record   PIC X(100).
```

関連タスク

- 251 ページの『入力プロシージャのコーディング』
- 256 ページの『ソートまたはマージの要求』

関連参照

- 14 ページの『FILE SECTION 記入項目』

例: SORT 用のソート・ファイルおよび入力ファイルの記述

次の例は、ソート作業ファイルおよび入力ファイルを記述するのに必要な ENVIRONMENT DIVISION および DATA DIVISION の記入項目を示しています。

```
ID Division.  
Program-ID. Smp1Sort.  
Environment Division.  
Input-Output Section.  
File-Control.  
*  
* Assign name for a working file is treated as documentation.  
*  
  Select Sort-Work-1 Assign To SortFile.  
  Select Sort-Work-2 Assign To SortFile.  
  Select Input-File   Assign To InFile.  
  . . .  
Data Division.  
File Section.  
SD Sort-Work-1  
  Record Contains 100 Characters.  
01 Sort-Work-1-Area.  
  05 Sort-Key-1   Pic X(10).  
  05 Sort-Key-2   Pic X(10).  
  05 Filler       Pic X(80).  
SD Sort-Work-2  
  Record Contains 30 Characters.  
01 Sort-Work-2-Area.  
  05 Sort-Key     Pic X(5).  
  05 Filler       Pic X(25).  
FD Input-File  
  Label Records Are Standard  
  Block Contains 0 Characters  
  Recording Mode is F  
  Record Contains 100 Characters.  
01 Input-Record   Pic X(100).  
  . . .  
Working-Storage Section.  
01 EOS-Sw         Pic X.
```



```

01 Filler.
   05 Table-Entry Occurs 100 Times
       Indexed By X1    Pic X(30).
   . . .

```

関連タスク

256 ページの『ソートまたはマージの要求』

入力プロシージャーのコーディング

入力ファイルのレコードを、それらがソート・プログラムに解放される前に処理する場合には、形式 1 SORT ステートメントの INPUT PROCEDURE 句を使用してください。

入力プロシージャーを使用して、以下のことを行うことができます。

- データ項目を WORKING-STORAGE または LOCAL-STORAGE からソート・ファイルに解放する。
- プログラム内の別な場所で既に読み取られているレコードを解放する。
- 入力レコードからレコードを読み取り、それらを選択または処理し、それらをソート・ファイルに解放する

それぞれの入力プロシージャーは、段落またはセクションのいずれかで構成されなければなりません。例えば、WORKING-STORAGE または LOCAL-STORAGE の表からのレコードをソート・ファイル SORT-WORK-2 に解放するには、次のようにコーディングすることができます。

```

      SORT SORT-WORK-2
      ON ASCENDING KEY SORT-KEY
      INPUT PROCEDURE 600-SORT3-INPUT-PROC
      . . .
600-SORT3-INPUT-PROC SECTION.
      PERFORM WITH TEST AFTER
          VARYING X1 FROM 1 BY 1 UNTIL X1 = 100
          RELEASE SORT-WORK-2-AREA FROM TABLE-ENTRY (X1)
      END-PERFORM.

```

レコードをソート・プログラムに転送するためには、すべての入力プロシージャーに少なくとも 1 つの RELEASE または RELEASE FROM ステートメントが含まれていなければなりません。例えば、X から A を解放するには、次のようにコーディングできます。

```

MOVE X TO A.
RELEASE A.

```

あるいは、次のようにコーディングできます。

```

RELEASE A FROM X.

```

次の表では、RELEASE ステートメントと RELEASE FROM ステートメントを比較しています。

RELEASE	RELEASE FROM
<pre> MOVE EXT-RECORD TO SORT-EXT-RECORD PERFORM RELEASE-SORT-RECORD . . . RELEASE-SORT-RECORD. RELEASE SORT-RECORD </pre>	<pre> PERFORM RELEASE-SORT-RECORD . . . RELEASE-SORT-RECORD. RELEASE SORT-RECORD FROM SORT-EXT-RECORD </pre>

関連参照

254 ページの『入出力プロシージャーに関する制約事項』

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。

1. ENVIRONMENT DIVISION の FILE-CONTROL 段落に SELECT 節をコーディングし、出力ファイルの名前を指定します。 以下に例を示します。

```

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT Output-File ASSIGN TO OutFile.
  
```

Output-File は、プログラム内のファイルの名前です。 この名前を使用してファイルを参照してください。

2. その出力ファイル (マージの場合は複数のファイル) を、DATA DIVISION の FILE SECTION の FD 記入項目で記述します。 以下に例を示します。

```

DATA DIVISION.
FILE SECTION.
FD Output-File
  LABEL RECORDS ARE STANDARD
  BLOCK CONTAINS 0 CHARACTERS
  RECORDING MODE IS F
  RECORD CONTAINS 100 CHARACTERS.
01 Output-Record PIC X(100).
  
```

関連タスク

『出力プロシージャーのコーディング』

256 ページの『ソートまたはマージの要求』

関連参照

14 ページの『FILE SECTION 記入項目』

出力プロシージャーのコーディング

ソート済みのレコードをソート作業ファイルから別のファイルに書きこむ前に、それらの選択、編集、またはそれ以外の変更を行うには、形式 1 SORT ステートメントの OUTPUT PROCEDURE 句を使用してください。

それぞれの出力プロシージャーは、セクションまたは段落のいずれかで構成されなければなりません。また、出力プロシージャーには、以下の両方を含めなければなりません。

- 少なくとも 1 つ以上の RETURN ステートメントまたは INTO 句を含んだ 1 つの RETURN ステートメント
- レコードの処理に必要なステートメント。レコードは、RETURN ステートメントによって一度に 1 つずつ使用可能になります。

RETURN ステートメントによって、ソート済みの各レコードが出力プロシージャ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 のコーディング: DFSORT の使用時に、COBOL プログラムが実行を終了する前に RETURN ステートメントが AT END 条件を検出しなかった場合、形式 1 SORT ステートメントは異常終了して DFSORT メッセージ IEC025A が出されることがあります。この状態を回避するには、必ず RETURN ステートメントを AT END 句と一緒にコーディングするようにしてください。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。

『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

254 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、プログラムが実行を終了する前に RETURN ステートメントが AT END 条件に出合うようにしたコーディング手法を示しています。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。
- 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)。
- 入力または出力プロシージャの中から、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。
- 言語環境プログラムの条件処理では、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

251 ページの『入力プロシージャのコーディング』

252 ページの『出力プロシージャのコーディング』

言語環境プログラム 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。

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

264 ページの『ソート動作の制御』

267 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

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
- サイズが異なるレコードを定義する 2 つ以上のレコード記述 (あるいは、OCCURS DEPENDING ON 節を含むレコード)

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 (モーダル長) を指定します。

関連タスク

266 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』

264 ページの『ソート動作の制御』

ソートまたはマージの要求

事前処理を行わずに 1 つの入力ファイル (MERGE の場合は複数のファイル) からレコードを読み取るには、SORT . . . USING または MERGE . . . USING と、SELECT 節で宣言した入力ファイルの名前を使用します。

ソート済みまたはマージ済みレコードを、これ以上処理せずに、ソート・プログラムまたはマージ・プログラムから別のファイルへ転送するには、SORT . . . GIVING または MERGE . . . GIVING、および SELECT 節で宣言された出力ファイルの名前を使用してください。以下に例を示します。

```
SORT Sort-Work-1
  ON ASCENDING KEY Sort-Key-1
  USING Input-File
  GIVING Output-File.
```

SORT . . . USING または MERGE . . . USING の場合、コンパイラーは入力プロシージャを生成します。このプロシージャは、ファイルを開き、レコードを読み取り、レコードをソート/マージ・プログラムに解放し、ファイルを閉じます。

SORT または MERGE ステートメントが実行を開始するとき、ファイルが開いた状態にしないでください。SORT . . . GIVING または MERGE . . . GIVING の場合、コンパイラーは出力プロシージャを生成します。このプロシージャは、ファイルを開き、レコードを返し、レコードを書き込み、ファイルを閉じます。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。

250 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』

また、FASTSORT コンパイラー・オプションを使用して、IBM DFSORT または同等の製品で、Enterprise COBOL の代わりにソート入出力を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、377 ページの『FASTSORT』を参照してください。

ソート・レコードがソートされる前にソート・レコードに対して入力プロシージャが実行されるようにする場合は、SORT . . . INPUT PROCEDURE を使用します。ソート済みレコードに対して出力プロシージャが実行されるようにする場合は、SORT . . . OUTPUT PROCEDURE を使用します。以下に例を示します。

```
SORT Sort-Work-1
  ON ASCENDING KEY Sort-Key-1
  INPUT PROCEDURE EditInputRecords
  OUTPUT PROCEDURE FormatData.
```

258 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、マージ済みレコードに対して出力プロシージャが実行されるようにしたい場合は、MERGE . . . OUTPUT PROCEDURE を使用します。以下に例を示します。

```
MERGE Merge-Work
ON ASCENDING KEY Merge-Key
USING Input-File-1 Input-File-2 Input-File-3
OUTPUT PROCEDURE ProcessOutput.
```

FILE SECTION では、SD 記入項目の *Merge-Work*、および FD 記入項目の入力ファイルを定義する必要があります。

関連タスク

255 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。

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、98 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。

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。

SORT および *MERGE* キーは、クラス英字、英数字、国別、数値 (ただし、*USAGE* *NATIONAL* の数値ではない) にすることができます。 *USAGE* *NATIONAL* を持っている場合、キーはカテゴリー国別にするか、あるいは国別編集または数字編集データ項目にすることができます。キーを、国別 10 進数データ項目にしたり、国別浮動小数点データ項目にすることはできません。

国別キーの照合順序は、キーのバイナリーの順序によって決まります。キーとして国別データ項目を指定する場合は、*SORT* または *MERGE* ステートメントの *COLLATING SEQUENCE* 句はいずれもそのキーに適用されません。

同じ COBOL プログラムの中に SORT ステートメントと MERGE ステートメントを混在させることができます。プログラムはいくつのソート操作およびマージ操作でも実行することができます。ただし、1 つの操作が終了してからでなければ、次の操作を開始することはできません。

関連タスク

98 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.

```

. . .
    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

256 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。 OBJECT-COMPUTER 段落で PROGRAM COLLATING SEQUENCE 節をコーディングしない限り、デフォルトの照合シーケンスは、EBCDIC です。

デフォルト・シーケンスをオーバーライドするには、SORT または MERGE ステートメントの COLLATING SEQUENCE 句を使用してください。 プログラムの SORT または MERGE ステートメントごとに異なる照合シーケンスを使用することができます。

PROGRAM COLLATING SEQUENCE 節および COLLATING SEQUENCE 句は、クラス英字または英数字のキーにのみ適用されます。

ASCII ファイルをソートまたはマージする場合は、ASCII 照合シーケンスを要求する必要があります。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』

257 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

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。

ソートまたはマージの成否の判断

DFSORT プログラムは、各ソートまたはマージ操作の終了後に、0 (正常終了) または 16 (異常終了) のいずれかの完了コードを返します。完了コードは、SORT-RETURN 特殊レジスターに保管されます。

それぞれの SORT または MERGE ステートメントの後で、正常終了かどうかをテストしなければなりません。以下に例を示します。

```

SORT SORT-WORK-2
  ON ASCENDING KEY SORT-KEY
  INPUT PROCEDURE IS 600-SORT3-INPUT-PROC
  OUTPUT PROCEDURE IS 700-SORT3-OUTPUT-PROC.
IF SORT-RETURN NOT=0
  DISPLAY "SORT ENDED ABNORMALLY. SORT-RETURN = " SORT-RETURN.
. . .
600-SORT3-INPUT-PROC SECTION.
. . .
700-SORT3-OUTPUT-PROC SECTION.
. . .

```

プログラムの中で SORT-RETURN をまったく参照しないと、COBOL ランタイムで完了コードがテストされます。16 の場合、COBOL は、ランタイム診断メッセージを出します。

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

264 ページの『NOFASTSORT によるソート・エラーの検査』

264 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。

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。FASTSRT を使用した場合は、DFSORT プロダクトが (Enterprise COBOL の代わりに)、SORT . . . USING および SORT . . . GIVING ステートメントで指定した入出力ファイルに入出力操作を実行します。

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の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

377 ページの『FASTSRT』

『JCL に関する FASTSRT の要件』

『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「ソート入出力ファイル」は形式 1 SORT ステートメントにのみ関連します。

FASTSORT 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 IGZSRTECD 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となります。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 FASTSORT 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 FASTSORT 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 FASTSORT 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 FASTSORT に不適格である場合でも、出力について FASTSORT に適格である可能性があります。

FASTSORT 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 FASTSORT 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 FASTSORT 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ー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

NOFASTSRT によるソート・エラーの検査

NOFASTSRT 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ります。

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 32. NOFASTSRT 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ると、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

- 260 ページの『ソートまたはマージの成否の判断』
- 277 ページの『ファイル状況キーの使用』
- 276 ページの『ERROR 宣言のコーディング』
- 260 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。

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 33. ソート動作を制御する方法

設定またはテスト対象	特殊レジスターまたはコンパイラ・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 特殊レジスター	OPTION (キーワード RESINV)
使用される主記憶域の量	SORT-CORE-SIZE 特殊レジスター	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ーダル長	SORT-MODE-SIZE 特殊レジスター	SMS=nnnnn
ソート制御ステートメント・データ・セットの名称 (デフォルト IGZSRTCD)	SORT-CONTROL 特殊レジスター	なし
ソート・メッセージ・ファイルの名称 (デフォルト SYSOUT)	SORT-MESSAGE 特殊レジスター	OPTION (キーワード MSGDDN)
ソート・レコードの数	SORT-FILE-SIZE 特殊レジスター	OPTION (キーワード FILSZ)
ソート完了コード	SORT-RETURN 特殊レジスター	なし

ソート特殊レジスター: SORT-CONTROL は、ソート制御ステートメント・ファイルの DD 名が入れられる、8 文字の COBOL 特殊レジスターです。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 特殊レジスターは、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ーは、特殊レジスター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 特殊レジスターを使用すると、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ーごとに、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

- 260 ページの『ソートまたはマージの成否の判断』
- 260 ページの『ソートまたはマージ操作の途中停止』
- 266 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』
- 267 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

- 266 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。

(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。

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)。

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

264 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

『IGZSRTCD データ・セットのデフォルトの特性』

IGZSRTCD データ・セットのデフォルトの特性

IGZSRTCD 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 IGZSRTCD 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

264 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

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なりません。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
- 言語環境プログラムのランタイム・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

264 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

264 ページの『ソート動作の制御』

251 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。

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。

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 IGZSRITCD) 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。

関連概念

767 ページの『第 35 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

266 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』

767 ページの『チェックポイントの設定』

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。しかし、形式 1 SORT ステートメント (および CICS のもとで稼働する独自のソート・プログラム) を使用すれば、少量のデータをソートすることができます。CICS の下でも形式 2 SORT ステートメントを使用すればテーブルをソートできます。

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。

形式 2 SORT ステートメントは CICS の下でサポートされています。特別な SORT プログラムを作成する必要はありません。

関連タスク

251 ページの『入力プロシージャのコーディング』

252 ページの『出力プロシージャのコーディング』

501 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

『CICS SORT アプリケーションの制約事項』

511 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

CICS のもとで実行され、形式 1 SORT ステートメントを使用する COBOL アプリケーションには、いくつかの制約事項が適用されます。

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

511 ページの『CICS 予約語テーブル』

第 13 章 エラーの処理

起こり得るシステムまたは実行時の問題を予測するコードをプログラムに入れてください。そのようなコードを含めない場合、出力データまたはファイルが破損しても、ユーザーは問題が発生していることに気付くことさえない可能性があります。

エラー処理コードでは、状態の処理、メッセージの発行、プログラムの停止などのアクションを取ることができます。例えば、データ入力エラーまたはご使用のシステムで定義されたエラーに対して、独自のエラー検出ルーチンを作成することができます。どのようなイベントであっても、警告メッセージをコーディングするのはよいことです。

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
- CALL ステートメントの ON EXCEPTION または ON OVERFLOW
- エラー処理用のユーザー作成ルーチン

関連タスク

272 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』

273 ページの『算術演算でのエラーの処理』

273 ページの『入出力操作でのエラーの処理』

283 ページの『プログラム呼び出し時のエラーの処理』

283 ページの『エラー処理用のルーチンの作成』

ダンプの要求

言語環境プログラム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、言語環境プログラム 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 言語環境プログラム・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

- 423 ページの『TEST』
- 言語環境プログラム デバッグのガイド
- 言語環境プログラム プログラミング・リファレンス (CEE3DMP-- ダンプの生成)

ストリングの結合および分割におけるエラーの処理

ストリングの結合または分割中に、STRING または UNSTRING に使用されるポインターは、受信フィールドの範囲外になる可能性があります。 オーバーフロー条件が存在する可能性はありますが、COBOL ではオーバーフローの発生を許可しません。

その代わりに、STRING 操作や UNSTRING 操作は完了せず、受信フィールドは未変更のままとなり、制御は次の順次ステートメントに移動します。 STRING または UNSTRING ステートメントの ON OVERFLOW 句をコーディングしないと、操作未完了の通知が出されません。

次のステートメントを考えてください。

```
String Item-1 space Item-2 delimited by Item-3
      into Item-4
      with pointer String-ptr
      on overflow
        Display "A string overflow occurred"
End-String
```

以下に、ステートメントの実行前と実行後のデータ値を示します。

データ項目	PICTURE	実行前の値	実行後の値
Item-1	X(5)	AAAAA	AAAAA
Item-2	X(5)	EEEEAA	EEEEAA
Item-3	X(2)	EA	EA
Item-4	X(8)	bbbbbb ¹	bbbbbb ¹
String-ptr	9(2)	0	0
1. 記号 <i>b</i> は、ブランク・スペースを表します。			

String-ptr の値は (0) で、受信フィールドには達しないため、オーバーフロー条件が発生し、STRING 操作は完了しません (String-ptr が 9 より大きい場合にも、オーバーフローが起こります)。 ON OVERFLOW が指定されていなかった場合は、Item-4 の内容が未変更のままであったことについて通知されません。

算術演算でのエラーの処理

算術演算の結果が、それらを入れる固定小数点フィールドより大きかったり、0 除算が試みられたりすることがあります。いずれの場合も、ADD、SUBTRACT、MULTIPLY、DIVIDE、または COMPUTE ステートメントの後の ON SIZE ERROR 節でその状況を処理することができます。

ON SIZE ERROR が固定小数点オーバーフローおよび 10 進オーバーフローで正常に動作するようにするためには、TRAP(ON) ランタイム・オプションを指定する必要があります。

以下の場合、ON SIZE ERROR 節の命令ステートメントが実行され、結果フィールドは変更されません。

- 固定小数点オーバーフロー
- 0 による除算
- 0 の 0 乗
- 0 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、ゼロ除算が発生した場合にプログラムが通知メッセージを出すように ON SIZE ERROR 命令ステートメントをコーディングする方法を示しています。

```
DIVIDE-TOTAL-COST.  
  DIVIDE TOTAL-COST BY NUMBER-PURCHASED  
    GIVING ANSWER  
    ON SIZE ERROR  
      DISPLAY "ERROR IN DIVIDE-TOTAL-COST PARAGRAPH"  
      DISPLAY "SPENT " TOTAL-COST, " FOR " NUMBER-PURCHASED  
      PERFORM FINISH  
    END-DIVIDE  
  ...  
  FINISH.  
  STOP RUN.
```

ゼロ除算が発生すると、プログラムはメッセージを作成し、プログラム実行を停止します。

入出力操作でのエラーの処理

入力または出力操作が失敗しても、COBOL が自動的に訂正処置をとることはありません。重大エラーではない入出力エラー後にプログラムの実行を継続するかどうかを選択してください。

特定の入力または出力の状態またはエラーの代行受信および処理には、以下のいずれかの技法を使用することができます。

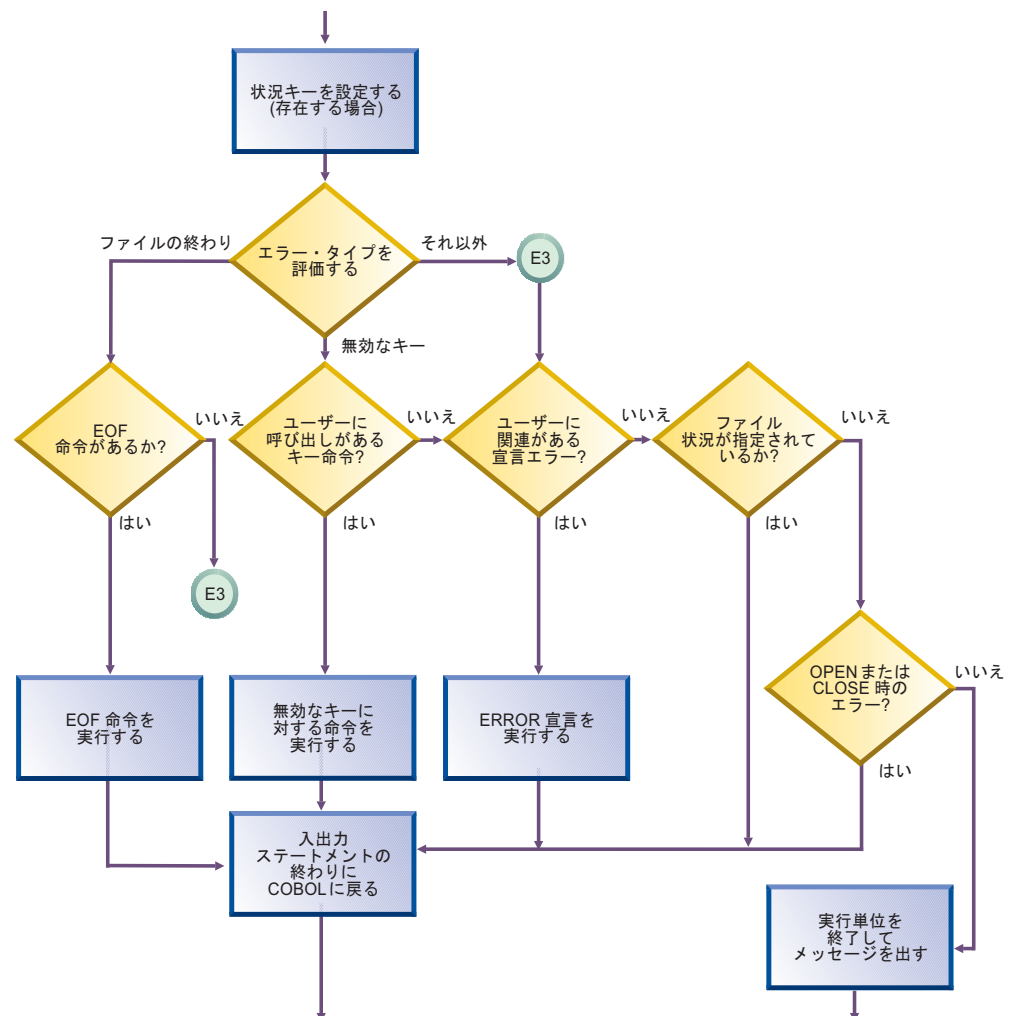
- ファイル終わり条件 (AT END)
- ERROR 宣言
- FILE STATUS 節およびファイル状況キー
- ファイル・システム状況コード
- READ または WRITE ステートメント内の命令ステートメント句

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。

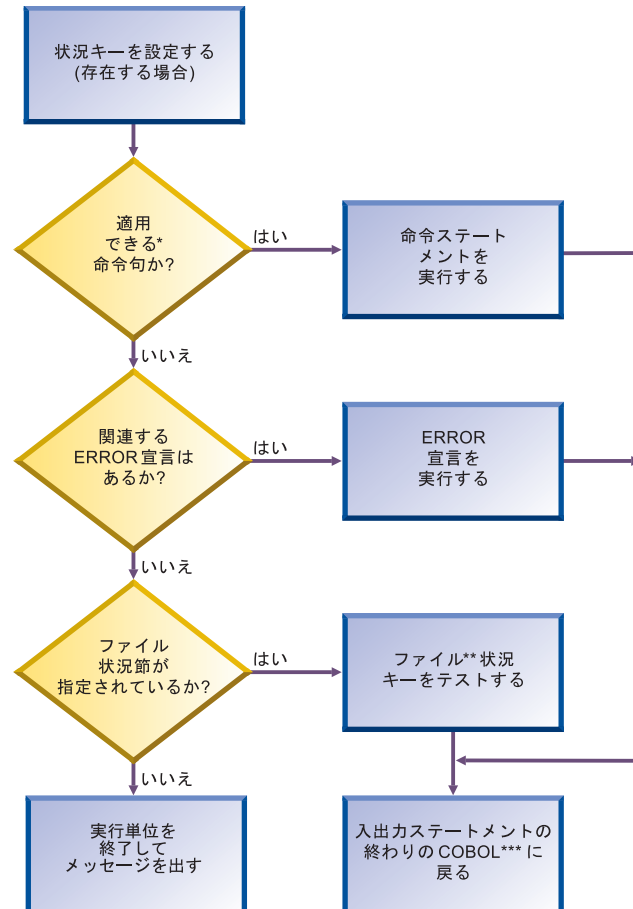
- INVALID KEY 句

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 の言語環境プログラム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 の場合の可能な句は、AT END、AT END-OF-PAGE、および INVALID KEY です。行順次の場合は、AT END です。

**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

- 276 ページの『ファイルの終わり条件 (AT END) の使用』
- 276 ページの『ERROR 宣言のコーディング』
- 277 ページの『ファイル状況キーの使用』
- 196 ページの『QSAM ファイルのエラーの処理』
- 279 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』
- 246 ページの『行順次ファイルのエラーの処理』
- 281 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。

多くの設計では、ファイルの終わりまでの順次読み取りが意図的に行われており、AT END 条件が予期されます。例えば、マスター・ファイルを更新するために、トランザクションが入っているファイル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 句はいずれも、READ ステートメントが正常に完了した場合にのみ実行されます。ファイルの終わり以外の条件が原因で READ 操作が失敗した場合は、AT END 句も NOT AT END 句も実行されません。その代わりに、関連する宣言型プロシージャを実行した後で、READ ステートメントの終わりに制御が渡されます。

AT END 句または EXCEPTION 宣言型プロシージャのいずれもコーディングせずに、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を 1 つ以上コーディングすることができます。そのようなプロシージャをコーディングしないと、入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。

このようなプロシージャをそれぞれ PROCEDURE DIVISION の宣言セクションに入れます。以下のものをコーディングすることができます。

- プログラム全体用の単一の共通プロシージャ
- 各ファイル・オープン・モードのプロシージャ (INPUT、OUTPUT、I-O、または EXTEND)
- それぞれのファイルごとの個々のプロシージャ

ERROR 宣言プロシージャでは、訂正処置の試行、操作の再試行、実行の継続または終了などをコーディングすることができます。(ただし、ブロック化ファイルの処理を続行すると、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おそれがあります。) エラーについてさらに詳しい分析を行う場合は、**ERROR** 宣言型プロシージャとファイル状況キーを組み合わせて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・アプリケーションで入出力宣言をコーディングする際には、デッドロックを避けてください。入出力操作の結果として制御権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の実行中は、ファイルに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。

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。

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。

ファイル状況キーは、次のようにコーディングして設定してください。

- **FILE-CONTROL** 段落の **FILE STATUS** 節:

```
FILE STATUS IS data-name-1
```

- DATA DIVISION (WORKING-STORAGE、LOCAL-STORAGE、または LINKAGE SECTION) のデータ定義 (一例として):

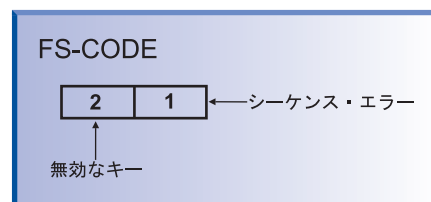
```
WORKING-STORAGE SECTION.  
01 data-name-1 PIC 9(2) USAGE NATIONAL.
```

ファイル状況キー *data-name-1* を、2 文字のカテゴリ英数字またはカテゴリ国別項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目として指定してください。この *data-name-1* を可変位置にすることはできません。

ご使用のプログラムで、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、また発生した場合はどのようなタイプのエラーが発生したかを検出できます。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。



各ファイルについて、以下の規則に従ってください。

- ファイルごとに異なるファイル状況キーを定義します。

すると、アプリケーション論理エラーやディスク・エラーのような、ファイル入力または出力例外の原因を判別することができます。

- それぞれの入力または出力要求の後で、ファイル状況キーを検査します。

ファイル状況キーが 0 以外の値を含んでいる場合、プログラムはエラー・メッセージを発生するか、またはその値に基づいてアクションを実行できます。

ファイル状況キー・コードをリセットする必要はありません。ファイル状況キー・コードは各入出力が試みられた後で設定されます。

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。

ファイル状況キーは単独でも、INVALID KEY 句と一緒にでも使用でき、EXCEPTION または ERROR 宣言を補足するためにも使用できます。このようにファイル状況キーを使用すると、それぞれの入力または出力操作の結果に関する正確な情報が得られます。

279 ページの『例: ファイル状況キー』

関連タスク

279 ページの『VSAM 状況コードの使用 (VSAM ファイルのみ)』

281 ページの『INVALID KEY 句のコーディング』

453 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 MASTERFILE ASSIGN TO AS-MASTERA  
    FILE STATUS IS MASTER-CHECK-KEY  
    . . .  
DATA DIVISION.  
    . . .  
WORKING-STORAGE SECTION.  
01 MASTER-CHECK-KEY          PIC X(2).  
    . . .  
PROCEDURE DIVISION.  
    OPEN INPUT MASTERFILE  
    IF MASTER-CHECK-KEY NOT = "00"  
        DISPLAY "Nonzero file status returned from OPEN " MASTER-CHECK-KEY  
    . . .
```

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。

FILE STATUS IS *data-name-1 data-name-8*

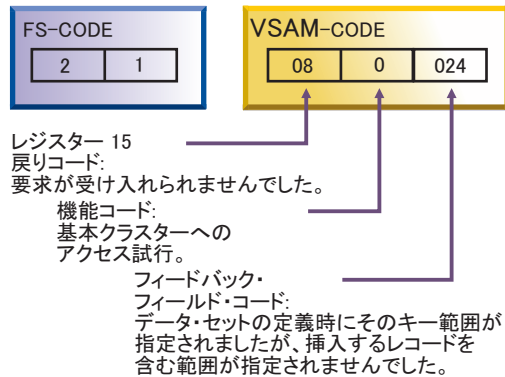
上記のデータ項目 *data-name-1* は COBOL ファイル状況キーを指定します。これは、2 文字の英数字または国別データ項目として、あるいは 2 桁のゾーン 10 進数または国別 10 進数項目として定義されます。

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。



『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。

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:

- 限定された種類のエラーに対してのみ働きます。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。
- 1 つの入出力操作に固有です。ERROR 宣言はより一般的です。

INVALID KEY 条件を引き起こすステートメントで INVALID KEY をコーディングすると、制御は INVALID KEY 命令ステートメントに転送されます。コーディングした ERROR 宣言は実行されません。

NOT INVALID KEY 句をコーディングした場合、ステートメントが正常に完了したときにのみ実行されます。INVALID KEY 以外の条件のために操作が失敗すると、INVALID KEY 句も NOT INVALID KEY 句も実行されません。代わりに、関連した ERROR 宣言をプログラムが実行した後、制御はステートメントの最後に渡されます。

『例: FILE STATUS および INVALID KEY』

例: FILE STATUS および INVALID KEY

次の例は、ファイル状況コードおよび INVALID KEY 句を使用して、入力または出力ステートメントが失敗した理由をもっと明確に判別する方法を示しています。

マスター顧客レコードを含むファイルがあり、トランザクション更新ファイルからの情報でそれらのレコードの一部を更新する必要があるとします。プログラムは、各トランザクション・レコードを読み取り、マスター・ファイル内で対応するレコードを検出し、必要な更新を行います。どちらのファイルのレコードにもそれぞれ顧客番号用のフィールドがあり、マスター・ファイルの中の各レコードには固有の顧客番号があります。

顧客レコードのマスター・ファイル用の FILE-CONTROL 記入項目には、索引編成を定義するステートメント、ランダム・アクセス、基本レコード・キーとして MASTER-CUSTOMER-NUMBER、およびファイル状況キーとして CUSTOMER-FILE-STATUS が含まれています。

```
.
. (read the update transaction record)
.
MOVE "TRUE" TO TRANSACTION-MATCH
MOVE UPDATE-CUSTOMER-NUMBER TO MASTER-CUSTOMER-NUMBER
READ MASTER-CUSTOMER-FILE INTO WS-CUSTOMER-RECORD
  INVALID KEY
    DISPLAY "MASTER CUSTOMER RECORD NOT FOUND"
    DISPLAY "FILE STATUS CODE IS: " CUSTOMER-FILE-STATUS
    MOVE "FALSE" TO TRANSACTION-MATCH
END-READ
```

プログラム呼び出し時のエラーの処理

プログラムが、別々にコンパイルされたプログラムを動的に呼び出すとき、呼び出されるプログラムが使用できないことがあります。例えば、システムがストレージ不足だったり、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。

一連のステートメントを実行してユーザー定義のエラー処理を行う場合は、ON EXCEPTION 句を使用します。例えば、以下のフラグメントでは、プログラム REPORTA が利用不可である場合、制御は ON EXCEPTION 句に渡されます。

```
MOVE "REPORTA" TO REPORT-PROG
CALL REPORT-PROG
  ON EXCEPTION
    DISPLAY "Program REPORTA not available, using REPORTB."
    MOVE "REPORTB" TO REPORT-PROG
    CALL REPORT-PROG
  END-CALL
END-CALL
```

ON EXCEPTION 句は、初期ロードでの呼び出し先プログラムの可用性に対してのみ適用されます。呼び出し先プログラムがロードされたが、何かしらの理由（初期設定など）で失敗した場合、ON EXCEPTION 句は実行されません。

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。

しかし、Enterprise COBOL および 言語環境プログラムでは、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。言語環境プログラム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。

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・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、402 ページの『OPTIMIZE』を参照してください。

言語環境プログラム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 言語環境プログラムに対して示し、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プログラム・サービスに渡すことができます。

関連タスク

568 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

402 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 14 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。

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 UNIX 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

- 288 ページの『JCL を使用したコンパイル』
- 295 ページの『TSO のもとでのコンパイル』
- 321 ページの『第 15 章 z/OS UNIX のもとでのコンパイル』
- 298 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』
- 300 ページの『コンパイラー入出力の定義』
- 307 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『複数プログラムのコンパイル (バッチ・コンパイル)』
- 316 ページの『ソース・プログラムのエラーの訂正』

関連参照

- 445 ページの『第 18 章 コンパイラー指示ステートメント』

300 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』

310 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 UNIX 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。

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とは、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。

594 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

『カタログ式プロシージャの使用』

293 ページの『プログラムをコンパイルするための JCL の作成』

307 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』

313 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』

592 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

300 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』

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。

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
//STEP1 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・セットに格納されている場合は、SYSIN DD * を、データ・セットを記述する適切なパラメーターに置き換えてください。

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、不要なパラメーターを検出すると、それらを見捨てるか、または代替パラメーターで置き換えます。

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、指定するコンパイラ・ユーティリティ・ファイル (SYSUTx) およびコピー・ライブラリー (SYSLIB) は MVS データ・セットでなければなりません。

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、言語環境プログラム情報の中で記載されています。

関連タスク

言語環境プログラム プログラミング・ガイド

関連参照

『コンパイル・プロシージャ (IGYWC)』
291 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)』
292 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。以下に例を示します。

```
//COBOL.SYSLIB DD DISP=SHR,DSN=DEPT88.BOBS.COBLIB
```

```
//IGYWC PROC  LNGPRFX='IGY.V6R2M0',
//          LIBPREFIX='CEE'
//*
//*  COMPILER A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE  USAGE
//*  LNGPRFX   IGY.V6R2M0      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 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=SYSLALDA,
// DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
//SYSUT1 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT8 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT9 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT10 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT11 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT12 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT13 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT14 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSUT15 DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))
//SYSMDECK DD UNIT=SYSLALDA,SPACE=(CYL,(1,1))

```

(1) STEPLIB は、インストール先によって異なります。

『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:

次のジョブは、プロシージャール IGYWC を使用して、z/OS UNIX ファイル・システムにある 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・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.COBIOL='LIST,MAP,RENT,FLAG(I,I),XREF,ADATA'
//SYSPRINT DD PATH='/u/userid/cobol/demo.lst', (1)
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC), (2)
// PATHMODE=SIRWXU, (3)
// FILEDATA=TEXT (4)
//SYSLIN DD PATH='/u/userid/cobol/demo.o',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//SYSADATA DD PATH='/u/userid/cobol/demo.adt',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//SYSIN DD PATH='/u/userid/cobol/demo.cbl',
// PATHOPTS=ORDONLY,
// FILEDATA=TEXT,
// RECFM=F

```

- (1) PATH は、z/OS UNIX ファイル・システムでのファイルのパス名を指定します。
- (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。

コンパイル用 DD ステートメント (この例ではオーバーライドとして示されています) では、z/OS UNIX ファイル・システム (PATH='unix-directory-path') 内のファイルと従来の MVS データ・セット (DSN=mvs-data-set-name) の組み合わせを使用できます。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 データ・セットでなければなりません。

関連参照

300 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』
UNIX システム・サービス・コマンド解説書
MVS JCL 解説書

コンパイルおよびリンク・エディット用プロシージャ (IGYWCL)

IGYWCL プロシージャは、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)
```

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。以下に例を示します。

```
//COBOL.SYSLIB DD DISP=SHR,DSN=DEPT88.BOBS.COBLIB

//IGYWCL PROC  LNGPRFX='IGY.V6R2M0',
//              LIBPRFX='CEE',
//              PGMLIB='&&GOSET',GOPGM=GO
//*
//*  COMPILE AND LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE  USAGE
//*  LNGPRFX    IGY.V6R2M0      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 DSN=&LNGPRFX..SIGYCOMP,DISP=SHR          (1)
//          DD DSN=&LIBPRFX..SCEERUN,DISP=SHR
//          DD DSN=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSN=&&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 DSNNAME=&LIBPRFX..SCEELKEX,DISP=SHR (2)
// DD DSNNAME=&LIBPRFX..SCEELKED,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSNNAME=&&LOADSET,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
//SYSLMOD DD DSNNAME=&PGMLIB(&GOPGM),
// SPACE=(CYL,(3,1,1)),
// UNIT=SYSALLDA,DISP=(MOD,PASS),DSNTYPE=LIBRARY
//SYSUT1 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。

(2) SYSLIB は、インストール先によって異なります。

コンパイル、リンク・エディット、実行用のプロシージャー (IGYWCLG)

IGYWCLG は、プログラムのコンパイル、リンク・エディット、および実行を行う 3 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。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。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。以下に例を示します。

```

//COBOL.SYSLIB DD DISP=SHR,DSN=DEPT88.BOBS.COBLIB
//IGYWCLG PROC LNGPRFX='IGY.V6R2M0',
// LIBPRFX='CEE',GOPGM=GO
//*
//* COMPILE, LINK EDIT AND RUN A COBOL PROGRAM
//*
//* PARAMETER DEFAULT VALUE USAGE
//* LNGPRFX IGY.V6R2M0 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 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=&&GOSSET(&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 DSNAME=&LIBPRFX..SCEERUN,DISP=SHR (1)
// DD 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ー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。

```

//jobname JOB acctno,name,MSGCLASS=1 (1)
//stepname EXEC PGM=IGYCRCTL,PARM=(options) (2)
//STEPLIB DD DSNAME=IGY.V6R2M0.SIGYCOMP,DISP=SHR (3)
// DD DSNAME=SYS1.SCEERUN,DISP=SHR
// DD DSNAME=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、ジョブの始まりを示します。
- (2) EXEC ステートメントは、Enterprise COBOL コンパイラー (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 コンパイラーが常駐するデータ・セットを定義します。
言語環境プログラム データ・セットが LNKLIST で使用可能でない場合は、言語環境プログラム SCEERUN データ・セットおよび SCEERUN2 データ・セットを (コンパイラー 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、コンパイラーがソース・プログラムを処理するために使用するユーティリティ・データ・セットを定義します。SYSUT ファイルはすべて、直接アクセス記憶装置上に置かなければなりません。
- (5) SYSPRINT DD ステートメントは、LIST や MAP などのコンパイラー・オプションからの出力を受け取るデータ・セットを定義します。SYSOUT=A は、システム出力装置を宛先とするデータ・セットの標準指定です。
- (6) SYSLIN DD ステートメントは、OBJECT コンパイラー・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するデータ・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用 DD ステートメントでは、z/OS UNIX ファイル・システム (PATH='unix-directory-path') 内のファイルと従来の MVS データ・セット (DSN=mvs-data-set-name) の組み合わせを使用できます。

- 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、コンパイラー・ユーティリティ・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。

JCL 内のカタログ式プロシージャーの詳しい例については、288 ページの『カタログ式プロシージャーの使用』とその後続トピックを参照してください。

『例: コンパイル用のユーザー作成 JCL』

594 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 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

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.V6R2M0.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 UNIX 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。

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。

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 (SYSUT n) 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・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・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・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.V6R2M0.SIGYCOMP(IGYCRCTL)'
```

ALLOCATE コマンドおよび **CALL** コマンドは、TSO コマンド行に指定することができます。また、z/OS UNIX ファイルを使用していない場合は、これらのコマンドを **CLIST** に指定することができます。

SYSUT x 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・データ・セットにも z/OS UNIX 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.cb1')
Pathopts(ORDONLY) Filedata(TEXT)
```

『例: TSO のもとでコンパイルするための **ALLOCATE** および **CALL**』

297 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

300 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』

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.V6R2M0.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.V6R2M0.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。

以下に例を示します。

```
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、COBOL*optionlist* 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 が入っていない必要があります。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 `SYSUT n` 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	SYPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSUT4
12	SYSTEM
13	SYSUT5
14	SYSUT6
15	SYSUT7
16	SYSADATA
17	SYSJAVA
18	該当なし
19	SYSMDECK
20	DBRMLIB
21	SYSOPTF
22	SYSUT8
23	SYSUT9
24	SYSUT10
25	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

『コンパイラー入出力の定義』

関連参照

『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』

310 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。

関連タスク

303 ページの『ソース・コード・データ・セットの定義 (SYSIN)』

303 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)』

304 ページの『ソース・ライブラリーの指定 (SYSLIB)』

304 ページの『出力データ・セットの定義 (SYSPRINT)』

305 ページの『コンパイラー・メッセージの端末への送信 (SYSTEM)』

305 ページの『オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)』

306 ページの『関連データ・ファイル (SYSADATA) の定義』

306 ページの『Java ソース出力ファイル (SYSJAVA) の作成』

307 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

『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』

310 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 34. コンパイラー・データ・セット

タイプ	DD 名	機能	必要か?	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	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	カード読取装置 ; 中間記憶装置; 直接アクセス	任意	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 (PDS または PDSE) の読み取り	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	直接アクセス	SYSDA	いいえ

表 34. コンパイラー・データ・セット (続き)

タイプ	DD 名	機能	必要か?	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?
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
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 TM 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 UNIX 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

表 34. コンパイラー・データ・セット (続き)

タイプ	DD 名	機能	必要か?	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?
SYSMDECK	MDECK オプシ ョンまたは作 業データ・セ ット (NOMDECK が指定された 場合) の処理	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

『論理レコード長とブロック・サイズ』

374 ページの『EXIT』

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 UNIX ファイル 以外のコンパイラ
ー・データ・セットの場合は、DCB パラメーターの BLKSIZE サブパラメーターを使
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 35. 固定長コンパイラー・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	$80 \times n$

表 35. 固定長コンパイラー・データ・セットのブロック・サイズ (続き)

データ・セット	RECFM	LRECL (バイト)	BLKSIZE ¹
SYSTEM	F または FB	80	80 x <i>n</i>
<ol style="list-style-type: none"> 1. <i>n</i> = 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 36. 可変長コンパイラー・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。

```
//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 ステートメントは、DD * ステートメントの後ろに続ける必要があります。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。

コンパイラー・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ー・オプションが入ったデータ・セットを定義します。

```
//SYSOPTF DD DSNAME=dsname,UNIT=SYSDA,VOLUME=(subparms),DISP=SHR
```

コンパイラー・オプション・データ・セットを使用するには、OPTFILE を、コンパイラー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ー・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、コンパイラー・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

302 ページの『論理レコード長とブロック・サイズ』

400 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。

```
//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。

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。以下に例を示します。

```
//SYSPRINT DD SYSOUT=A
```

SYSPRINT 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 z/OS UNIX ファイルにすることができます。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

302 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。

```
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)

OBJECT コンパイラー・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 UNIX ファイルとしてディスクに保管したり、あるいはテープに保管することができます。コンパイラーは、SYSLIN DD ステートメントまたは SYSPUNCH DD ステートメントで定義されたファイルを使用します。

```
//SYSLIN DD DSNAME=dsname,UNIT=SYSDA,  
// SPACE=(subparms),DISP=(MOD,PASS)
```

SYSLIN DD ステートメントの DISP パラメーターを使用して、オブジェクト・コード・データ・セットが以下のうちどのように処理されるかを示すことができます。

- 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。

インストール先で、DECK オプションおよび SYSPUNCH DD ステートメントを使用することができます。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

399 ページの『OBJECT』

367 ページの『DECK』

関連データ・ファイル (SYSADATA) の定義

ADATA コンパイラー・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。

```
//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 UNIX ファイルにすることができます。

関連参照

350 ページの『ADATA』

Java ソース出力ファイル (SYSJAVA) の作成

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。

```
//SYSJAVA DD PATH='/u/userid/java/Classname.java',  
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),  
// PATHMODE=SIRWXU,  
// FILEDATA=TEXT
```

SYSJAVA ファイルは、z/OS UNIX ファイル・システムに入っている必要があります。

関連タスク

339 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 からコンパイルするときに、TEST(. . .,SEP,. . .) コンパイラー・オプションを指定すると、シンボリック・デバッグ情報テーブルは、SYSDEBUG DD ステートメントで指定されるデータ・セットに書き込まれます。

```
//SYSDEBUG DD DSNAME=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。

言語環境プログラム は、そのダンプ・サービス用に SYSDEBUG を使用します。TEST|NOTEST(. . .,SEPARATE(DSNAME),...) コンパイラー・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。データ・セットの名前は、SYSDEBUG COBOL デバッグ・ファイル・ユーザー出口 (IGZIUXB) を使用して実行時に変更することができます。IBM z/OS Debugger は、SET DEFAULT LISTINGS コマンド、ユーザー出口 EQAUEDAT、または EQADEBUG DD ステートメントを使用して、名前を変更したデータ・セットに送信することができます。

DDNAME SYSDEBUG に指定したデータ・セット名は、複数の IBM 製品 (言語環境プログラム、IBM z/OS Debugger、Fault Analyzer、およびアプリケーション・パフォーマンス・アナライザーなど) が使用する可能性があります。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

言語環境プログラム カスタマイズ (COBOL デバッグ・ファイル名の修正)
Debug Tool User's Guide (Debug Tool が COBOL および PL/I の個別のデバッグ・ファイルの場所を探索する方法)

関連参照

302 ページの『論理レコード長とブロック・サイズ』
423 ページの『TEST』

ライブラリー処理出力ファイル (SYSMDECK) の定義

SYSMDECK データ・セットはすべてのコンパイルに必要です。MDECK コンパイラー・オプションを指定する場合、SYSMDECK DD 割り振りでは、永続データ・セットを指定する必要があります。ただし、NOMDECK オプションを使用する場合、SYSMDECK をユーティリティ (一時) データ・セットとして指定できます。

```
//SYSMDECK DD DSN=dsname,UNIT=SYSDA
```

SYSMDECK ファイルには、ライブラリー処理 (COPY、BASIS、REPLACE、EXEC SQL INCLUDE、および EXEC SQLIMS INCLUDE ステートメントの結果) の後で更新された入力ソースのコピーが入ります。このファイルは、従来の MVS データ・セットまたは z/OS UNIX ファイルにすることができます。

関連参照

390 ページの『MDECK』

z/OS 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、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、コンパイラー・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ー・オプションはオーバーライドできません。

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。
- コンパイラー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 コンパイラー・オプションを指定する。

コンパイラーは、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ー・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. コンパイラー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 コンパイラー・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ー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の対は XREF|NOXREF です。相互参照リストが必要な場合は、XREF を指定します。そうでない場合には、NOXREF を指定します。

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。

309 ページの『例: JCL によるコンパイラー・オプションの指定』

309 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

303 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)』

『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』

313 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

310 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』

345 ページの『第 17 章 コンパイラー・オプション』

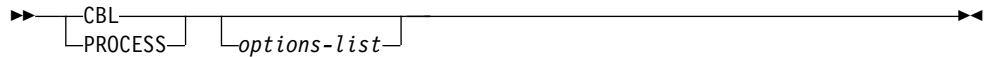
349 ページの『矛盾するコンパイラー・オプション』

400 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。

CBL/PROCESS 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 つ以上の空白で区切らなければなりません。オプションはそれぞれコンマまたは空白で区切ります。個々のオプションとそのサブオプションの間にスペースを入れてはなりません。

複数の PROCESS 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・オプションおよびコンパイラ出力

コンパイラがソース・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・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 37. z/OS のもとでのコンパイラ出力のタイプ

コンパイラ・オプション	コンパイラ出力	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
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 Debugger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 Debugger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 に入れます。進行メッセージと診断メッセージは、SYSTEM 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。独立したデバッグ・ファイルは、SYSDEBUG 内に定義されるデータ・セットです。

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

言語環境プログラム 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

317 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』

345 ページの『第 17 章 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)。

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。

『例: バッチ・コンパイル』

関連タスク

313 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

392 ページの『NAME』

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 は単独でリンク・エディットされ、PROG3 という名前のプログラム・オブジェクトが作成されます。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=&&GOSET          (1)
/*
//P2      EXEC PGM=PROG2
//STEPLIB DD  DSN=&&GOSET,DISP=(SHR,PASS) (2)
. . .    (3)
/*
//P3      EXEC PGM=PROG3
//STEPLIB DD  DSN=&&GOSET,DISP=(SHR,PASS) (2)
. . .    (4)
/*
//

```

- (1) LKED ステップ SYSLMOD のデータ・セット名が、一時名 &&GOSET に変更されます。メンバー名は指定しません。
- (2) 一時データ・セット &&GOSET は、コンパイル済みプログラムを実行するために、ステップ P2 および P3 の STEPLIB として使用されます。言語環境プログラム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

言語環境プログラム プログラミング・ガイド (IBM 提供のカatalog式プロシージャ)

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・シーケンスの各プログラムのコンパイラー・オプションは、プログラムの実行前またはコンパイラーの起動時に CBL または PROCESS ステートメントを使用して指定することができます。

CBL または PROCESS ステートメントが現行プログラムに指定されていると、コンパイラーは、最初のプログラムの前に有効なオプションと一緒に CBL または PROCESS ステートメントを解決します。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は、コンパイラーは、前のプログラムで有効であったオプションの設定を使用します。

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で有効にされた BUFSIZE、DEFINE、OUTDD、SQL、および SQLIMS コンパイラー・オプションの値
3. 現行プログラムに対する CBL または PROCESS ステートメント上のオプション (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で BUFSIZE、DEFINE、OUTDD、SQL、または SQLIMS オプションが必要な場合は、バッチ・シーケンス内の最初のプログラムでそのオプションを有効にする必要があります。(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。)

バッチで オプションを指定する場合は、バッチ・コンパイル時に 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、LIB オプションのもとで NUMBER および 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状態に戻します。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従います。

『例: バッチ・コンパイルでのオプションの優先順位』

315 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

368 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ー・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 6.2.0 P170724 Date 09/08/2017. . .
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 6.2.0 P170724 Date 09/08/2017. . .
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) **!!** 英数字ではないため廃棄されます。
- (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 (**JP**) です。したがっ

て、コンパイラーが CBL オプションの処理を終了すると、日本語が有効な言語になります。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。CBL ステートメントの言語オプションは、最後に指定された 2 文字の英数字言語 ID である YY を解決します。YY は存在しない言語であるため、有効な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、そのエラーが発生した場所 (LINEID) を示します。メッセージのテキストは、何が問題であるかを知らせます。この情報を使用して、ソース・プログラムを訂正することができます。

エラーを訂正するとしても、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ため、訂正しなければなりません。

低い方の 4 つのレベルの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。コンパイラー自体またはオペレーティング・システム内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。多数の S レベルの構文エラーがあるプログラムについてメッセージが出された場合は、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

317 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

317 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。

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

852 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

『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

316 ページの『ソース・プログラムのエラーの訂正』

『コンパイラー・メッセージのリストの生成』

関連参照

『コンパイラー診断メッセージの形式』

318 ページの『コンパイラー診断メッセージの重大度コード』

378 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 IGypxxxx-l message-text
```

nnnnnn

コンパイラーが処理している最後の行のソース・ステートメントの番号。ソース・ステートメント番号は、プログラムのソース印刷出力にリストされます。コンパイル時に NUMBER オプションを指定すると、番号は元のソース・プログラム番号になります。NONNUMBER 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、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

『コンパイラー診断メッセージの重大度コード』

378 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に分けられます。

表 38. 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	戻りコード	目的
通知 (I)	0	通知にすぎません。アクションを取る必要はありません。プログラムは正しく実行されます。
警告 (W)	4	エラーの可能性あることを示します。プログラムはおそらく、書かれたとおりに正しく実行されます。
エラー (E)	8	明確にエラーである条件があることを意味します。コンパイラーはエラーの訂正を試みましたが、プログラムの実行結果は予期したものではない可能性があります。エラーを訂正しなければなりません。
重大 (S)	12	重大なエラーを示す条件があることを意味します。コンパイラーはエラーを訂正できませんでした。プログラムは正しく実行されず、また、実行を試みてはなりません。オブジェクト・コードは作成されない可能性があります。
回復不能 (U)	16	コンパイルが終了するほどの重大なエラー条件があることを示します。

コンパイル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

852 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

850 ページの『MSGEXIT の処理』

第 15 章 z/OS UNIX のもとでのコンパイル

z/OS UNIX 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 UNIX 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。

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、コンピューターで処理できる言語 (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

『z/OS UNIX のもとでの環境変数の設定』

323 ページの『z/OS UNIX のもとでのコンパイラー・オプションの指定』

324 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』

330 ページの『スクリプトを使用したコンパイル』

333 ページの『z/OS UNIX 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

300 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』

310 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 UNIX 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。

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 UNIX 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んでください。以下に、その例を示します。

```
export COBOPT="TRUNC(OPT) XREF"
```

COBOL_INSTALL_DIR

通常、cob2 ユーティリティおよび関連ファイルは HFS ディレクトリー /usr/lpp/IBM/cobol/igyv6r2 の下にインストールされます。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。

COPY ステートメントに明示的なライブラリー名がコーディングされていない場合、コンパイラーは以下の順序で z/OS UNIX ディレクトリー内の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

検索するどの z/OS UNIX ディレクトリーにもコピーブックが見つからず、明示ライブラリー名なしで、またはライブラリー名 SYSLIB で指定された COPYLOC オプションのインスタンスがある場合、こ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

323 ページの『z/OS UNIX のもとでのコンパイラー・オプションの指定』

324 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』

542 ページの『環境変数の設定およびアクセス』

関連参照

445 ページの『第 18 章 コンパイラー指示ステートメント』

345 ページの『第 17 章 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 UNIX 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。

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. バッチ・コンパイルの最初のプログラムで有効にされた BUFSIZE、SQL、SQLIMS、および OUTDD オプションの値。
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 UNIX 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てします。
- `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 UNIX の下では動作しません。

- z/OS UNIX で、SQLIMS コンパイラー・オプションを使用しないでください。
- OPTFILE オプションは、z/OS UNIX で cob2 コマンドを使用してコンパイルした場合、無視されます。

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

308 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』
 321 ページの『z/OS UNIX のもとでの環境変数の設定』
 『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

349 ページの『矛盾するコンパイラー・オプション』
 345 ページの『第 17 章 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 UNIX 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。

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 UNIX 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.V6R2M0 をコンパイラー PDSE として指定します。

```
export STEPLIB=IGY.V6R2M0.SIGYCOMP
```

cob2 コマンドは、リンク・ステップには z/OS UNIX シェル・コマンド c89 を暗黙的に使用します。c89 は、リンカー (z/OS プログラム管理バインダー) へのシェル・インターフェースです。

コンパイラー入出力のデフォルトの位置は、現行ディレクトリーです。

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

『z/OS UNIX のもとでの DLL の作成』

334 ページの『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』
UNIX システム・サービス ユーザーズ・ガイド

関連参照

327 ページの『cob2 の構文およびオプション』

329 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 UNIX のもとでの DLL の作成

z/OS UNIX 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。

```
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` であれば、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
```

326 ページの『例: z/OS UNIX 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

591 ページの『第 26 章 DLL または DLL アプリケーションの作成』

592 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

327 ページの『cob2 の構文およびオプション』

329 ページの『cob2 入出力ファイル』

例: z/OS UNIX のもとでの cob2 によるコンパイルおよびリンク

次の例では、cob2 の使用法を示します。

- 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それらのファイルをコンパイルします。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 および NOADATA オプションを使用して 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
```

- 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、SQL オプションを付けて alpha.cbl をコンパイルするには、次のように入力します。

```
cob2 -qsql alpha.cbl -o alpha -dbrmli=USER.COBOL.DBRMLIB
```

注: これが動作するためには、SQL コプロセッサが STEPLIB になければなりません。

- データベース要求モジュール (DBRM) が z/OS UNIX ファイル alpha.dbrm に書き込まれるよう、SQL オプションを付けて alpha.cbl をコンパイルするには、次のように入力します。

```
cob2 -qsql alpha.cbl -o alpha -dbrmli
```

注: これを正常に実行するためには、SQL コプロセッサが STEPLIB になければなりません。また、Db2 V12 (APAR PI88171 適用済み) 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

```
▶▶cob2 [options] [filenames]▶▶
```

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tring xxx をパラメーターとしてリンカーに渡します。xxx は、コマンドで区切られた、*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 d11 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 String 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。

-dbrm1ib=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 UNIX 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。

-I_{xxx} *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 *L* ではありません。)

-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。

-q_{xxx} *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、または空白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 6.2.0 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

- 324 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』
- 325 ページの『z/OS UNIX のもとでの DLL の作成』
- 321 ページの『z/OS UNIX 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 39. 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 40. **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
ファイル	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。
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

324 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

350 ページの『ADATA』

390 ページの『MDECK』

423 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。

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。例えば、`-qOPTIMIZE(1),XREF` の代わりに、`-qOPTIMIZE=1:,XREF` をコーディングします。
- 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 16 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 UNIX 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。

関連タスク

『z/OS UNIX 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』
338 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』
343 ページの『Java SDKs for z/OS の使用』

z/OS UNIX 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 UNIX 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。

関連タスク

『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コンパイル』
334 ページの『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』
336 ページの『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実行』

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コンパイル

z/OS UNIX 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。

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構文を含む COBOL ソース・コードや、Java のサービスを使用する COBOL ソース・コードをコンパイルするには、RENT、DLL、THREAD、および DBCS コンパイラー・オプションを使用する必要があります。(RENT および 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。cob2 コマンドの -I オプションを使用するか、SYSLIB 環境変数を設定することによって、include サブディレクトリーを指定することができます。

関連タスク

- 321 ページの『第 15 章 z/OS UNIX のもとでのコンパイル』
- 『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』
- 336 ページの『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実行』
- 542 ページの『環境変数の設定およびアクセス』
- 743 ページの『JNI サービスへのアクセス』

関連参照

- 327 ページの『cob2 の構文およびオプション』
- 367 ページの『DBCS』
- 371 ページの『DLL』
- 409 ページの『RENT』
- 428 ページの『THREAD』

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

オブジェクト指向 COBOL アプリケーションをリンクするには、cob2 コマンドを使用します。

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 Software Development Kit で提供されています。
- igzcjava.x。cobol ディレクトリーの lib サブディレクトリーにおいて、z/OS UNIX ファイル・システム で提供されています。この DLL サイド・ファイルは、SCEELIB PDS (言語環境プログラムの一部) 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。

1. 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、lib`classname`.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。例えば、Account クラスが com.acme パッケージの一部である場合、外部クラス名 (クラスの

REPOSITORY 段落記入項目に定義されている) は com.acme.Account であり、このクラスの DLL モジュール名は libcom_acme_Account.so でなければなりません。

- 2. 生成された Java ソースを Java コンパイラーでコンパイルして、クラス・ファイル (.class) を生成します。

Classname に対応するクラス定義が格納されている COBOL ソース・ファイル Classname.cbl の場合は、次のコマンドを使用して、アプリケーションのコンポーネントのコンパイルとリンクを行います。

表 41. 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
cob2 -bdll -o libClassname.so Classname.o /usr/lpp/java/IBM/J8.0/bin/j9vm/libjvm.x /usr/lpp/cobol/igyv6r2/lib/igzcjava.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。

『例: z/OS UNIX のもとでの COBOL クラス定義のコンパイルとリンク』

関連タスク

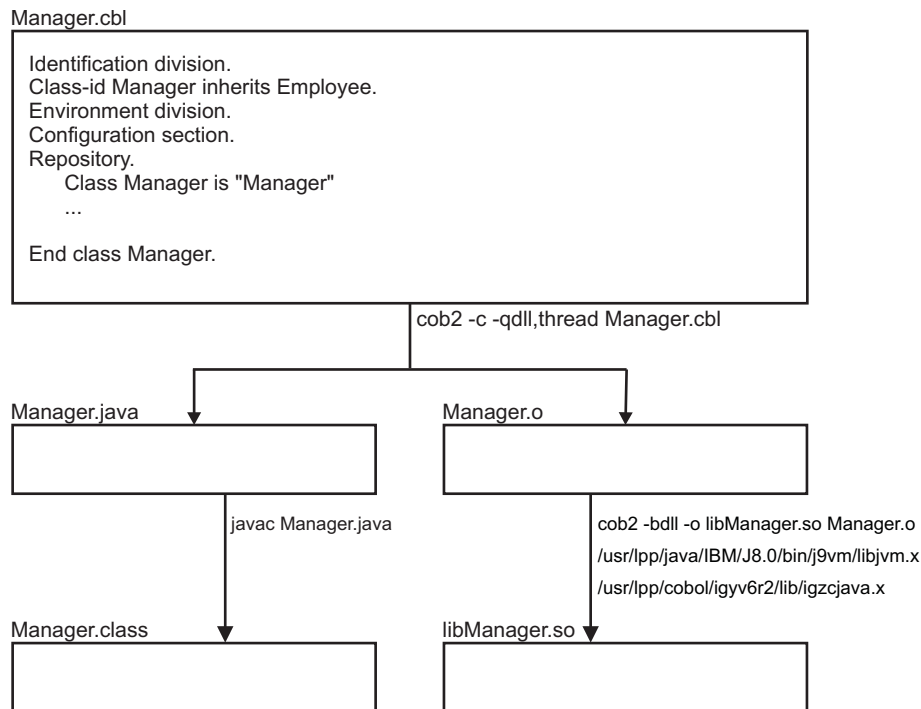
- 321 ページの『第 15 章 z/OS UNIX のもとでのコンパイル』
- 699 ページの『クラス定義用の REPOSITORY 段落』

関連参照

- 327 ページの『cob2 の構文およびオプション』
- 344 ページの『オブジェクト指向構文、および Java 6 以降』

例: z/OS UNIX のもとでの COBOL クラス定義のコンパイルとリンク

次の例は、z/OS UNIX のシェル・コマンドを使用して Manager.cbl という COBOL クラス定義をコンパイルおよびリンクするときに使用するコマンドと生成されるファイルを示したものです。



クラス・ファイル **Manager.class** と DLL モジュール **libManager.so** は、アプリケーションの実行可能コンポーネントであり、現行作業ディレクトリーに生成されます。

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実行

z/OS UNIX 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お勧めします。Java プログラムまたは COBOL クラスの **main** ファクトリー・メソッドでアプリケーションが始まる場合は、UNIX アプリケーションとして実行しなければなりません。

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。

関連タスク

337 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』

337 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』

338 ページの『J2EE COBOL クライアントの実行』

- 541 ページの『第 23 章 z/OS UNIX のもとでの COBOL プログラムの実行』
- 542 ページの『環境変数の設定およびアクセス』
- 693 ページの『第 33 章 オブジェクト指向プログラムの作成』
- 739 ページの『OO アプリケーションの構造化』

main メソッドで始まるオブジェクト指向アプリケーションの実行

COBOL と Java の混合アプリケーションの最初のルーチンが、Java クラスの main メソッドまたは COBOL クラスの main ファクトリー・メソッドである場合は、java コマンドを使用し、main メソッドを含むクラスの名前を指定して、アプリケーションを実行します。

java コマンドは、Java 仮想マシン (JVM) を初期化します。JVM の初期化をカスタマイズするには、以下の例で示すように、java コマンドのオプションを指定します。

表 42. JVM をカスタマイズするための Java コマンド・オプション

目的	オプション
システム・プロパティを指定する	<code>-Dname=value</code>
ガーベッジ・コレクションについての詳しいメッセージを JVM が生成するよう要求する	<code>-verbose:gc</code>
クラス・ロードに関する詳細メッセージを JVM が生成するよう要求する	<code>-verbose:class</code>
ネイティブ・メソッドおよび他の Java ネイティブ・インターフェース・アクティビティについての詳しいメッセージを JVM が生成するよう要求する	<code>-verbose:jni</code>
Java の初期ヒープ・サイズを <i>value</i> バイトに設定する	<code>-Xmsvalue</code>
Java の最大ヒープ・サイズを <i>value</i> バイトに設定する	<code>-Xmxvalue</code>

JVM がサポートするオプションの詳細については、java -h コマンドの出力、または関連参照を参照してください。

関連参照

IBM SDK for Java - Tools Documentation

WebSphere® for z/OS: アプリケーション (Java Naming and Directory Interface (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。

JVM の初期化をカスタマイズするには、COBJVMINITOPTIONS 環境変数を設定してオプションを指定します。オプションを区切るには、ブランクを使用します。以下に例を示します。

```
export COBJVMINITOPTIONS="-Xms100000000 -Xmx200000000 -verbose:gc"
```

関連タスク

- 343 ページの『Java SDKs for z/OS の使用』
- 541 ページの『第 23 章 z/OS UNIX のもとでの COBOL プログラムの実行』
- 542 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 Naming and Directory Interface (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。

COBOL J2EE クライアントを実行する前に、Java システム・プロパティー `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。以下に例を示します。

```
export COBJVMINITOPTIONS  
="-Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory"
```

756 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 UNIX 環境で行うことが推奨されます。

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

- 339 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』
- 340 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』
- 333 ページの『z/OS UNIX 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 を使用してオブジェクト指向 COBOL プログラムまたはクラス定義をコンパイルした場合は、通常、生成されるオブジェクト・ファイルは、DD 名が SYSLIN または SYSPUNCH であるデータ・セットに書き込まれます。コンパイラ・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 と DBCS はデフォルトです。

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 の呼び出し可能サービスにアクセスする場合は、z/OS UNIX ファイル・システム から JNI という名前の PDS メンバーまたは PDSE メンバーに JNI.cpy ファイルをコピーし、SYSLIB DD ステートメントでそのライブラリーを指定して、COBOL ソースでは COPY JNI の形式で 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。生成される Java ソース・ファイルを z/OS UNIX ファイル・システム 中のファイルに書き込むには、SYSJAVA DD 名を使用します。以下に例を示します。

```
//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。

Java クラス定義をコンパイルするには、z/OS UNIX シェルのコマンド・プロンプトから javac コマンドを使用するか、または BPXBATCH ユーティリティを使用します。

341 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

288 ページの『JCL を使用したコンパイル』

295 ページの『TSO のもとでのコンパイル』

304 ページの『ソース・ライブラリーの指定 (SYSLIB)』

306 ページの『Java ソース出力ファイル (SYSJAVA) の作成』

743 ページの『JNI サービスへのアクセス』

333 ページの『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コンパイル』

334 ページの『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』

関連参照

367 ページの『DBCS』

371 ページの『DLL』

409 ページの『RENT』

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 UNIX 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。

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 UNIX 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。オブジェクト指向の構文を含んだ 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。
- 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。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 igzjava.x に対して、INCLUDE 制御ステートメントを指定します。以下に、その例を示します。

```
| INCLUDE '/usr/lpp/java/IBM/J8.0/bin/j9vm/libjvm.x'
| INCLUDE '/usr/lpp/cobol/igv6r2/lib/igzjava.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 は、PATH、LIBPATH、および CLASSPATH の環境変数を設定する以下の 3 行を含んで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は、 JAVAERR DD を使用します。
- STEPLIB DD ステートメントなどを使用して、SCEERUN2 と SCEERUN のロード・ライブラリーをシステム・ライブラリーの探索順序でできるようにします。

『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

334 ページの『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』

336 ページの『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実行』

739 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 (BPXBATC ユーティリティ)
言語環境プログラム 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)

言語環境プログラム 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 UNIX 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.V6R2M0'
// 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/igyv6r2/lib/igzccjava.x'
/*
//GO EXEC PGM=TSTHELLO,COND=(4,LT,LKED),
//        PARM='/ENVAR("_CEE_ENVFILE=/u/userid/ootest/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/ootest/tsthello/javaout',
//      PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//      PATHMODE=(SIRUSR,SIWUSR,SIRGRP)
//JAVAERR DD PATH='/u/userid/ootest/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。結果として生成される .class ファイルは、z/OS UNIX ファイル・システム ディレクトリー `u/userid/ootest/tsthello` に格納されます。このディレクトリーは、環境変数設定ファイルの **CLASSPATH** 環境変数で指定されています。

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/ootest/tsthello
```

環境変数設定ファイルは、`u/userid/ootest/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、言語環境プログラムで定義されている XPLINK リンケージ規約に基づきます。

アプリケーションが、Java プログラムから、または COBOL クラスの main ファクトリー・メソッドから開始される場合、JVM を開始しアプリケーションを実行する java コマンドによって XPLINK 環境は自動的に開始されます。

COBOL クラスまたは Java クラスのメソッドを呼び出す COBOL プログラムからアプリケーションが開始される場合、XPLINK 環境が初期化されるように XPLINK(ON) ランタイム・オプションを指定する必要があります。ただし、XPLINK(ON) をデフォルト設定にすることはお勧めしません。XPLINK(ON) は、明確にこの設定を必要とするアプリケーションにのみ使用してください。

z/OS UNIX のもとでアプリケーションを実行する場合、次のように **_CEE_RUNOPTS** 環境変数を使用して XPLINK(ON) オプションを設定できます。

```
_CEE_RUNOPTS="XPLINK(ON)"
```

ただし、z/OS UNIX シェル・セッション全体で有効になるように **_CEE_RUNOPTS="XPLINK(ON)"** をエクスポートすることはお勧めしません。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す。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

336 ページの『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実行』

542 ページの『環境変数の設定およびアクセス』

関連参照

『オブジェクト指向構文、および Java 6 以降』

543 ページの『ランタイム環境変数』

言語環境プログラム 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 V5.2 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 Enterprise COBOL アプリケーションの旧バージョンは、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、以下の手順を行ってください。

1. Enterprise COBOL V5.2 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. Java 6 以降の `javac` コマンドを使用して、各オブジェクト指向 COBOL クラスに関連付けられている、生成された Java クラスを再コンパイルします。

関連タスク

334 ページの『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』

第 17 章 コンパイラー・オプション

コンパイルに対する指示および制御の方法には、コンパイラー・オプションを使用する方法と、コンパイラー指示ステートメント (コンパイラー指示) を使用する方法とがあります。

コンパイラー・オプションは、次の表にリストされているプログラムの局面に影響を与えます。各オプションに結び付けられている情報は、そのオプションを指定するための構文を与えるもので、オプション、そのパラメーター、および他のパラメーターとの相互作用を説明しています。

表 43. コンパイラー・オプション

プログラムの局面	コンパイラー・オプション	デフォルト	オプションの省略形
ソース言語	352 ページの『APOST/QUOTE』	QUOTE	APOST Q
	354 ページの『ARITH』	ARITH (COMPAT)	AR (C E)
	358 ページの『CICS』	NOCICS	なし
	359 ページの『CODEPAGE』	CODEPAGE (1140)	CP (ccsid)
	365 ページの『CURRENCY』	NOCURRENCY	CURR NOCURR
	367 ページの『DBCS』	DBCS	なし
	393 ページの『NSYMBOL』	NSYMBOL (NATIONAL)	NS (DBCS NAT)
	394 ページの『NUMBER』	NONUMBER	NUM NONUM
	408 ページの『QUALIFY』	QUALIFY (COMPAT)	QUA (C E)
	414 ページの『SEQUENCE』	SEQUENCE	SEQ NOSEQ
	416 ページの『SQL』	NOSQL	なし
	417 ページの『SQLCCSID』	SQLCCSID	SQLC NOSQLC
	418 ページの『SQLIMS』	NOSQLIMS	なし
	422 ページの『SUPPRESS』	SUPPRESS	SUPP
	435 ページの『WORD』	NOWORD	WD NOWD
	436 ページの『XMLPARSE』	XMLPARSE (XMLSS)	XP (X) XP (C)
日付処理	385 ページの『INTDATE』	INTDATE (ANSI)	なし
マップおよびリスト	385 ページの『LANGUAGE』	LANGUAGE (ENGLISH)	LANG (EN UE JA JP)
	386 ページの『LINECOUNT』	LINECOUNT (60)	LC
	387 ページの『LIST』	NOLIST	なし
	388 ページの『MAP』	NOMAP	なし
	399 ページの『OFFSET』	NOOFFSET	OFF NOOFF
	415 ページの『SOURCE』	SOURCE	S NOS
	416 ページの『SPACE』	SPACE (1)	なし
	423 ページの『TERMINAL』	NOTERMINAL	TERM NOTERM
	433 ページの『VBREF』	NOVBREF	なし
	437 ページの『XREF』	XREF (FULL)	X NOX

表 43. 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・デックの生成	362 ページの『COMPILE』	NOCOMPILE(S)	C NOC
	364 ページの『著作権』	NOCOPYRIGHT	CPYR NOCPYR
	367 ページの『DECK』	NODECK	D NOD
	392 ページの『NAME』	NONAME、または NAME(NOALIAS) (NAME だけが指定された場合)	なし
	399 ページの『OBJECT』	OBJECT	OBJ NOOBJ
	405 ページの『PGMNAME』	PGMNAME(COMPAT)	PGMN(CO LU LM)
	415 ページの『SERVICE』	NOSERVICE	SERV NOSERV
オブジェクト・コード制御	351 ページの『ADV』	ADV	なし
	351 ページの『AFP』	AFP(NOVOLATILE)	なし
	352 ページの『ARCH』	ARCH(7)	なし
	355 ページの『AWO』	NOAWO	なし
	356 ページの『BLOCK0』	NOBLOCK0	なし
	368 ページの『DEFINE』	NODEFINE	DEF NODEF
	370 ページの『DISPSIGN』	DISPSIGN(COMPAT)	DS(S C)
	371 ページの『DLL』	NODLL	なし
	377 ページの『EXPORTALL』	NOEXPORTALL	EXP NOEXP
	377 ページの『FASTSRT』	NOFASTSRT	FSRT NOFSRT
	381 ページの『HGPR』	HGPR(PRESERVE)	なし
	384 ページの『INLINE』	INLINE	INL NOINL
	389 ページの『MAXPCF』	MAXPCF(100000)	なし
	395 ページの『NUMCHECK』	NONUMCHECK	NC NONC
	398 ページの『NUMPROC』	NUMPROC(NOPFD)	なし
	402 ページの『OPTIMIZE』	OPTIMIZE(0)	OPT(n)
	403 ページの『OUTDD』	OUTDD(SYSOUT)	OUT
	404 ページの『PARMCHECK』	NOPARMCHECK	PC NOPC
	430 ページの『TRUNC』	TRUNC(STD)	なし
	433 ページの『VLR』	VLR(STD)	VLR(C S)
	438 ページの『ZONECHECK』	NOZONECHECK	NOZC ZC(MSG) ZC(ABD)
	440 ページの『ZONEDATA』	ZONEDATA(PFD)	ZD(PFD) ZD(MIG) ZD(NOPFD)
	443 ページの『ZWB』	ZWB	なし
仮想記憶域の使用量	357 ページの『BUFSIZE』	4096	BUF
	366 ページの『DATA』	DATA(31)	なし
	373 ページの『DYNAM』	NODYNAM	DYN NODYN
	409 ページの『RENT』	RENT	なし
	410 ページの『RMODE』	AUTO	なし
	421 ページの『STGOPT』	NOSTGOPT	SO NOSO

表 43. 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
デバッグと診断	369 ページの『DIAGTRUNC』	NODIAGTRUNC	DTR NODTR
	372 ページの『DUMP』	NODUMP	DU NODU
	378 ページの『FLAG』	FLAG(I,I)	F NOF
	379 ページの『FLAGSTD』	NOFLAGSTD	なし
	382 ページの『INITCHECK』	NOINITCHECK	IC NOIC
	411 ページの『RULES』	NORULES	RULES(ENDP,EVENP,LXPRF, SLCKB,OOM) RULES(NOENDP, NOEVENP,NOLXPRF,NOSLCKB, NOOOM,NOUNRA NOUNRS)
	420 ページの『SSRANGE』	NOSSRANGE	SSR(ZLEN NOZLEN,MSG ABD) NOSSR
	423 ページの『TEST』	NOTEST	なし
その他	350 ページの『ADATA』	NOADATA	なし
	362 ページの『COPYLOC』	NOCOPYLOC	CPLC
	374 ページの『EXIT』	NOEXIT	NOEX EX(INX NOINX, LIBX NOLIBX, PRTX NOPRTX, ADX NOADX, MSGX NOMSGX)
	390 ページの『MDECK』	NOMDECK	NOMD MD MD(C NOC)
	400 ページの『OPTFILE』	なし	なし
	428 ページの『THREAD』	NOTHREAD	なし
	434 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。

パフォーマンスに関する考慮事項: The AFP、ARCH、ARITH、AWO、BLOCK0、DYNAM、FASTSRT、HGPR、MAXPCF、NUMCHECK、NUMPROC、OPTIMIZE、PARMCHECK、RENT、SQLCCSID、SSRANGE、STGOPT、TEST、THREAD、TRUNC、ZONECHECK、および ZONEDATA コンパイラー・オプションは、実行時のパフォーマンスに影響を及ぼすことがあります。

関連タスク

- 287 ページの『第 14 章 z/OS のもとでのコンパイル』
- 295 ページの『TSO のもとでのコンパイル』
- 321 ページの『第 15 章 z/OS UNIX のもとでのコンパイル』
- 779 ページの『第 36 章 プログラムのチューニング』

関連参照

- 349 ページの『矛盾するコンパイラー・オプション』
- 445 ページの『第 18 章 コンパイラー指示ステートメント』
- 『85 COBOL 標準に準拠するオプション設定』
- 788 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

FLAGSTD コンパイラー・オプションを使用して、IBM 拡張などの非準拠エレメントにフラグを立てることができます。

以下のランタイム・オプションが必要です。

- AIXBLD
- CBLQDA(ON)
- TRAP(ON)

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。

矛盾するオプションが階層中の同じレベルに指定されている (例えば、PROCESS または CBL ステートメントに DECK と NODECK の両方が指定されている) 場合は、最後に指定し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 の両方を指定すると、指定した順序に関係なく、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 ステートメントでは OFFSET をコーディングし、PROCESS ステートメントでは LIST をコーディングすると、PROCESS ステートメントでコーディングされたオプションと、PROCESS ステートメントで強制的にオンにされたオプションの優先順位の方が高いため、LIST が有効になります。

表 44. 相互に排他的なコンパイラー・オプション

指定される	無視される ¹	強制的にオンにされる ¹
CICS	DYNAM	NODYNAM
	NORENT	RENT
DLL	DYNAM	NODYNAM
	NORENT	RENT
EXPORTALL	NODLL	DLL
	DYNAM	NODYNAM
	NORENT	RENT
NORENT	RMODE (ANY)	RMODE (24)
NSYMBOL (NATIONAL)	NODBCS	DBCS
OBJECT	DECK	NODECK
OFFSET	LIST	NOLIST
PGMNAME (LM LU)	NAME	NONAME
TEST	NOOBJECT および NODECK	OBJECT および NODECK
THREAD	INITIAL	NOINITIAL
	NORENT	RENT
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。		

I

関連タスク

- 307 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』
- 313 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』
- 323 ページの『z/OS UNIX のもとでのコンパイラー・オプションの指定』

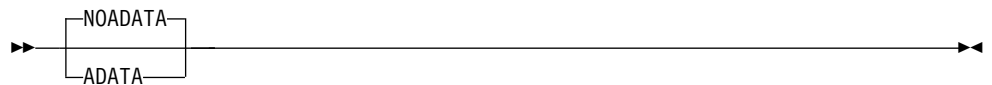
関連参照

- 400 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: 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
- cob2 コマンド・オプションとして
- 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

- 321 ページの『z/OS UNIX のもとでの環境変数の設定』
- 327 ページの『cob2 の構文およびオプション』
- 871 ページの『付録 G. 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: 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、z/Architecture® プロセッサに備わっている追加浮動小数点 (AFP) レジスタのコンパイラによる使用を制御します。

AFP オプションの構文



デフォルト: AFP(NOVOLATILE)

省略形: なし

Enterprise COBOL コンパイラーは、z/Architecture プロセッサに備わっている 16 個の浮動小数点レジスター (FPR) をすべて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)を指定した場合、AFP レジスター 8 から 15 は揮発性であると見なされます。つまり、これらのレジスター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ります。このため、COBOL コンパイラ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) を指定した場合、AFP レジスター 8 から 15 は不揮発性であるとみなされます。つまり、これらのレジスターは呼び出されたサブプログラムによって変更されないこと、または保持されることが分かっています。このため、コンパイラ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。

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



デフォルト: QUOTE

省略形: Q|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の生成対象となるマシン・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: ARCH(7)

省略形: なし

より高い ARCH レベルを指定すると、コンパイラーは、新しいより高速な命令を使用するコードを生成します。ご使用のアプリケーションは、ARCH オプションで指定したものよりも低いアーキテクチャー・レベルのプロセッサで実行した場合、異常終了するおそれがあります。アプリケーションを実行する最もレベルの低いマシン・アーキテクチャーに合致する ARCH レベルを使用してください。

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。

- 7 z/Architecture モードの 2094-xxx (IBM System z9[®] EC) モデルおよび 2096-xxx (IBM System z9 BC) モデルで有効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(7) マシンおよびその後継マシンには、以下の機能でサポートされる命令が追加されています。

- 拡張即時 (Extended-immediate) 機能
- 10 進数浮動小数点機能。これらの命令は、10 進数データが数値演算で使用されている場合に生成される可能性があります。

- 8 z/Architecture モードの 2097-xxx (IBM System z10[®] EC) モデルおよび 2098-xxx (IBM System z10 BC) モデルで有効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(8) マシンおよびその後継マシンには、汎用命令拡張機能でサポートされる命令が追加されています。

- 9 z/Architecture モードの 2817-xxx (IBM zEnterprise[®] 196) モデルおよび 2818-xxx (IBM zEnterprise 114) 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(9) マシンおよびその後継マシンには、以下の機能でサポートされる命令が追加されています。

- 上位ワード機能
- インターロック・アクセス機能
- 条件付きロード/ストア機能
- 独立オペランド機能
- ポピュレーション・カウント機能

- 10 z/Architecture モードの 2827-xxx (IBM zEnterprise EC12) モデルおよび 2828-xxx (IBM zEnterprise BC12) モデルで有効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(10) マシンおよびその後継マシンには、以下の機能でサポートされる命令が追加されています。

- 実行ヒント機能
- ロード・アンド・トラップ機能
- 各種命令拡張機能
- トランザクション実行機能

- ゾーン 10 進数データ項目と 10 進浮動小数点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。算術計算を実行するためにゾーン 10 進数データ項目をパック 10 進数データ項目に変換する代わりに、コンパイラはゾーン 10 進数データ項目を 10 進浮動小数点データ項目に直接変換し、計算の完了後に再度ゾーン 10 進数データ項目に戻します。

11 z/Architecture モードの 2964-xxx (IBM z13[®]) モデルおよび 2965-xxx (IBM z13s[®]) モデルで有効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) マシンおよびその後継マシンには、以下の機能のサポートを使用する命令が追加されています。

- 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。
- いくつかの INSPECT REPLACING ステートメントおよび INSPECT TALLYING ステートメントのための、ベクトル拡張機能 (SIMD) 命令の利用。

ベクトル拡張機能 (SIMD) 命令を使用するには、APAR OA43803 および PI12412 の PTF がインストールされている z/OS V2.1、または z/OS V2.2 で稼働するマシンにおいてコードを実行する必要があります。

12 z/Architecture モードの 3906-xxx (IBM z14) および 3907-xxx (IBM z14 ZR1) モデルで利用できる命令が使用されるコードが生成されます。

具体的には、ARCH(12) マシンとその後継マシンでは、ベクトル・パック 10 進数機能をサポートする命令が追加されます。ベクトル・パック 10 進数機能では、中間結果がメモリーではなくベクトル・レジスターに格納されるためパック 10 進数およびゾーン 10 進数の計算が高速になります。

注: より高い ARCH レベルには、低い ARCH レベルの機能が含まれています。例えば、ARCH(12) には、それよりも低い ARCH レベルの機能がすべて含まれています。

これらの機能について詳しくは、「z/Architecture 解説書」を参照してください。

ARITH

ARITH は、整数にコーディングできる最大桁数、および固定小数点の中間結果で 사용되는桁数に影響を与えます。

ARITH オプションの構文

▶▶ ARITH(COMPAT
EXTEND) ▶▶

デフォルト: ARITH(COMPAT)

省略形: AR(C|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桁数の最大数が、18 から 31 に上がります。以下を含め、数値リテラルが現在許可されているどの場所でも、長精度の数値リテラルを使用できます。
 - 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、 および数字編集の各データ項目の PICTURE 節の桁位置の最大数は 18 です。
- 固定小数点数値リテラルに指定できる桁数の最大数は、18 です。
- NUMVAL、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は、18 です。
- FACTORIAL 関数への整数引数の最大値は、28 です。
- 算術ステートメントの中間結果は、互換モード を使用します。

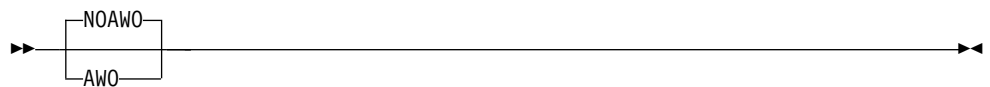
関連概念

805 ページの『付録 A. 中間結果および算術精度』

AWO

AWO を指定すると、暗黙の APPLY WRITE-ONLY 節が、ブロック化可変長レコードが入った、プログラム内のすべての QSAM ファイルに対して活動化されます。

AWO オプションの構文



デフォルト: 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

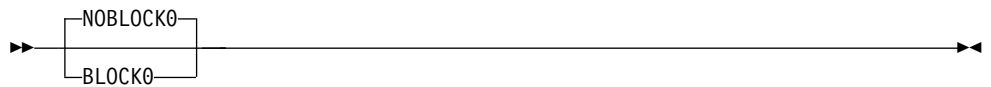
『BLOCK0』

APPLY WRITE-ONLY 節 (*Enterprise COBOL for z/OS 言語解説書*)

BLOCK0

BLOCK0 を使用して、(BLOCK CONTAINS 0 を指定した場合と同じように) QSAM ファイルのコンパイラ・デフォルトを非ブロック化からブロック化に変更し、出力ファイルに関するシステム決定のブロック化の利点を利用します。

BLOCK0 オプションの構文



デフォルト: 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。

- FILE-CONTROL 段落で、ORGANIZATION SEQUENTIAL が指定されているか、ORGANIZATION 節が省略されています。
- FD 項目に RECORDING MODE U が指定されていません。
- FD 項目に BLOCK CONTAINS 節が指定されていません。

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:

- NOBLOCK0 が有効で、上記の 3 つの基準を満たすファイルのファイル記述に APPLY WRITE-ONLY が指定されている場合、APPLY WRITE-ONLY はブロック化ファイルのみに適用されるため、コンパイラがエラー・メッセージを発行します。しかし、BLOCK0 が有効の場合には、結果としてファイルがブロック化され、APPLY WRITE-ONLY 節は受諾されます。
- 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場合があります。

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。以下に例を示します。

- 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。

ブロック化に関する推奨については、「Enterprise COBOL for z/OS 移行ガイド」の関連参照 (iCMPR2 から NOCMPR2 への移行に関する情報の中) を参照してください。

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』

188 ページの『ブロック・サイズの設定』

関連参照

355 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 は、コンパイラ作業データ・セットごとに、ある容量の主記憶域をバッファに割り振るために使用します。通常、バッファ・サイズを大きくすると、コンパイラのパフォーマンスが向上します。

BUFSIZE オプションの構文

➡➡—BUFSIZE($\overbrace{\text{nnnnn}}^{\text{nnnK}}$)—➡➡

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。

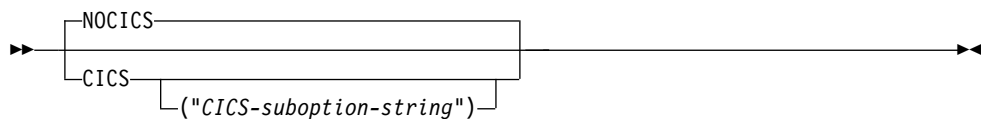
nnnK は、1 KB 単位で 10 進数を指定します。1KB = 1024 バイトです。

BUFSIZE は、使用される装置のトラック容量を超えてはならず、また、データ管理サービスで許可される最大量を超えてはなりません。

CICS

CICS コンパイラー・オプションを指定すると、組み込みの CICS 変換プログラムが使用可能になり、CICS サブオプションを指定できるようになります。COBOL ソース・プログラムに EXEC CICS ステートメントまたは EXEC DLI ステートメントが含まれており、プログラムが別個の CICS 変換プログラムで処理されていない場合は、CICS オプションを使用する必要があります。

CICS オプションの構文



デフォルト: NOCICS

省略形: なし

CICS オプションは、CICS プログラムをコンパイルする場合にのみ使用してください。CICS オプションを指定してコンパイルされたプログラムは、非 CICS 環境では実行することができません。

NOCICS オプションを指定した場合、ソース・プログラム内で検出された CICS ステートメントはすべて診断され、破棄されます。

引用符または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。例えば。

```
//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。

関連概念

508 ページの『組み込みの CICS 変換プログラム』

関連タスク

506 ページの『CICS オプションを使用したコンパイル』

508 ページの『CICS サブオプションの分離』

CICS Application Programming Guide (CICS 変換プログラム・オプションの指定)

関連参照

349 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: 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。

- COBOL ソース・プログラム内における英数字、国別、および DBCS リテラルのエンコード
- 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
- XML エlementおよび属性名を作成するために XML GENERATE ステートメントが処理する際の、DBCS ユーザー定義語のエンコード
- 文書の受け取りデータ項目が英数字である場合に XML GENERATE ステートメントが作成する XML 文書のデフォルト・エンコード
- 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
- 移動操作の一部としての英数字データと国別 (Unicode) データ間の変換、比較、または組み込み関数 DISPLAY-OF および 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 ストリング・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。

- 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、デフォルト通貨記号は (CURRENCY コンパイラー・オプションまたは SPECIAL-NAMES 段落の CURRENCY SIGN 節によって変更されていない限り) 常に値が X'5B' の文字であり、引用符は常に値が X'7F' の文字になります。

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。

- コード・ページを第 2 引数として指定する、DISPLAY-OF および NATIONAL-OF 組み込み関数
- WITH ENCODING 句を指定する XML PARSE ステートメント
- WITH ENCODING 句を指定する XML GENERATE ステートメント

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す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する通貨記号

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 45. 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。

関連概念

522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

72 ページの『通貨記号の使用』

627 ページの『第 31 章 XML 入力の処理』

675 ページの『第 32 章 XML 出力の生成』

関連参照

365 ページの『CURRENCY』

417 ページの『SQLCCSID』

423 ページの『TEST』

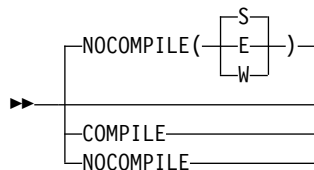
648 ページの『XML 文書のエンコード方式』

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: NOCOMPILE(S)

省略形: 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

457 ページの『コーディング・エラーの検出』

関連参照

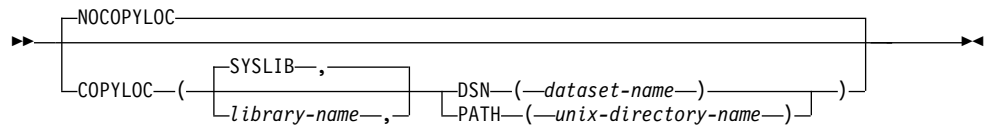
317 ページの『コンパイラ検出エラーに関するメッセージおよびリスト』

COPYLOC

COPYLOC コンパイラ・オプションを使用して、PDSE (または PDS) データ・セット、あるいは z/OS UNIX ディレクトリを、ライブラリー・フェーズ中に COPY メンバーを探索する場所として追加します。COPYLOC オプションで指定された場所は、COPY メンバーを探索する場所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。

COPYLOC オプションの構文



デフォルト: NOCOPYLOC

省略形: CPLC | NOCPLC

library-name

コピー場所が関連付けられるライブラリー名。 *library-name* が指定されない場合、デフォルトは SYSLIB です。これは、明示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 *library-name* を参照する COPY ステートメントの処理時に、コンパイラーが COPY メンバーを探索する PDS データ・セットまたは PDSE データ・セットの名前。

unix-directory-name

ライブラリー *library-name* を参照する COPY ステートメントの処理時に、コンパイラーが COPY メンバーを探索する z/OS UNIX 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 UNIX 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 オプションの複数のインスタンスがサポートされます。指定可能な z/OS UNIX 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・セットには 256 の制限があります。コピー場所は、COPYLOC オプションで指定した順に探索されます。これはユーザーに、探索内で PDSE (または PDS) の場所と z/OS UNIX ディレクトリーを混在させる機能を提供します。

NOCOPYLOC オプションが指定された場合、COPYLOC オプションの前のインスタンスは無視されます。

COPYLOC オプションを排他使用して COPY メンバーの探索を制御するには、コピーブックの場所を指示するための既存の方法 (データ・セットを JCL 内の DD 名に割り当てる、cob2 コマンドの -I オプションを指定するなど) をいずれも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、現行ディレクトリー内に COPY メンバーを保存しないようにすることも必要です。現行ディレクトリーは常に、COPYLOC の場所が探索される前に探索されるためです。

ヒント: COPYLOC オプションを排他使用して COPY メンバーの探索を制御するのが便利です (特にコンパイラーが cob2 コマンドから呼び出される場合)。

CBL ステートメントで指定される COPYLOC 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 COPYLOC 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 COPYLOC オプション (および JCL の PARM で指定された COPYLOC 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 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場所は、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場所の後に探索されます。

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、先行するオプションがコンパイラの単一呼び出しで指定され、JCL で指定された場所 (または cob2 コンパイル用に指示された場所) に COPY メンバーが見つからないと、COPY メンバーの追加の探索が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 UNIX ディレクトリー /home/userid/copylib1
3. USERID.COBOL.COPYLIB2 データ・セット

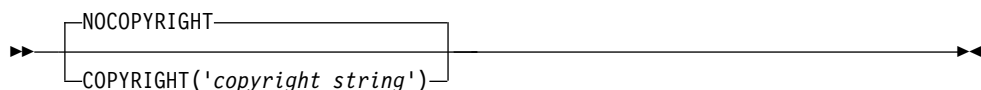
関連参照

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*)

著作権

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: NOCOPYRIGHT

省略形: 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: NOCURRENCY

省略形: 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
- 英大文字 A B C D E G N P R S V X Z またはその英小文字
- スペース
- 特殊文字 * + - / , . ; () " =
- 形象定数
- ヌル終了リテラル
- DBCS リテラル
- 国別リテラル

プログラムが 1 つの通貨タイプしか処理しない場合には、CURRENCY SIGN 節の代わりに CURRENCY オプションを使用して、プログラムの PICTURE 節で使用する通貨記号を指定できます。プログラムで複数の通貨タイプを処理する場合は、CURRENCY SIGN 節と WITH PICTURE SYMBOL 句を併用して、異なる通貨記号タイプを指定しなければなりません。

CURRENCY オプションと CURRENCY SIGN 節の両方をプログラムで使用した場合は、CURRENCY オプションの方が無視されます。CURRENCY SIGN 節で指定した通貨記号を、PICTURE 節で使用することができます。

NOCURRENCY オプションが有効なときに、CURRENCY SIGN 節を省略すると、通貨記号の PICTURE 記号としてドル記号 (\$) が使用されます。

区切り文字: CURRENCY オプション・リテラルは、APOST|QUOTE コンパイラー・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。

DATA

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

▶ DATA($\begin{array}{|c|} \hline 31 \\ \hline 24 \end{array}$) ▶

デフォルト: DATA(31)

省略形: なし

再入可能プログラムの場合は、DATA コンパイラー・オプションおよび HEAP ランタイム・オプションによって、動的データ域のストレージ (WORKING-STORAGE や FD レコード域など) を 16MB 境界より下から獲得するか (DATA(24))、制限のないストレージから獲得するか (DATA(31)) が制御されます。(DATA は LOCAL-STORAGE データの位置に影響を与えません。代わりに、STACK ランタイム・オプションが、プログラムの AMODE とともに、その位置を制御します。)

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

- DATA(24) が有効で、ALLOCATE ステートメントの LOC 31 句が指定されていない場合、ALLOCATE は 16 MB 境界より下からストレージを獲得します。
- DATA(31) が有効で、ALLOCATE ステートメントの LOC 24 句が指定されていない場合、ALLOCATE は 16 MB 境界より上からストレージを獲得しようとします。

関連概念

43 ページの『ストレージとそのアドレス可能度』

関連タスク

570 ページの『プログラムを再入可能にする』

言語環境プログラム 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

204 ページの『QSAM ファイル用のバッファの割り振り』

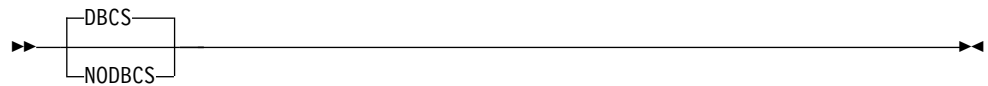
409 ページの『RENT』

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、コンパイラーは、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: DBCS

省略形: なし

DBCS が有効であると、リテラルはカテゴリー英数字のままで、リテラルの 2 バイト部分が構文検査されます。

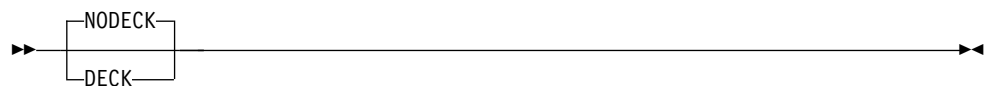
関連参照

349 ページの『矛盾するコンパイラー・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: NODECK

省略形: DINOD

関連タスク

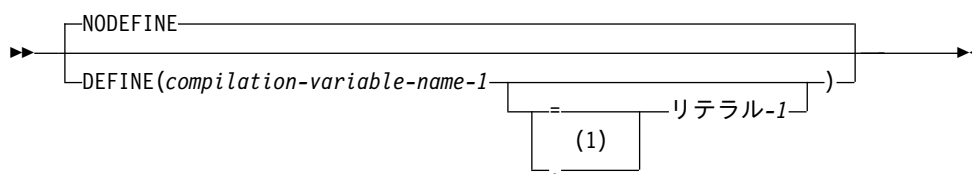
305 ページの『オブジェクト・コードの作成 (SYSLIN または SYSPUNCH)』

DEFINE

DEFINE コンパイラー・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ー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 コンパイラー・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 コンパイラーを呼び出す場合、「=」は使用できません。

デフォルト: NODEFINE

省略形: 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 コンパイラー・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。詳しくは、「Enterprise COBOL for z/OS 言語解説書」にある「ユーザー定義語」を参照してください。

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。 国別リテラル、DBCS リテラル、およびヌル終了英数字リテラル (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。

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

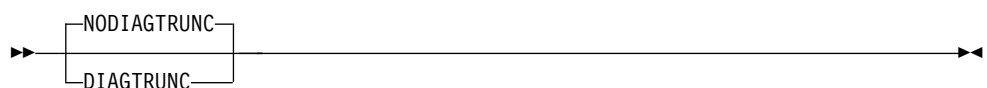
条件付きコンパイル (「Enterprise COBOL for z/OS 言語解説書」)

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: 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せん。

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

54 ページの『数値データの形式』

126 ページの『参照修飾子』

関連参照

430 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: 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。オーバーパ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、またはオーバーパ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 46. 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: 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。

関連タスク

555 ページの『動的呼び出しの作成』

関連参照

349 ページの『矛盾するコンパイラー・オプション』

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

DUMP

DUMP は、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・ダンプを作成する場合に使用します。

DUMP オプションの構文



デフォルト: NODUMP

省略形: 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 IGYppnnnnn で出されます。一般に、メッセージ IGYppnnnnn 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、IGYpp5nnnn メッセージと IGYpp1nnnn メッセージはどちらも、ユーザー異常終了 1nnnn 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 5nnnn と 1nnnn 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) を使用してコンパイルした場合は、行番号が正しく示されないことがあります。エラーによっては、プログラムの最後の行が示される場合があります。)
- 汎用レジスターの内容。

DUMP および OPTIMIZE(1|2) コンパイラー・オプションと一緒に使用すると、コンパイラーは、次の最適化プログラム・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

言語環境プログラム デバッグのガイド (異常終了コードの理解)

関連参照

349 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: NODYNAM

省略形: 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。

- CICS 変換プログラムまたは CICS コンパイラー・オプションによって処理される COBOL プログラム
- 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

561 ページの『静的呼び出しと動的呼び出しの両方の作成』

527 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

349 ページの『矛盾するコンパイラー・オプション』

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

EXIT

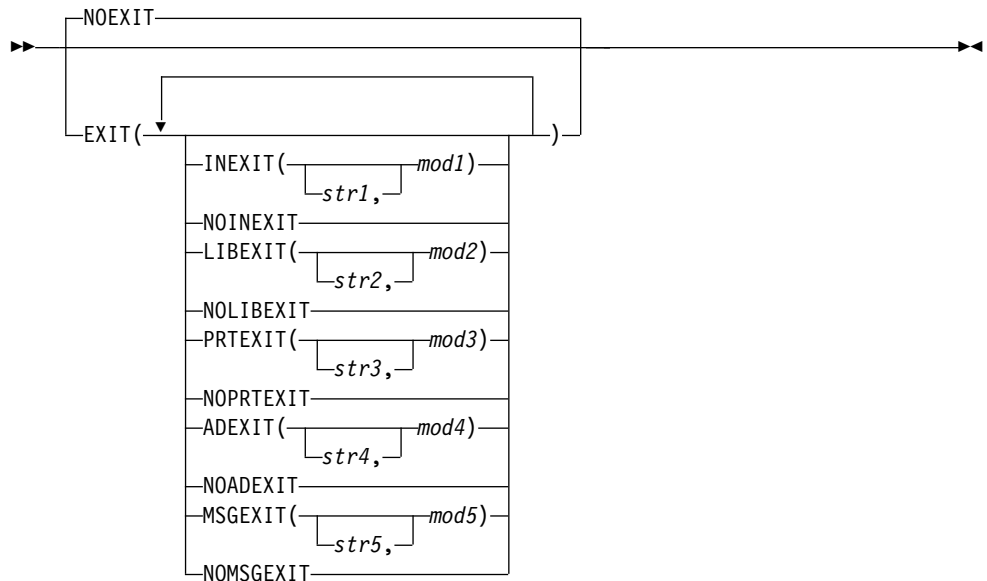
EXIT オプションを使用して、各種コンパイラー関数に代わるユーザー提供モジュールを指定します。

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。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: NOEXIT

省略形: 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。

PRTEXTIT(['str3',]mod3)

コンパイラーは、プリンター宛先の出力を、SYSPRINT ではなく、ユーザー提供のプログラム・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。ストリングは最高 64 文字までの長さにすることができ、2 つのアポストロフィ (' ') で囲む必要があります。任意の文字をストリングに使用できますが、アポストロフィを組み込む場合は二重 (") にしなければなりません。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*

855 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。

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

349 ページの『矛盾するコンパイラー・オプション』

379 ページの『FLAGSTD』

841 ページの『付録 E. 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: NOEXPORTALL

省略形: 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

349 ページの『矛盾するコンパイラー・オプション』

FASTSRT

FASTSRT 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。

FASTSRT オプションの構文



デフォルト: NOFASTSRT

省略形: FSRT|NOFSRT

関連タスク

261 ページの『FASTSRT 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。



デフォルト: FLAG(I,I)

省略形: FINOF

x および y は、I、W、E、S、U のいずれかになります。

FLAG(x,y) を使用すると、重大度レベル x 以上のエラーに関して診断メッセージをソース・リストの終わりに作成し、重大度レベル y 以上のエラーに関してはエラー・メッセージをソース・リストに直接組み込むことができます。 y に指定する重大度は、 x に指定する重大度より低い値にすることができません。FLAG(x,y) を使用する場合は、SOURCE コンパイラー・オプションも指定する必要があります。

ソース・リスト内のエラー・メッセージは、メッセージ・コードを指す矢印の中にステートメント番号を埋め込むことによって強調されます。メッセージ・コードの後にメッセージ・テキストが続きます。以下に例を示します。

```
000413      MOVE CORR WS-DATE TO HEADER-DATE
==000413==>  IGYPS2121-S      " WS-DATE " was not defined as a data-name. . . .
```

FLAG(x,y) が有効である場合は、重大度 y 以上のメッセージが、リスト内でそのメッセージの原因となった行の後に組み込まれます。(例外のメッセージについては、以下に示す関連参照資料を参照してください。)

エラーのフラグ付けを抑止する場合は、NOFLAG を使用してください。NOFLAG を使用しても、コンパイラー・オプションのエラー・メッセージは抑止されません。

組み込みメッセージ

- レベル U メッセージを組み込みに指定するのはお勧めできません。レベル U のメッセージの組み込みの指定は受け入れられますが、ソース内には何のメッセージも作成されません。
- FLAG オプションは、コンパイラー・オプションの処理前に作成された診断メッセージには影響しません。
- コンパイラー・オプション、CBL ステートメントまたは PROCESS ステートメント、または BASIS、COPY、および REPLACE の各ステートメントの処理中に生成された診断メッセージがソース・リストに組み込まれることはありません。このようなメッセージはすべて、コンパイラー出力の先頭に表示されます。
- *CONTROL または *CBL ステートメントの処理中に作成されたメッセージは、ソース・リストに組み込まれません。

関連参照

317 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』

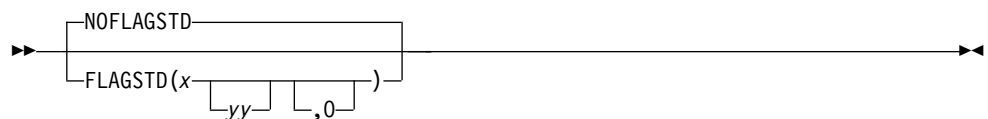
FLAGSTD

FLAGSTD は、準拠していると見なされる 85 COBOL 標準のレベルまたはサブセットを指定し、プログラムに組み込まれている 85 COBOL 標準エレメントに関して通知メッセージを取得する場合に使用します。

フラグ付け処理には、次の項目のどれかを指定することができます。

- 連邦情報処理標準 (FIPS) COBOL の選択されたサブセット
- オプション・モジュールのいずれか
- 廃止される言語エレメント
- 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、「非規格準拠外」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: 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 が含められます (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

852 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

349 ページの『矛盾するコンパイラー・オプション』

850 ページの『MSGEXIT の処理』

HGPR

HGPR オプションは、z/Architecture プロセッサに備わっている 64 ビット・レジスターをコンパイラーが使用する方法を制御します。

HGPR オプションの構文

```
➡➡ HGPR( PRESERVE  
NOPRESERVE ) ➡➡
```

デフォルト: HGPR(PRESERVE)

省略形: なし

Enterprise COBOL コンパイラーは、64 ビット幅の z/Architecture 汎用レジスター (GPR) を使用します。HGPR は「High-halves of 64-bit GPRs (64 ビット GPR の上位半分)」の略語であり、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、プログラムが使用する 64 ビット GPR の高位半分の、関数のプロローグで保存し、エピローグで復元すること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++ のどのコンパイラー生成コード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、プログラムが使用する 64 ビット GPR の高位半分の保存するため、パフォーマンスが向上します。

- 初期化されていないデータ項目が BY REFERENCE で渡される場合、警告メッセージは発行されません。ただし、INITCHECK 分析では、BY CONTENT および BY VALUE で渡される初期化されていないデータ項目に対しては警告が発せられます。
- 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。

注:

- INITCHECK 分析はいずれもコンパイル時にのみ行われます。
- INITCHECK オプションは、プログラムのコンパイル後はプログラムの動作やパフォーマンスに影響しません。
- INITCHECK オプションを使用すると、コンパイル時間が長引いたりメモリ消費量が増えたりすることがあります。

INITIAL

INITIAL コンパイラー・オプションによって、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。

INITIAL オプションの構文



デフォルト: 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

349 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・プログラムで PERFORM ステートメントによって参照されるプロシージャー (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: INLINE

省略形: INL | NOINL

INLINE

OPTIMIZE(1) または OPTIMIZE(2) が有効であるときに、INLINE 節が指定されると、コンパイラーは、ソース・プログラムにおいて 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を許可します。特定の PERFORM ブロックでプロシージャーをインライン化するかどうかはコンパイラーが判断します。これをオーバーライドするには、>>INLINE OFF ディレクティブを使用します。

NOINLINE

NOINLINE が指定されている場合は、有効になっている最適化レベル設定に関係なく、PERFORM ステートメントによって参照されるプロシージャーはインライン化¹されません。これは、>>INLINE ON ディレクティブを使用してもオーバーライドできません。

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ー (段落またはセクション) の PERFORM をそのプロシージャー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。

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プログラム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: INTDATE(ANSI)

省略形: なし

INTDATE(LILIAN) を使用すると、日付組み込み関数は、言語環境プログラムの日付呼び出し可能サービスと互換性のある結果を戻します。

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

67 ページの『データ呼び出し可能サービスの使用』

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(*name*)——▶▶

デフォルト: LANGUAGE(ENGLISH)

省略形: LANG(EN|UE|JA|JP)

name は、コンパイラ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 47. LANGUAGE コンパイラ・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・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・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 プログラム・ランタイム・オプション NATLANG も設定する必要があります。コンパイル JCL では CEEOPTS DD を使用することをお勧めします。

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・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・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: 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

▶▶
┌ NOLIST
└ LIST
————▶▶

デフォルト: 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
- 静的マップ
- 自動マップ

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、リストには含められません。

関連タスク

464 ページの『リストの入手』

関連参照

349 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*)

MAP

DATA DIVISION 項目および暗黙的に宣言された全項目のリストを作成するには、**MAP** オプションを使用します。リストの **MAP** 出力に 16 進または 10 進のどちらのオフセットを表示するかを指定することもできます。

MAP オプションの構文



デフォルト: **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

MAP 出力を制限したい場合は、DATA DIVISION で *CONTROL MAP または NOMAP ステートメントを使用してください。*CONTROL NOMAP の後のソース・ステートメントは、*CONTROL MAP ステートメントによって出力が通常の MAP 形式に戻されない限り、リストには含められません。以下に例を示します。

*CONTROL NOMAP	*CBL NOMAP
01 A	01 A
02 B	02 B
*CONTROL MAP	*CBL MAP

MAP(HEX|DEC) オプションが有効になっている場合は、ソース・コード・リストに MAP 報告書も組み込まれます。圧縮 MAP 情報は、DATA DIVISION の WORKING-STORAGE SECTION、FILE SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION のデータ名定義の右側に示されます。XREF データと組み込み MAP 要約の両方が同じ行にある場合は、組み込み MAP 要約が先にリストされます。

469 ページの『例: MAP 出力』

関連概念

451 ページの『第 19 章 デバッグ』

関連タスク

464 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: MAXPCF(100000)

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。

- CICS、SQL または SQLIMS オプションから生成されたステートメント、および COPY や REPLACE ステートメントの拡張を含む、PROCEDURE DIVISION 内の COBOL ステートメント数
- 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
- DATA DIVISION 内の可変長グループまたはサブグループのサイズを実行時に計算する操作

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測るには、IBM Developer for z Systems で提供される Metrics のような機能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は、コンパイラー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、コンパイラーに必要なメモリー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 がゼロでない場合に、プログラム複雑度係数が n を超えると、OPTIMIZE(1) または OPTIMIZE(2) の指定はすべて 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) に上げることにより、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、コンパイラーがコンパイルするのに過度の時間がかかったり、メモリー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

402 ページの『OPTIMIZE』

MDECK

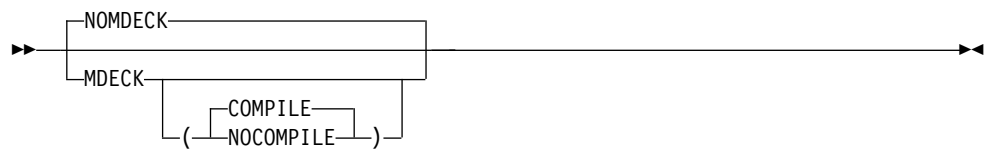
MDECK コンパイラー・オプションは、ライブラリー処理 (すなわち、COPY、BASIS、REPLACE、EXEC SQL INCLUDE、および EXEC SQLIMS INCLUDE ステートメントの結果) 後の更新済み入力ソースのコピーがファイルに書き込まれることを指定します。

Enterprise COBOL が z/OS UNIX 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。TSO またはバッチで稼働している Enterprise COBOL の場合、MDECK 出力は、SYSMDECK DD 割り振りで定義されたデータ・セットに書き込まれます。この割り振りでは、RECFM F または FB で、LRECL が 80 バイトの MVS データ・セットを指定している必要があります。

注: z/OS TSO またはバッチによってコンパイルする場合、MDECK または NOMDECK オプションを以下のように指定していても、COBOL コンパイラーはすべてのコンパイルに SYSMDECK データ・セット割り振りを必要とします。

- MDECK オプションを指定する場合、SYSMDECK DD 割り振りでは永続データ・セットを指定する必要があります。
- NOMDECK オプションを指定する場合、SYSMDECK DD 割り振りでは一時ユーティリティー・データ・セットまたは永続データ・セットを指定できます。

MDECK オプションの構文



デフォルト: NOMDECK

省略形: NOMD | MD | MD(C | NOC)

オプション指定:

PROCESS (または CBL) ステートメントで MDECK オプションを指定することはできません。指定できるのは、以下のいずれかの方法に限られます。

- 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
- cob2 コマンド・オプションとして
- 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:

- MDECK(COMPILE) が有効である場合、ライブラリー処理および MDECK 出力ファイルの生成が完了した後、正常にコンパイルが継続されますが、その際、コンパイルは COMPILE|NOCOMPILE、DECK|NODECK、および OBJECT|NOOBJECT コンパイラー・オプションの設定値に従って行われます。
- 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

パイラーは、COMPILE、DECK、および OBJECT コンパイラー・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 オプションまたは SQLIMS オプションを使用してコンパイルを行うと、対応する EXEC SQL INCLUDE ステートメントまたは EXEC SQLIMS INCLUDE ステートメントが MDECK 出力において拡張されます。

CBL、PROCESS、*CONTROL、および *CBL カード・イメージは、MDECK 出力ファイルの適切な位置に渡されます。

バッチ・コンパイル (単一入力ファイル内に複数の COBOL ソース・プログラムが含まれている) の場合、完全な拡張ソースを含んでいる単一 MDECK 出力ファイルが作成されます。

SEQUENCE コンパイラー・オプション処理はすべて MDECK ファイル内に反映されます。

COPY ステートメントは、コメントとして MDECK ファイルに組み込まれます。

関連タスク

298 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』

307 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

349 ページの『矛盾するコンパイラー・オプション』

445 ページの『第 18 章 コンパイラー指示ステートメント』

NAME

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。

NAME オプションの構文



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 節で使用する N 記号の解釈を制御し、国別処理や DBCS 処理が必要かどうかを指示します。

NSYMBOL オプションの構文



デフォルト: NSYMBOL(NATIONAL)

省略形: NS(NAT|DBCS)

NSYMBOL(NATIONAL) を指定した場合:

- USAGE 節のない、記号 N のみからなる PICTURE 節で定義されたデータ項目は、USAGE NATIONAL 節が指定されている場合のように扱われます。
- N". . ." または N'. . .' の形式のリテラルは、国別リテラルとして扱われます。

NSYMBOL(DBCS) を指定した場合:

- USAGE 節のない、記号 N のみからなる PICTURE 節で定義されたデータ項目は、USAGE DISPLAY-1 節が指定されている場合のように扱われます。
- N". . ." または N'. . .' の形式のリテラルは、DBCS リテラルとして扱われます。

NSYMBOL(DBCS) オプションは、前リリースの IBM COBOL との互換性を提供します。 NSYMBOL(NATIONAL) オプションは、上記の言語エレメントの処理を、この点に関して 2002 COBOL 標準に準拠させます。

NSYMBOL(NATIONAL) は、Unicode データや Java とのインターオペラビリティのためのオブジェクト指向構文を使用するアプリケーションの場合の推奨オプションです。

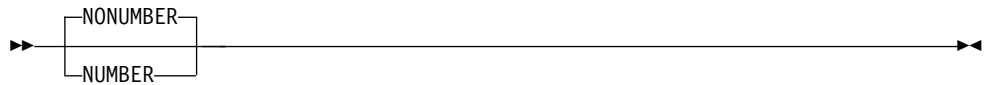
関連参照

349 ページの『矛盾するコンパイラ・オプション』

NUMBER

NUMBER コンパイラ・オプションは、ソース・コードの中に行番号があり、それらの番号がエラー・メッセージと SOURCE、MAP、LIST、および XREF のリストで必要な場合に使用してください。

NUMBER オプションの構文



デフォルト: NONUMBER

省略形: NUM|NONUM

NUMBER を要求すると、コンパイラは、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おり、NUMBER 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。

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、NONUMBER を使用してください。NONUMBER 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。

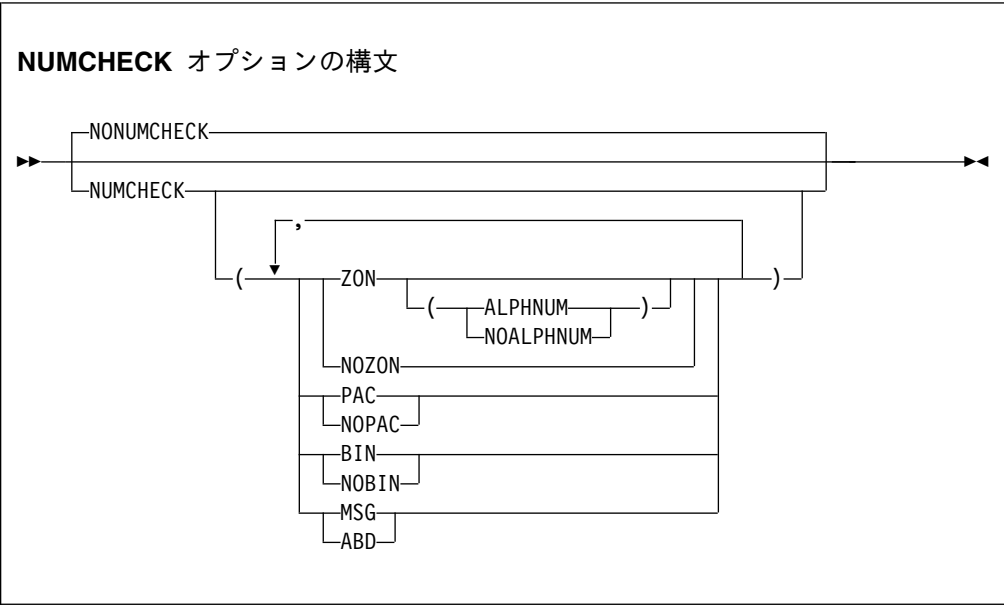
NUMCHECK

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ている場合にデータ項目を検証するために追加の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) データ項目およびパック 10 進数 (COMP-3) データ項目の場合、コンパイラーは送信フィールドごとに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、データ項目の桁が、PICTURE 節で許可される桁を超えているかどうかを確認するために SIZE ERROR 検査を生成します。

APAR PH08642 対応 PTF がインストールされ、冗長性検査が取り除かれることによってパフォーマンスが向上するよう、NUMCHECK オプションが変更されました。この APAR が適用された後、表示されるランタイム・メッセージは以前よりも少なくなります。

冗長性検査を取り除くように行われた分析は、OPT(0) の場合よりも、OPT(1|2) で複雑になります。OPT(0) では、より単純な形式の分析が行われるため、可能な限りコンパイル時間が短くなります。より高い OPT レベルで、メッセージはより少なくなります。

コンパイル時に、検査によって常に無効データが見つかるものとコンパイラーが判断できる場合、コンパイル時メッセージが生成され、その検査は取り除かれます。(以下の MSG|ABD を参照してください。)



デフォルト: 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)、PAC、BIN、および MSG です。例えば、NUMCHECK は NUMCHECK(ZON(ALPHNUM),PAC,BIN,MSG) と同じ効果があります。
- データベース・サブオプションが指定されない場合のデフォルト・データ型サブオプションは ZON(ALPHNUM)、PAC、および BIN です。例えば、NUMCHECK(ABD) は NUMCHECK(ZON(ALPHNUM),PAC,BIN,ABD) と同じ効果があります。
- データ型サブオプションが 1 つのみ指定されている場合のデフォルトは NOZON、NOPAC、NOBIN、および MSG です。例えば、NUMCHECK(BIN) は NUMCHECK(NOZON,NOPAC,BIN,MSG) と同じ効果があります。
- NO が付いたデータ型サブオプションがすべて指定された場合、リストには NONUMCHECK が示されます。例えば、NUMCHECK(NOZON,NOPAC,NOBIN) は NONUMCHECK と同じ効果があります。

省略形: NONC | NC

ZON[(ALPHNUM|NOALPHNUM)] | NOZON

ZON がサブオプションなしで指定された場合、デフォルトは ZON(ALPHNUM) です。

ZON または ZON(ALPHNUM) 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側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) 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側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。ただし、それらの項目が英数字データ項目、英数字リテラル、または英数字形象定数との比較に使用される場合を除きます。

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 の場合は、警告メッセージ (NUMCHECK(ZON,MSG) の場合) または強制終了メッセージ (NUMCHECK(ZON,ABD) の場合) が出力されます。
- データが NUMERIC の場合、ステートメントの外部動作は、低速になる以外は NUMCHECK(NOZON) と同じです。

PAC | NOPAC

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) データ項目に関する暗黙

的な数値クラス・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 ステートメントに関しては、NUMCHECK(ZON) および NUMCHECK(PAC) によって、ゾーン 10 進数やパック 10 進数である BY CONTENT データ項目が検査されますが、BY REFERENCE パラメーターは検査されません。(ゾーン 10 進数データ項目もパック 10 進数データ項目も BY VALUE 句では指定できません。)

BIN | NOBIN

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 コードは生成されません。

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は、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記されたランタイム警告メッセージが発行されます。
- ABD が有効になっている場合は、強制終了メッセージが発行され、異常終了が引き起こされます。

コンパイル時に、検査によって常に無効データが見つかるものとコンパイラーが判断できる場合、コンパイル時メッセージが生成されます。

- MSG が有効であれば、メッセージは警告レベル・メッセージで、検査はランタイムに行われます。
- ABD が有効であれば、メッセージはエラー・レベル・メッセージで、検査は取り除かれます。

パフォーマンスの考慮: 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、NUMCHECK が指定されたときは、NONUMCHECK が指定されたときよりもはるかに時間がかかります。

APAR PH08642 対応 PTF がインストールされているため、NUMCHECK のパフォーマンスは向上しています。ただし、NONUMCHECK を指定したときのパフォーマンスが最高であり、より高い OPT レベルで向上します。

注: ZONECHECK は非推奨ですが、互換性のために使用できるようにはなっています。これは NUMCHECK(ZON(ALPHNUM)) で置き換えられます。

関連タスク

61 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

398 ページの『NUMPROC』

430 ページの『TRUNC』
438 ページの『ZONECHECK』
440 ページの『ZONEDATA』

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: 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。

パフォーマンスの考慮事項: NUMPROC(PFD) を使用すると、内部 10 進数データおよびゾーン 10 進数データの処理の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。PL/I または FORTRAN で書かれたプログラムを COBOL プログラムで呼び出す場合は、NUMPROC(NOPFD) を使用してください。

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

61 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

60 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: OBJECT

省略形: 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。

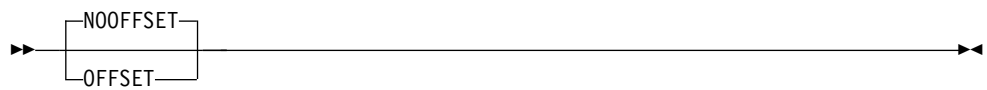
関連参照

349 ページの『矛盾するコンパイラー・オプション』

OFFSET

OFFSET は、PROCEDURE DIVISION の圧縮リストを作成する場合に使用します。

OFFSET オプションの構文



デフォルト: NOOFFSET

省略形: OFFIN00FF

OFFSET を使用した場合、PROCEDURE DIVISION の圧縮リストには、行番号およびステートメント参照が含まれるほか、ステートメントごとに生成された最初の命令の位置も含まれます。

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
- 静的マップ
- 自動マップ

注:

- 最適化プログラムは段落をインライン化するか、コードを周囲に移動させるか、またはあまり使用されないコード (エラー・メッセージ・フォーマット設定コードなど) であれば、そのコードを実際にはプログラム本文の後に置く場合があります。これで、以前のコンパイラーで作成されたものよりも役立たない、OFFSET レポートが作成される可能性があります。代わりに、LIST 出力を調べてください (OFFSET および LIST は相互に排他的なオプションであることに注意してください)。詳しくは、474 ページの『LIST 出力の読み取り』を参照してください。
- エラー・メッセージ・フォーマット設定において調和しないコードが原因で、
「From compile unit {name} at entry point {name} at compile unit offset {offset}...」に示されている 言語環境プログラム 生成のオフセットが、プログラムのオフセット範囲からはみ出してしまう可能性があります。このような場合、問題を見つけるためには、COBOL メッセージ (IGZnnnns) でステートメント番号を調べてください。

関連参照

349 ページの『矛盾するコンパイラー・オプション』

495 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 は、z/OS UNIX 環境で cob2 コマンドを使用してコンパイルした場合には、無視されます。(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

298 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』

303 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)』

307 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』

321 ページの『第 15 章 z/OS UNIX のもとでのコンパイル』

関連参照

349 ページの『矛盾するコンパイラー・オプション』

OPTIMIZE

OPTIMIZE は、オブジェクト・プログラムの実行時間を短縮するために使用します。最適化によって、オブジェクト・プログラムが使用するストレージの量を減らすこともできます。

OPTIMIZE オプションの構文

▶▶ OPTIMIZE-($\begin{array}{|c|} \hline 0 \\ \hline 1 \\ \hline 2 \\ \hline \end{array}$)————▶▶

デフォルト: OPTIMIZE(0)

省略形: 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 がある外部 10 進数データは、無効なゾーン・ビットを許可するために ZONEDATA(MIG|NOPFD) が使用される場合を除き、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) は、制限付きの最適化を指定し、コンパイル時間が最短になります。TEST オプションを指定すると、全デバッグ機能を使用可能になります。
- OPTIMIZE(1) は、アプリケーションのランタイム・パフォーマンスを向上させる最適化を指定します。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) では、共通する副次式の除去や値の伝搬など、いくつかのブロック内最適化が行われます。TEST オプションを指定した場合は、大半のデバッグ機能を使用できます。
- OPTIMIZE(2) は、より高レベルの最適化を指定します。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 オプションを指定した場合は、一部のデバッグ機能を使用できます。

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、言語環境プログラム サービス CEEHDLR によって登録されたユーザー作成条件ハンドラーには注意が必要です。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に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 節を使用して定義し、ハンドラーがデータ項目の最新値を使用するようにする必要があります。もしくは、条件ハンドラー・プログラムを TEST コンパイラー・オプションを使用してコンパイルすることができます。VOLATILE 節の使用は、TEST オプションの使用に優先します。TEST オプションではプログラム全体の最適化が低下する可能性があります。VOLATILE では

最適化の低下が局所的になるためです。 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。

| OPTIMIZE(1) または OPTIMIZE(2) が有効になっていて INLINE が指定されている場
| 合、コンパイラーは PERFORM ステートメントのプロシーチャーのインライン化
| を検討します。詳細は、 384 ページの『INLINE』を参照してください。

注: Enterprise COBOL V5 以降では、NOOPTIMIZE、OPTIMIZE、OPTIMIZE(STD)、お
よび OPTIMIZE(FULL) オプションは削除されましたが、互換性のために許容されま
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 48. 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(1)
OPTIMIZE(STD)	OPTIMIZE(1)
OPTIMIZE(FULL)	OPTIMIZE(1) および STGOPT

関連概念

786 ページの『最適化』

関連タスク

283 ページの『エラー処理用のルーチンの作成』

関連参照

349 ページの『矛盾するコンパイラー・オプション』

| 384 ページの『INLINE』

389 ページの『MAXPCF』

423 ページの『TEST』

421 ページの『STGOPT』

VOLATILE 文節 (Enterprise COBOL for z/OS 言語解説書)

OUTDD

OUTDD は、システム論理出力装置に向けられる DISPLAY 出力を特定の DD 名に送
りたいことを指定する場合に使用します。

OUTDD で指名する DD 名で、z/OS UNIX ファイル・システム内のファイルを指定
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: OUTDD(SYSOUT)

省略形: OUT

OUTDD コンパイラー・オプションと言語環境プログラム MSGFILE ランタイム・オプションで同じ DD 名 (どちらもデフォルトでは SYSOUT) が指定された場合は、言語環境プログラム・メッセージ機能を使用してシステム論理出力装置に DISPLAY 出力が書き込まれます。

制約事項: OUTDD オプションは、CICS では無効です。

関連タスク

40 ページの『システム論理出力装置上でのデータの表示』

501 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

言語環境プログラム プログラミング・リファレンス (MSGFILE)

PARMCHECK

PARMCHECK オプションは、WORKING-STORAGE における最後の項目の後に追加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。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。

呼び出し側プログラムが PARMCHECK でコンパイルされている場合、コンパイラーは、WORKING-STORAGE セクションにおいて最後にある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 V4 以前のコンパイラーから COBOL V6 以降のコンパイラーに移行するときに役立つ可能性があります。また、このオプションは、不正な COBOL データをクリーンアップしたり適正なプログラミング基準を検査したりするためにも使用できます。

PARMCHECK オプションの構文

▶—NOPARMCHECK————▶
▶—PARMCHECK——(——MSG——ABD——,n——)————▶

デフォルト: NOPARMCHECK

省略形: NOPC | PC

サブオプションのデフォルトは以下のとおりです。

- サブオプションが指定されていない場合のデフォルトは PARMCHECK(MSG,100) です。
- MSG または ABD のみが指定されている場合のデフォルトは PARMCHECK(MSG|ABD,100) です。例えば、PC(ABD)=PC(ABD,100) のようになります。
- n のみが指定されている場合のデフォルトは PARMCHECK(MSG,n) です。例えば、PC(5000)=PC(MSG,5000) のようになります。

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が記されたランタイム警告メッセージが発行されます。
- 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、そのメッセージは異常終了を引き起こす強制終了レベルのメッセージです。

n

WORKING-STORAGE における最後の項目の後に追加されるバッファのサイズ (バイト)。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 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。良好なパフォーマンスを得るには、NOPARMCHECK を指定する必要があります。

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、プログラム名および入り口点名の処理を制御します。

PGMNAME オプションの構文

▶▶ PGMNAME ([COMPAT
LONGMIXED
LONGUPPER]) ▶▶

デフォルト: PGMNAME (COMPAT)

省略形: PGMN(LM|LU|CO)

LONGUPPER は、UPPER、LU、または U と省略することができ、LONGMIXED は、MIXED、LM、または M と省略することができます。

PGMNAME オプションは、以下のコンテキストで使用される名前の処理を制御します。

- PROGRAM-ID 段落で定義されたプログラム名
- ENTRY ステートメントのプログラム入り口点名
- 以下におけるプログラム名参照:
 - ネストされたプログラムを参照する 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
 - 静的にリンクされたプログラム、または DLL を参照する SET プロシーチャー・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

PGMNAME(COMPAT)

PGMNAME(COMPAT) を使用すると、次のように、プログラムは、COBOL コンパイラーの旧バージョン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。

外部プログラム名はコンパイラー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) を使用する場合、PROGRAM-ID 段落で COBOL ユーザー定義語として指定されるプログラム名は、次のようなユーザー定義語に関する通常の COBOL 規則に従っていなければなりません。

- プログラム名の長さは最大 30 文字です。
- 名前で使用される文字はすべて英字、数字、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。

外部プログラム名はコンパイラー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ー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、切り捨ても変換も行われません。

PGMNAME(LONGMIXED)

PGMNAME(LONGMIXED) を使用する場合、プログラム名は、切り捨てられたり、変換されたり、大文字への変換をされることなく、現状のまま処理されます。

PGMNAME(LONGMIXED) を使用する場合、すべてのプログラム名定義は、プログラム名のリテラル形式を使用して、PROGRAM-ID 段落または ENTRY ステートメントで指定する必要があります。プログラム名に使用されるリテラルには、X'41' から X'FE' の範囲の文字を含めることができます。

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 の中の割り当て名、および COPY ステートメントの中の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。
- リンク・エディットの考慮事項: PGMNAME(LONGUPPER) または PGMNAME(LONGMIXED) オプションを使用してコンパイルされた 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ある場合を除き、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 プログラムへの動的呼び出しは許可されません。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) によってサポートされる拡張文字セットを使用する際には、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) またはシステム規約のうち該当するものに適合する名前を使用してください。

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*)

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( COMPAT  
            EXTEND )◄◄
```

デフォルト: 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           *> 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 C of B      *> Refers to 03 level C (unique by COBOL standard rules)

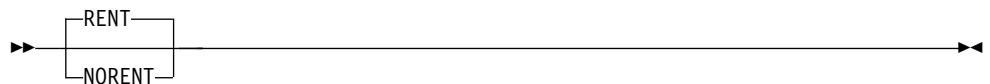
```

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。NORENT としてコンパイルされたプログラムは再入不可オブジェクト・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: 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性に影響を与える他のコンパイラー・オプションと相互作用します。再入可能プログラムで、動的データ域を制限のないストリングで割り振るか 16 MB より下から取得したストレージで割り振るかを制御するには、DATA(24|31) オプションを使用します。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。

DATA オプションの設定は、NORENT を指定してコンパイルされたプログラムに影響を与えません。

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。

プログラム・オブジェクト内のいずれかのプログラムが再入可能でない場合は、プログラム・オブジェクトを RENT または REUS のリンク・エディット属性でリンク・エディットしてはいけません。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。

関連概念

43 ページの『ストレージとそのアドレス可能度』

関連タスク

570 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 (再入可能コードの使用)

関連参照

349 ページの『矛盾するコンパイラー・オプション』

366 ページの『DATA』

『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

```
➡➡ RMODE ( 

|      |
|------|
| AUTO |
| 24   |
| ANY  |

 ) ————— ➡➡
```

デフォルト: 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ーと互換性があります。これらのコンパイラーは、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。

RMODE(24) オプションでコンパイルされたプログラムは、NORENT または RENT のいずれが指定されたかに関係なく、RMODE 24 になります。

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。このプログラムは RMODE ANY 属性を持ちます。

NORENT オプションを指定した場合は、RMODE(24) または RMODE(AUTO) のいずれかのコンパイラー・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性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性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、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・コードに属性 RMODE ANY がある場合には、RMODE ANY または RMODE 24 でオブジェクト・コードをリンク・エディットすることができます。

関連概念

43 ページの『ストレージとそのアドレス可能性』

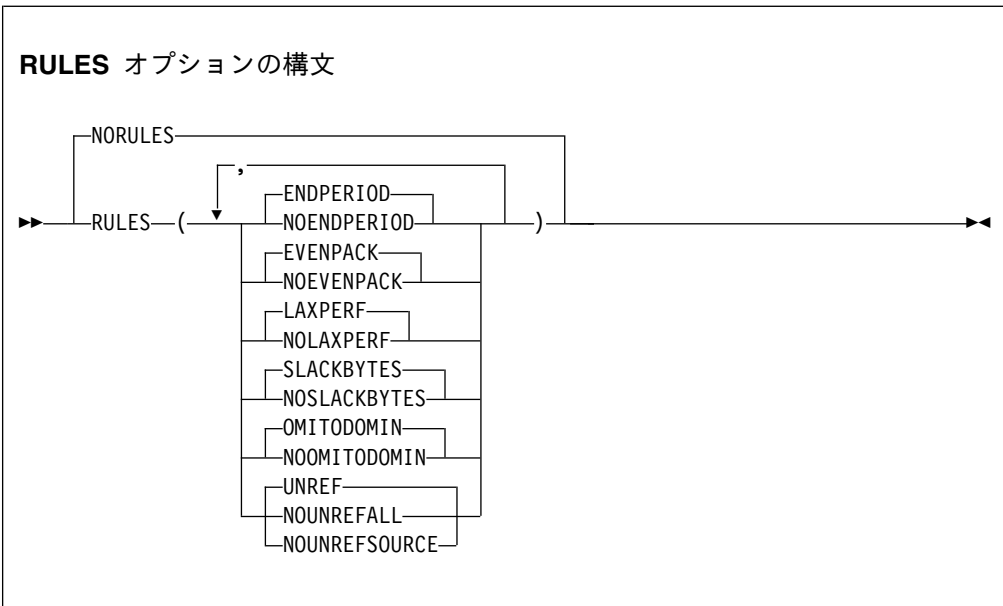
関連参照

204 ページの『QSAM ファイル用のバッファの割り振り』

349 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: NORULES

省略形:

- ENDP = ENDPERIOD
- EVENP = EVENPACK
- LXPRF = LAXPERF
- SLCKB = SLACKBYTES
- OOM = OMITODOMIN
- NOUNRA = NOUNREFALL
- NOUNRS = NOUNREFSOURCE

RULES に次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有効範囲が明示範囲終了符号 END-* ではなくピリオドで終了している場合に、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 が指定されると、コンパイラーは、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を発行します (そのデータ項目が持つ未使用ビットがゼロ以外の場合に、プログラムの動作が予期しないものになる可能性があるため)。

注:

- RULES(NOEVERPACK) 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ために役立ちます。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 CICS および Db2 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、効果のない COBOL 機能の使用に対して、コンパイラーから警告メッセージが出されます。このような機能には、算術ステートメントでの USAGE DISPLAY 数値データ項目、MOVE ステートメントでの大量スペース埋め込み、効果のないコンパイラー・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、コンパイラーが遊びバイト (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) を追加することになるすべての 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、コンパイラ

ーは、integer-1 (最小出現回数) なしで指定されたすべての OCCURS
DEPENDING ON 節に対して警告メッセージを出します。

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報告書
作成を行わないことを意味します。

NOUNREFALL が指定されると、FILE SECTION、WORKING-STORAGE SECTION、
LOCAL-STORAGE SECTION、および LINKAGE SECTION 内の参照されていないす
べての level-01 および level-77 のデータ項目が (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) すべて報告されます。これは、データ項
目の定義が、ユーザー・ソース・プログラムに直接現れるか、COPY メン
バーからプログラムに組み込まれたかにかかわらず行われます。

NOUNREFSOURCE が指定されると、FILE SECTION、WORKING-STORAGE
SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION 内の参照され
ていないすべての level-01 および level-77 のデータ項目が (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) すべて報告されます。これ
は、データ項目の定義が、ユーザー・ソース・プログラムに直接現れる場合
にのみ行われます。

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 CICS および Db2 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。

RULES オプションをサブオプションなしで指定した場合、デフォルトは
RULES(ENDPERIOD,EVENPACK,LAXPERF,SLACKBYTES,OMITODOMIN,UNREF) です。

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) と指定した場合は、RULES
(NOENDP,EVENP,LXPRF,NOSLCKB,OMITODOMIN,UNREF) が有効になります。
- オプションで、EXIT コンパイラー・オプションの MSGEXIT サブオプションとと
もに RULES オプションを使用して、ローカルのコーディング標準を強制するこ
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ために、
明示的な範囲区切り文字の代わりにピリオドを使用することがないようにしたい
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 RULES メッセージやその他の RULES
メッセージの重大度の変更方法に関する例については、IGYMSGXT という名前
の、SIGYSAMP にあるサンプル 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)

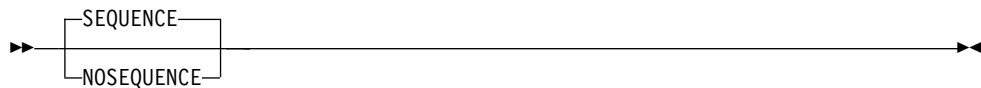
421 ページの『STGOPT』

SEQUENCE

SEQUENCE を使用すると、コンパイラーは桁 1 から 6 を調べ、ソース・ステートメントが 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、コンパイラーは診断メッセージを出します。

桁 1 から 6 がブランクのソース・ステートメントはこのシーケンス検査には関与しないため、メッセージは出されません。

SEQUENCE オプションの構文



デフォルト: SEQUENCE

省略形: 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。

NUMBER と SEQUENCE を使用すると、シーケンス検査は、EBCDIC 照合シーケンスではなく、数字に従って行われます。

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

458 ページの『行シーケンス問題の検出』

SERVICE

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: NOSERVICE

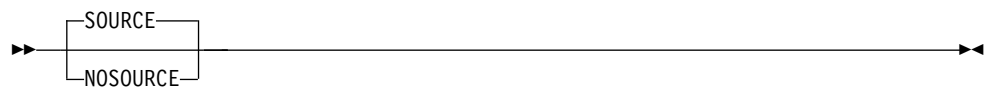
省略形: SERV|NOSERV

service string の長さは 64 文字に制限されています。

SOURCE

SOURCE は、ソース・プログラムのリストを入手する場合に使用します。このリストには、PROCESS または COPY ステートメントによって組み込まれたすべてのステートメントが入ります。

SOURCE オプションの構文



デフォルト: SOURCE

省略形: SINOS

ソース・リストに組み込みメッセージが必要な場合は、SOURCE を必ず指定します。

コンパイラ出力リストにソース・コードを出したくない場合は、NOSOURCE を使用してください。

SOURCE 出力を制限したい場合は、PROCEDURE DIVISION で *CONTROL SOURCE または NOSOURCE ステートメントを使用してください。Source *CONTROL NOSOURCE ステー

トメントの後のソース・ステートメントは、後続の *CONTROL SOURCE ステートメントによって出力が通常の SOURCE 形式に戻されない限り、リストには含められません。

469 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

▶▶SPACE($\left[\begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right]$)————▶▶

デフォルト: SPACE(1)

省略形: なし

SPACE が意味を持つのは、SOURCE コンパイラー・オプションが有効な場合だけです。

関連参照

415 ページの『SOURCE』

SQL

SQL コンパイラー・オプションを使用すると、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できるようになります。COBOL ソース・プログラムに SQL ステートメント(EXEC SQL ステートメント) が含まれていて、プログラムが Db2 プリコンパイラーによって処理されていない場合は、SQL オプションを指定する必要があります。

SQL オプションの構文

▶▶ $\left[\begin{array}{l} \text{NOSQL} \\ \text{SQL} \end{array} \right]$ ————▶▶
 $\left[\text{("DB2-suboption-string")} \right]$

デフォルト: NOSQL

省略形: なし

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 DD 名 DBRMLIB に書き込みます。Db2 がコンパイルするマシンで使用可能になっている必要があります

NOSQL オプションを指定した場合は、ソース・プログラム内で検出された SQL ステートメントは診断され、破棄されます。

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。

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・ストリングに分割して、複数の CBL ステートメントに置くことができます。以下に例を示します。

```
//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、連結される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。

関連概念

515 ページの『Db2 コプロセッサ』

522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

520 ページの『SQL オプションを使用したコンパイル』

521 ページの『Db2 サブオプションの分離』

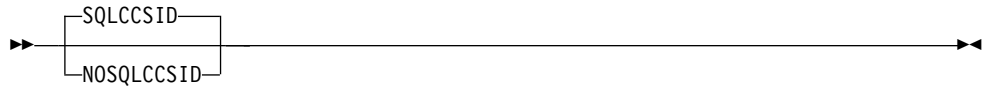
関連参照

349 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: SQLCCSID

省略形: SQLC|NOSQLC

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

515 ページの『Db2 コプロセッサ』

522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

523 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

523 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』

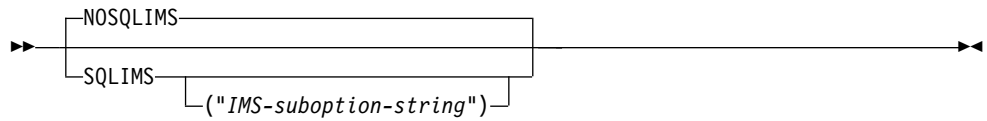
359 ページの『CODEPAGE』

416 ページの『SQL』

SQLIMS

SQLIMS コンパイラー・オプションは、情報管理システム (IMS) SQL コプロセッサを使用可能にして IMS サブオプションを指定する場合に使用します。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 オプションを指定した場合は、ソース・プログラム内で検出された SQLIMS 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、IMS サブオプションのストリングを区切ります。

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・ストリングに分割して、複数の CBL ステートメントに置くことができます。以下に例を示します。

```
//STEP1 EXEC IGYWC, . . .
// PARM.COBOL='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、コンパイラー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、連結される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。コンパイラーは、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

529 ページの『IMS SQL コプロセッサ』

関連タスク

532 ページの『SQLIMS オプションを使用したコンパイル』

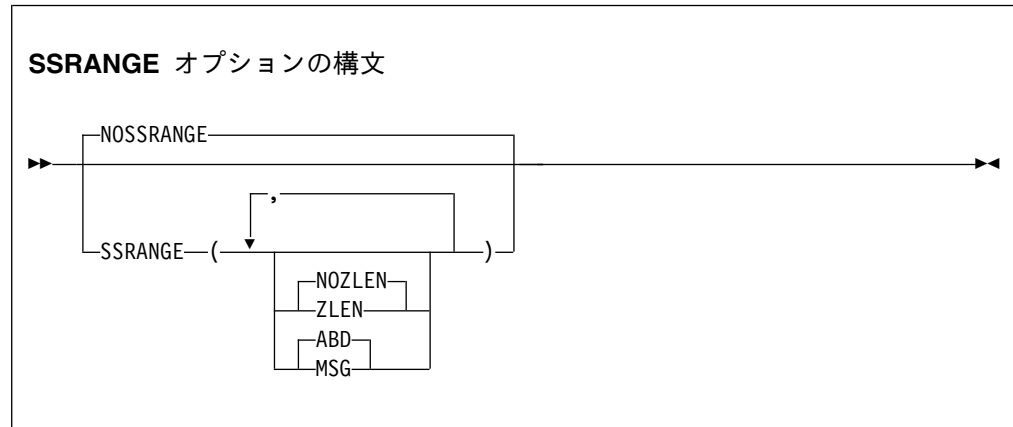
533 ページの『IMS サブオプションの分離』

関連参照

349 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: NOSSRANGE

SSRANGE のみが指定されている場合のサブオプションのデフォルトは NOZLEN,ABD です。

省略形: SSR|NOSSR

SSRANGE は、添え字 (ALL 添え字を含む) または指標が、関連するテーブルの領域外の区域を参照しようとしているかどうかを検査するコードを生成します。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指定された SSRANGE は、SSRANGE(NOZLEN,ABD) を指定したものと受け入れられます。

注: SSRANGE オプションが有効になっている場合は、範囲検査がコンパイラによって生成され、実行時には必ずその検査が実施されます。ランタイム・オプション CHECK(OFF) を指定しても、コンパイルされた範囲検査を実行時に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。

次の点を確認するために、参照変更式が検査されます。

- 開始位置が 1 以上である。
- 開始位置が、サブジェクト・データ項目の現在の長さより大きくない。
- 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超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 サブオプションおよび NOZLEN サブオプションは、コンパイラーによる参照変更長の検査方法を制御します。

- ZLEN が有効になっている場合、コンパイラーは、参照変更長がゼロ以上になるようにコードを生成します。ゼロ長の参照変更が指定されると、実行時に SSRANGE エラーが出力されません。
- NOZLEN が有効になっている場合、コンパイラーは、参照変更長が 1 以上になるようにコードを生成します。ゼロ長の参照変更が指定されると、実行時に SSRANGE エラーが発生します。これは、旧バージョンの COBOL での SSRANGE の動作方法と互換性があります。

サブオプション MSG および ABD は、範囲検査が失敗したときの COBOL プログラムの実行時動作を制御します。

- MSG が有効になっているときに範囲検査が失敗すると、ランタイム警告メッセージが発行されます。つまり、プログラムの実行は継続され、他の範囲外状態も特定される可能性があります。
- ABD が有効になっているときに範囲検査が失敗すると、最初の範囲外状態でランタイム・エラー・メッセージが発行され、プログラムが異常終了します。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

126 ページの『参照修飾子』

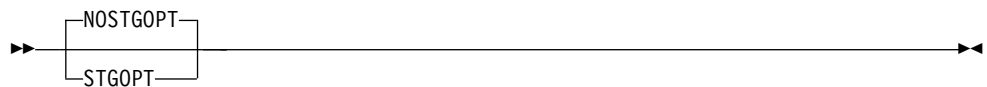
関連タスク

459 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: NOSTGOPT

省略形: SO|NOSO

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 節のある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。

また、STGOPT を使用すると、メモリー内で LOCAL-STORAGE SECTION のデータ項目を再配列し、パフォーマンスを最適化することができます。

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。

RULES(UNREF | NOUNREFALL | NOUNREFSOURCE) オプションを使用して、参照されていないデータ項目についての警告メッセージを出すかどうかを制御することもできます。詳しくは、411 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: SUPPRESS

省略形: SUPP | NOSUPP

NOSUPPRESS

COPY ステートメントの SUPPRESS 句を無視します。

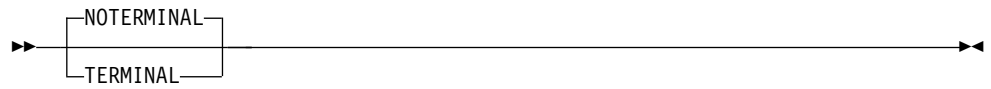
SUPPRESS

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: NOTERMINAL

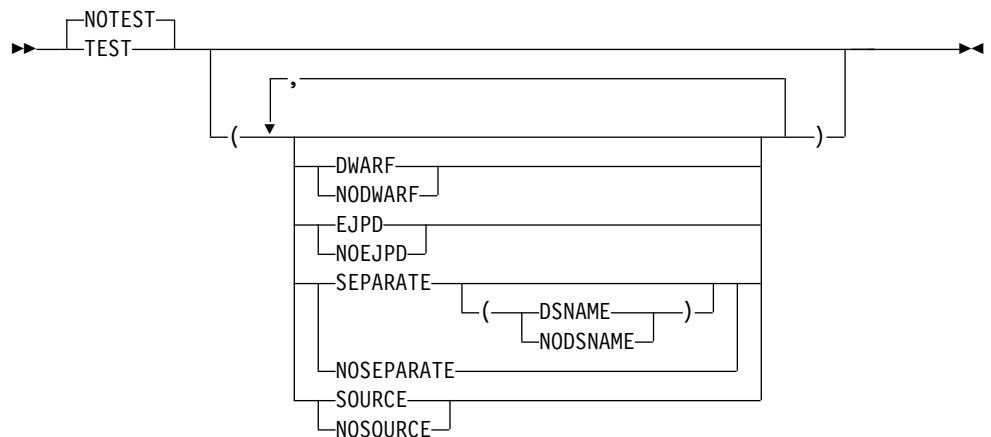
省略形: TERM NOTERM

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 は、IBM z/OS Debuggerや Fault Analyzer などの問題判別ツールを使用したデバッグを可能にするオブジェクト・コードを生成する場合に使用します。TEST を使用すれば、言語環境プログラムによって生成された定様式ダンプにあるシンボリック変数の組み込みも可能になります。

TEST オプションの構文



デフォルト: 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る場合)
- NOEJPD, DWARF, SOURCE, NOSEPARATE (TEST がサブオプションなしで指定される場合)

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 を表します。
- NOSEP | SEP: SEPARATE | NOSEPARATE を表します。

DWARF | NODWARF

TEST(DWARF) が有効になっている場合は、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、または SEPARATE サブオプションが有効になっていれば別のデバッグ・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) によって入手できる DWARF 情報のサブセットです。NOTEST(DWARF) が有効になっている場合に生成される DWARF 診断情報は、IBM z/OS Debugger では使用できません。デバッガー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したいが、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が向上している場合は、NOTEST(DWARF) の使用を検討してください。

コンパイラー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ることもありません。

注:

- SOURCE および SEPARATE は NODWARF と一緒に使用できません。
- TEST または NOTEST の DWARF サブオプションを指定した場合は、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、359 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 Debugger のコマンド JUMPTO および GOTO の有効化を制御します。EJPD および NOEJPD が有効になるのは、TEST オプション、およびゼロ以外の OPTIMIZE レベル (OPTIMIZE(1) または OPTIMIZE(2)) を指定した場合に限られます。

TEST(EJPD) およびゼロ以外の OPTIMIZE レベルを指定すると、以下が行われます。

- JUMPTO および GOTO が有効化されます。
- 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(NOEJPD) およびゼロ以外の OPTIMIZE レベルを指定すると、以下が行われます。

- JUMPTO コマンドおよび GOTO コマンドは有効になりませんが、SET WARNING OFF IBM z/OS Debugger コマンドを使用すれば、JUMPTO や GOTO を使用できます。この場合、JUMPTO および GOTO の結果は予測不能なものになります。
- 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 と一緒には使用できません。

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 は指定できません。

NOSOURCE を指定した場合、生成された DWARF デバッグ情報に拡張ソース・コードは組み込まれません。IBM z/OS Debugger で TEST(NOSOURCE) オプションを使用してデバッグを行うことはできなくなります。

SEPARATE[(DSNAME|NODSNAME)] | NOSEPARATE

SEPARATE がサブオプションなしで指定された場合、デフォルトは SEPARATE(NODSNAME) です。

SEPARATE または SEPARATE(NODSNAME) は、デバッグ能力を保持しながらディスク上のプログラム・オブジェクト・サイズを制御する場合に指定します (NOSEPARATE は、ロードされるプログラム・オブジェクトのサイズに影響しません)。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) が有効な場合、コンパイル中に使用される 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ます。この名前は、DWARF 情報が要求された場合、実行時にデフォルトとして使用されます。データ・セット名は、SYSDEBUG COBOL 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 が有効な場合、IGZIUXB ユーザー出口が提供するののは、プログラムの DWARF デバッグ情報を見つけるメカニズムのみであ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 は指定できません。

- IBM デバッガーを使用して、SYSDEBUG データ・セットに含まれる DWARF デバッグ情報をデバッグできるようにするには、以下のレベルにある何らかのツールが必要です。
 - IBM Debug for z Systems V14.1 (5655-Q50) (旧 IBM Debug Tool for z/OS) 以降
 - IBM Developer for z Systems V14.1 (5724-T07) 以降
 - IBM Application Delivery Foundation for z Systems V3.1 (5655-AC6) 以降
- NOSEPARATE は、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場合に指定します。

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 サブオプションを指定すると、コンパイラーは IBM z/OS Debugger がデータ名や段落名などの解決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のか、または別の SYSDEBUG データ・セットに書き込むのかを選択することができます。

- 実行可能ファイルを小さくする場合は、SEPARATE サブオプションを使用して、IBM z/OS Debugger セッションで使用するデバッグ・ファイルを別に保持します。
- 別のデバッグ・ファイルを管理する必要があるようにするには、NOSEPARATE サブオプションを使用してコンパイルを行います。ただし、このサブオプションを使用すると DASD 上でオブジェクト・プログラムのサイズが増えることに注意してください。NOSEPARATE オプションを使用すれば、オブジェクト・プログラムが仮想記憶にロードされるときにサイズが増えることはありません。

JCL または TSO から COBOL コンパイラーを呼び出して NOTEST|TEST (... ,SEPARATE,...) を指定した場合、DWARF デバッグ情報は、SYSDEBUG DD ステートメントで指定したデータ・セットに書き込まれます。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 UNIX シェルから COBOL コンパイラーを呼び出して NOTEST|TEST(. . . ,SEPARATE,. . .) を指定した場合、DWARF デバッグ情報は現行ディレクトリー内の *file.dbg* に書き込まれます (*file* 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。

- デバッグが一部制限されますが、最良のパフォーマンスを得るには、ゼロ以外の OPTIMIZE レベル、STGOPT、および TEST(NOEJPD) オプションを使用してコンパイルします。

- IBM z/OS Debugger のコマンド JUMPTO および GOTO がサポートされません。ただし、SET WARNING OFF IBM z/OS Debugger ・ コマンドを使用する場合、JUMPTO および GOTO は使用可能です。この場合、JUMPTO および GOTO の結果は予測不能なものになります。
- DESCRIBE ATTRIBUTES コマンドを除く IBM z/OS Debugger コマンドは、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 Debugger のコマンド AT CALL *entry-name* がサポートされません。
- 上記の実動デバッグ・シナリオよりプログラムのパフォーマンスは低下しますが、IBM z/OS Debugger のコマンド JUMPTO および GOTO の予測可能な動作を有効にするには、ゼロ以外の OPTIMIZE レベルと TEST(EJPD) を指定します。

STGOPT オプションによって破棄される項目の参照について、および AT CALL コマンドについての上記の制約は、ゼロ以外の OPTIMIZE レベルと TEST(EJPD) を使用した場合にも適用されます。

- パフォーマンスは最も遅くなりますが、デバッグ能力を最大にするには、OPTIMIZE(0)、NOSTGOPT および TEST を指定します。

OPTIMIZE(0) オプションを使用すると、コンパイラーはより低速のコードを生成しますが、すべてのIBM z/OS Debugger ・ コマンドがサポートされます。

言語環境プログラム:

TEST オプションが何らかのサブオプションとともに指定された場合、および NOTEST が DWARF とともに指定された場合に以下の 2 つの機能をダンプに追加すると、言語環境プログラムからのフォーマット済みダンプがより良くなる可能性があります。

- 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。
- プログラム変数の値 (DWARF が有効になっている場合)。

DWARF が指定されている場合、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示されます。NODWARF を使用すれば、ダンプでは、プログラム変数も失敗したステートメントの行番号も示されません。

Enterprise COBOL は、言語環境プログラムが提供するダンプ・サービスを使用して、言語環境プログラム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生成します。

処理されない条件に対して言語環境プログラムがダンプを作成するかどうかは、ランタイム・オプション TERMTHDACT の設定値によって決まります。TERMTHDACT(DUMP) を指定した場合は、重大度 2 以上の条件が処理されないと、ダンプが生成されます。

注: IBM z/OS Debugger は以下の製品のコンポーネントです。

- IBM Developer for z Systems Enterprise Edition (IBM Application Delivery Foundation for z Systems に組み込まれている)
- IBM Debug for z Systems (旧 IBM Debug Tool for z/OS)

- IBM Developer for z Systems

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/support/knowledgecenter/SSQ2R2_14.0.0/com.ibm.debugtool.doc/common/dcompo.html を参照してください。

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 (共通デバッグ・アーキテクチャーについて)

関連タスク

306 ページの『デバッグ・データ・セットの定義 (SYSDEBUG)』

463 ページの『デバッガーの使用』

497 ページの『CEEDUMP 処理での情報の抑止抑止』

TERMTHDACT による言語環境プログラム・ダンプの生成
 (言語環境プログラム デバッグのガイド)

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
 (Debug Tool User's Guide)

関連参照

302 ページの『論理レコード長とブロック・サイズ』

329 ページの『cob2 入出力ファイル』

349 ページの『矛盾するコンパイラー・オプション』

402 ページの『OPTIMIZE』

TEST | NOTEST

(言語環境プログラム プログラミング・リファレンス)

THREAD

THREAD は、COBOL プログラムで、複数の POSIX スレッドまたは PL/I タスクを持つ 言語環境プログラム・エンクレープでの実行が有効化されることを示します。

THREAD オプションの構文



デフォルト: 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、言語環境プログラムのエンクレープ内のすべての 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。

Enterprise COBOL より前のコンパイラーでコンパイルされたプログラムは、NOTHREAD を使用してコンパイルされたものとして処理されます。

THREAD オプションが有効化されている場合には、次のエレメントはサポートされません。これらの言語エレメントが検出された場合は、エラーとして診断されます。

- ALTER ステートメント
- DEBUG-ITEM 特殊レジスター
- プロシーチャー名が指定されていない GO TO ステートメント
- PROGRAM-ID 節の INITIAL 句または INITIAL コンパイラー・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、言語環境プログラム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、作成された再使用可能環境や、RTEREUS ランタイム・オプションを使用して作成された再使用可能環境は、THREAD オプションを使用してコンパイルされたプログラムをサポートしません。

パフォーマンスの考慮: THREAD オプションを使用する場合は、自動的に生成される逐次化ロジックのオーバーヘッドが原因で、実行時のパフォーマンスが多少低下することがあります。

関連タスク

603 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

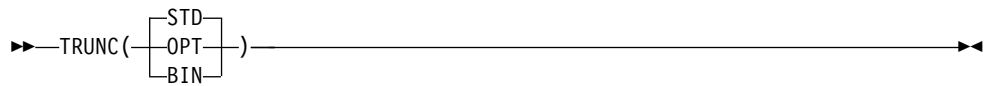
関連参照

349 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: TRUNC(STD)

省略形: なし

TRUNC は、COMP-5 データ項目には効力を持ちません。COMP-5 項目は、TRUNC サブオプションの指定に関係なく、TRUNC(BIN) が有効である場合と同様に処理されます。

TRUNC(STD)

TRUNC(STD) は、MOVE ステートメントおよび算術式の中の USAGE BINARY 受信フィールドにのみ適用されます。TRUNC(STD) が有効であると、算術式の最終結果または MOVE ステートメント中の送信フィールドは、BINARY 受信フィールドの PICTURE 節の桁数に切り捨てられます。

TRUNC(OPT)

TRUNC(OPT) はパフォーマンス・オプションです。TRUNC(OPT) が有効であると、コンパイラーは、データが MOVE ステートメントおよび算術式にある USAGE BINARY 受信フィールドの PICTURE の指定に従うものと想定します。結果は最適な方法で処理され、PICTURE 節の中の桁数か、またはストレージ内の 2 進数フィールドのサイズ (ハーフワード、フルワード、またはダブルワード) に切り捨てられます。

ヒント:

- TRUNC(OPT) オプションを使用するのは、2 進数区域に移動されるデータが、2 進数項目に対する PICTURE 節で定義された値よりも高い精度の値にならないことが確実である場合に限定してください。そうしないと、結果は予測できません。この切り捨ては、想定される最も効果的な方法で実行されます。そのため、結果は、生成される特定のコード・シーケンスに左右されます。特定のステートメントに対して生成されたコード・シーケンスを見なければ、切り捨ての予想は不可能です。

TRUNC(BIN)

TRUNC(BIN) オプションは、USAGE BINARY データを処理するすべての COBOL 言語に適用されます。TRUNC(BIN) が有効な場合、すべての 2 進数項目 (USAGE COMP、COMP-4、または BINARY) は、固有ハードウェア 2 進数項目として、すなわち、それぞれが個々に USAGE COMP-5 と宣言されたものとして処理されます。

- BINARY 受信フィールドは、ハーフワード、フルワード、またはダブルワード境界でのみ切り捨てられます。
- BINARY 送信フィールドは、受け取り側が数値であれば、ハーフワード、フルワード、またはダブルワードとして処理されます。受け取り側が数値でない場合、TRUNC(BIN) は効力を持ちません。
- フィールドの全 2 進内容が重要です。
- DISPLAY は切り捨てを行わずに、2 進フィールドの内容全体を変換します。

推奨事項: 他のプロダクトによって設定される 2 進値を使用するプログラムの場合、推奨オプションは TRUNC(BIN) です。他のプロダクト (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 には、個々のデータ項目に TRUNC(BIN) の性質を適用する効果があります。したがって、すべての 2 進数データ項目に対して TRUNC(BIN) を使用することによるパフォーマンス上のオーバーヘッドは、非 COBOL プログラムまたは他のプロダクトやサブシステムに渡されるデータ項目など、一部の 2 進数データ項目にのみ COMP-5 を指定することによって回避できます。COMP-5 の使用は、どの TRUNC サブオプションが有効であっても影響を受けません。

VALUE 節における大きなリテラル: コンパイラー・オプション TRUNC(BIN) を使用する場合、2 進数データ項目 (COMP、COMP-4、または BINARY) 用の VALUE 節に指定された数値リテラルは、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。

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	00101E213B	123451
受け取り側 TRUNC(STD)	51	00133	51
受け取り側 TRUNC(OPT)	-7621	E213B	2J
受け取り側 TRUNC(BIN)	-7621	E213B	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	491961021D3	1234567891
受け取り側 TRUNC(STD)	567891	001081AA153	567891
受け取り側 TRUNC(OPT)	567891	531AA108100	567891
受け取り側 TRUNC(BIN)	1234567891	491961021D3	1234567891

TRUNC(STD) を指定すると、送り出しデータは BINARY 受け取り側の PICTURE 節に適合するように、6 桁の整数に切り捨てられます。

TRUNC(OPT) を指定すると、コンパイラーは送り出しデータの精度が BINARY 受け取り側の PICTURE 節の精度よりも大きくないと想定します。この場合、最も効率のよいコード・シーケンスは、TRUNC(STD) が指定されているものとして切り捨てを行うことです。

TRUNC(BIN) を指定すると、BIN-VAR に割り振られた 2 進数フルワードにすべての送り出しデータが収まるため、切り捨ては行われません。

関連概念

54 ページの『数値データの形式』

関連タスク

506 ページの『CICS オプションを使用したコンパイル』

関連参照

395 ページの『NUMCHECK』

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

|

VBREF

VBREF は、ソース・プログラムの中で使用されるすべてのステートメントと、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: 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。これにより、ご使用のプログラムにレコード長の矛盾を引き起こす READ ステートメントがある場合に、旧バージョンから Enterprise COBOL V6 への移行が容易になります。

VLR オプションの構文



デフォルト: 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 コンパイラー・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) 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: この設定により、不正な長さの READ 状態で引き起こされる入出力問題を隠すことができます。VLR(COMPAT) オプションの使用には注意が必要です。また、READ ステートメントが正しいかどうかを確認してください。

VLR(STANDARD)

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。

FS=04 をテストするためのコードを追加して、レコード内の未定義データをアクセスしないように、また切り捨てられたレコード部分を参照しようとした場合に保護例外を受け取らないようにすることができます。

VLR(STANDARD) を使用すると、「不正な長さの READ」が発生した可能性がある場合にファイル状況によってそのことが示されるため、より信頼性の高いコードが生成され、入出力の問題は減少します。新しいコンパイラー・メッセージ MSGIGYP3178 も、「不正な長さの READ」の可能性がプログラムにあるかどうかを通知することにより、入出力問題の回避に役立ちます。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可能な「不正な長さの READ」の可能性を示すことにより、VLR(COMPAT) から VLR(STANDARD) への移行の支援に使用できます。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 コンパイラー・オプションの MSGEXIT サブオプションを使用して、メッセージ MSGIGYP3178 の重大度を I (RC=0) から S (RC=12)、E (RC=8)、または W (RC=4) に変更します。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

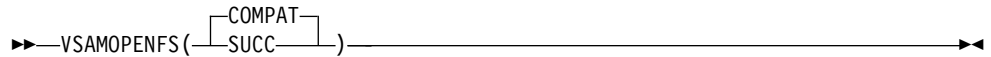
374 ページの『EXIT』

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、ファイル整合性検査の確認を必要とする、正常に実行された VSAM OPEN ステートメントから報告されるユーザー・ファイル状況に作用します。

VSAMOPENFS オプションの構文



デフォルト: VSAMOPENFS(COMPAT)

省略形: VS(C|S)

COMPAT

VSAMOPENFS(COMPAT) を指定した場合、VSAM OPEN ステートメントが正常に検査されると、ステートメントはファイル状況 97 を返します。これは、V6 より前の COBOL の実行時動作と互換性があります。

SUCC

VSAMOPENFS(SUCC) を指定した場合、VSAM OPEN ステートメントが正常に検査されると、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、他の正常な操作で通常行うように、返されたファイル状況の最初の桁で 0 の有無を確認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: NOWORD

省略形: 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。

代替予約語テーブルを使用して、IBM 提供のデフォルトの予約語テーブルに対する変更を行います。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

506 ページの『CICS オプションを使用したコンパイル』

関連参照

349 ページの『矛盾するコンパイラー・オプション』

511 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



図は、XMLPARSE オプションの構文を示しています。XMLPARSE() の括弧内には、XMLSS と COMPAT の二つのオプションが並列に配置されています。XMLSS は上向き、COMPAT は下向きの括弧で囲まれています。図の両端には矢印が描かれています。

デフォルト: XMLSS

省略形: XP(X|C)

XMLPARSE(XMLSS) オプションを指定すると、XML PARSE ステートメントは z/OS XML System Services パーサーを使用して処理されます。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 スキーマと照合した XML 入力文書の検証 (XML PARSE ステートメントの VALIDATING 句を使用)
- 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)
- 文書フラグメントの Unicode UTF-16 への自動変換 (XML PARSE ステートメントの RETURNING NATIONAL 句を使用)
- 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

XMLPARSE(COMPAT) オプションを指定した場合、XML PARSE ステートメントは、COBOL ライブラリー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、Enterprise COBOL バージョン 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バージョン 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

627 ページの『第 31 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

XREF

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。

XREF オプションの構文



デフォルト: XREF(FULL)

省略形: XINOX

XREF、XREF(FULL)、または XREF(SHORT) を選択できます。サブオプションを付けずに 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。

491 ページの『例: XREF 出力: データ名相互参照』

492 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。

493 ページの『例: XREF 出力: COPY/BASIS 相互参照』

EBCDIC データ名および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るデータ項目のみを表示します。XREF (SHORT) は、DBCS データ名とプロシージャー名、および単一バイトの名前に適用されます。

NOXREF を使用すると、このリストは抑止されます。

使用上の注意

- MOVE CORRESPONDING ステートメントで使用されるグループ名は、XREF リストに入れられます。それらのグループの基本名もリストされています。
- データ名の XREF リストでは、文字 M が前に付いている行番号は、そのデータ項目がその行のステートメントによって明示的に変更されたことを示しています。
- XREF リストは追加のストレージを使用します。
- 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

451 ページの『第 19 章 デバッグ』

関連タスク

464 ページの『リストの入手』

関連参照

言語環境プログラム デバッグのガイド (COBOL コンパイラー・オプション)

ZONECHECK

ZONECHECK オプションは、送信データ項目として使用するゾーン 10 進データ項目に対して IF NUMERIC クラス・テストを生成するようにコンパイラーに指示する場合に使用します。

注: ZONECHECK は非推奨ですが、互換性のために使用できるようにはなっています。これは NUMCHECK(ZON(ALPHNUM)) で置き換えられます。

ZONECHECK オプションの構文



デフォルト: NOZONECHECK

省略形: 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 テストを実行するよう要求し、データが無効 (例: NOT NUMERIC) の場合は行番号、データ項目名、データ項目内容、およびプログラム名を示すランタイム警告メッセージが出されます。ZONECHECK(MSG) は、NUMCHECK(ZON,MSG) が有効になっているように処理されます。

ABD

ABD 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 テストが実行されるよう要求し、データが無効 (例: NOT NUMERIC) の場合は強制終了メッセージが出され、異常終了となります。ZONECHECK(ABD) は、NUMCHECK(ZON,ABD) が有効になっているように処理されます。

ZONECHECK(MSG) と ZONECHECK(ABD) のどちらを使用した場合も、コンパイラーは COBOL ステートメント内で送信側として参照される各ゾーン 10 進数データ項目に対して、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 の場合は、警告メッセージ (ZONECHECK(MSG) の場合) または強制終了メッセージ (ZONECHECK(ABD) の場合) が出されます。
- データが NUMERIC の場合、ステートメントの外部動作は、低速になる以外は NOZONECHECK と同じです。

制約事項: CALL ステートメントの場合、ZONECHECK は、数値 USAGE DISPLAY である BY CONTENT データ項目と BY VALUE データ項目のみを検査し、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) と ZONECHECK(ABD) は、プログラム内の COBOL ステートメントで使用するゾーン 10 進数データ項目の数によっては、NOZONECHECK よりもはるかに低速になります。

関連タスク

61 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

395 ページの『NUMCHECK』

398 ページの『NUMPROC』

『ZONEDATA』

ZONEDATA

ZONEDATA オプションは、USAGE DISPLAY データ項目および 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通知し、有効でない場合はコンパイラーの動作を指示します。

ZONEDATA オプションの構文

→ ZONEDATA (PFD
MIG
NOPFD) →

デフォルト: ZONEDATA(PFD)

省略形: 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、SIGN LEADING のある上位バイトか、SIGN IS SEPARATE の場合は別個のバイトにあります。

ZONEDATA(PFD)

ZONEDATA(PFD) オプションが有効な場合、コンパイラーは、USAGE DISPLAY データ項目および 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、数値比較を行うために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、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、コンパイラーは、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。コンパイラーはまた、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

ードがある場合や、ゾーン 10 進数データ項目に無効なゾーン・ビットがある場合に、COBOL V4 (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ことが分かっている最適化の実行を避けます。

ZONEDATA(NOPFD)

ZONEDATA(NOPFD) オプションが有効な場合、コンパイラーは、COBOL V4 (またはそれ以前のバージョン) で NUMPROC(NOPFD|PFD) を使用した場合に COBOL V4 (またはそれ以前のバージョン) が行うのと同じ方法でゾーン 10 進数データの数値比較または英数字比較を行うための命令を生成します。

- COBOL V4 (またはそれ以前のバージョン) でゾーン・ビットが考慮されていた場合、コンパイラーは英数字比較を生成します。この比較でも、ゾーン 10 進数データ項目内の各桁のゾーン・ビットが考慮されます。ゾーン 10 進数値はゾーン 10 進数のままです。
- COBOL V4 (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、コンパイラーは数値比較を生成します。この比較では、ゾーン 10 進数データ項目内の各桁のゾーン・ビットは無視されます。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。

COBOL V4 (またはそれ以前のバージョン) で行われていたのと同じ方法でゾーン 10 進数データの比較をコンパイラーで生成するためには、COBOL V6 で使用される NUMPROC サブオプションが、COBOL V4 (またはそれ以前のバージョン) で使用された NUMPROC サブオプションに一致しなければなりません。

- COBOL V6 で COBOL V4 (またはそれ以前のバージョン) の NUMPROC(NOPFD) 動作になるようにするには、COBOL V6 で ZONEDATA(NOPFD) および NUMPROC(NOPFD) を使用します。
- COBOL V6 で COBOL V4 (またはそれ以前のバージョン) の NUMPROC(PFD) 動作になるようにするには、COBOL V6 で ZONEDATA(NOPFD) および NUMPROC(PFD) を使用します。

コンパイラーはまた、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ある場合や、ゾーン 10 進数データ項目に無効なゾーン・ビットがある場合に、COBOL V4 (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ことが分かっている最適化の実行を避けます。

注: 符号コードは、NUMPROC コンパイラー・オプション設定に準じた有効な符号コードでなければなりません。また、下位バイトには、SIGN IS LEADING または SIGN IS SEPARATE のいずれかによる符号なしおよび符号付きの有効なゾーン (x'F') が必要です。

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のある USAGE DISPLAY データ項目または PACKED-DECIMAL データ項目に対する、MOVE ステートメントの動作、比較、および計算に作用します。

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、データ項目は以下のように扱われます。

- COBOL V4 (またはそれ以前のバージョン) では、テストは、NUMPROC(MIG) オプションが使用されている場合は true であり、NUMPROC(NOPFD|PFD) の場合は false です。
- COBOL V5 以降のバージョン:
 - ZONEDATA(PFD) が使用されている場合、テストは OPT(0) で true であり OPT(1|2) で false です。
 - 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 V6 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、COBOL V6 の使用時に、ZONEDATA(PFD)、および COBOL V4 (またはそれ以前のバージョン) で使用したのと同じ NUMPROC 設定を使用します。
- 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、以下を行います。
 - COBOL V4 (またはそれ以前のバージョン) で NUMPROC(MIG) を使用した場合は、ZONEDATA(MIG) と NUMPROC(NOPFD) を COBOL V6 で使用します。
 - COBOL V4 (またはそれ以前のバージョン) で NUMPROC(NOPFD) を使用した場合は、ZONEDATA(NOPFD) と NUMPROC(NOPFD) を COBOL V6 で使用します。
 - COBOL V4 (またはそれ以前のバージョン) で NUMPROC(PFD) を使用した場合は、ZONEDATA(NOPFD) と NUMPROC(PFD) を COBOL V6 で使用します。

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、比較を行う場合でも、ZONEDATA(NOPFD) によって、どのようなケースでも COBOL V4 と同じ結果を得られるわけではありません。

パフォーマンスの考慮事項: ZONEDATA(PFD) では ZONEDATA(NOPFD|MIG) よりも良好なランタイム・パフォーマンスを得られます。ZONEDATA(NOPFD|MIG) を使用すると、NUMPROC(PFD) によって得られる最適化の一部が無効になります。

関連タスク

61 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

395 ページの『NUMCHECK』

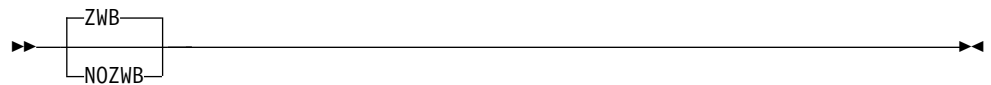
398 ページの『NUMPROC』

438 ページの『ZONECHECK』

ZWB

ZWB を使用してコンパイルすると、コンパイラーは、実行時に符号付きゾーン 10 進数 (DISPLAY) フィールドを英数字基本フィールドと比較する前に、そのフィールドから符号を除去します。

ZWB オプションの構文



デフォルト: ZWB

省略形: なし

ゾーン 10 進数項目がスケール項目である場合 (すなわち、記号 P をその PICTURE スtring内に含んでいる場合)、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。

第 18 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

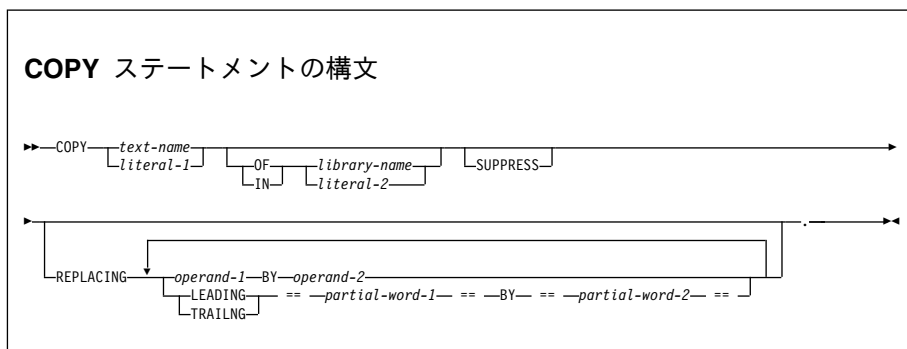
CALLINTERFACE 指示は、CALL ステートメントおよび SET ステートメントのインターフェース規約を指定します。指定した規約は、ソース内で別の CALLINTERFACE 指示が検出されるまで有効のままになります。

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。

text-name も *library-name* 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。

text-name および *library-name* の固有性は、システム依存名の形式化および変換の規則が適用された後で決められます。*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。

以下に例を示します。

```
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 UNIX 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。以下に例を示します。

```
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、COPY MYCOPY OF MYLIB が次のように設定されているとします。

```
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 ディレクティブ

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 ディレクティブ

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 ディレクティブ

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 ディレクティブ

INLINE ディレクティブを使用すれば、コンパイラがプロシージャー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されるコンパイラ・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ー: 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』
- 307 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』
- 323 ページの『z/OS UNIX のもとでのコンパイラー・オプションの指定』
- 321 ページの『z/OS UNIX のもとでの環境変数の設定』
- 795 ページの『反復コーディングの除去』

関連参照

- 327 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 19 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、デバッグ・ツールを使用できます。デバッグ・ツールは、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、デバッグ・ツールのフルスクリーン・サービスと 3270 装置上のセッション・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。デバッグ・ツールが、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

デバッグ・ツールは、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

ワークステーション・ユーザーは、z/OS 上で実行されるプログラムをデバッグするために、IBM Debug Tool Plug-in for Eclipse または IBM Problem Determination Tools を IBM Developer for z Systems で使用できます。

注: IBM Debug for z Systems は、IBM Debug Tool for z/OS に置き換わるものです。COBOL 文書ライブラリーで、IBM Debug Tool for z/OS への参照がすべて変更されているわけではありません。多くの COBOL アプリケーションのデバッグに、引き続き IBM Debug Tool for z/OS V13.1 を使用することができます。ただし、Enterprise COBOL V6 に用意されている新しいデバッグ機能を使用する場合は、最新バージョンの IBM Debug for z Systems が必要です。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/support/knowledgecenter/SSQ2R2_14.0.0/com.ibm.debugtool.doc/common/dcompo.html を参照してください。

関連タスク

『ソース言語によるデバッグ』

456 ページの『コンパイラー・オプションを使用したデバッグ』

463 ページの『デバッガーの使用』

464 ページの『リストの入手』

497 ページの『CEEDUMP 処理での情報の抑止抑止』

Debug Tool User's Guide

関連参照

Debug Tool for z/OS Debug Tool Utilities and Advanced Functions for z/OS リファレンスおよびメッセージ

言語環境プログラム デバッグのガイド (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。

障害のあるプログラムが既に実動中の大規模なアプリケーションの一部である場合 (ソース更新を除外する) は、プログラムの障害部分をシミュレートするような小さいテスト・ケースを作成してください。テスト・ケースでは、以下の問題の検出に役立つようなデバッグ機能をコーディングしてください。

- プログラム・ロジックのエラー
- 入出力エラー
- データ型のミスマッチ
- 初期化されていないデータ
- プロシーチャーの問題

関連タスク

『プログラム・ロジックのトレース』

453 ページの『入出力エラーの検出および処理』

454 ページの『データの妥当性検査』

454 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』

454 ページの『プロシーチャーに関する情報の生成』

関連参照

ソース言語のデバッグ (*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。

例えば、問題が EVALUATE ステートメントまたは 1 組のネストされた IF ステートメントにあると判断した場合は、それぞれのパスで DISPLAY ステートメントを使用して、ロジック・フローを調べます。問題の原因が数値の計算方法にあると判断した場合は、DISPLAY ステートメントを使用して、いくつかの中間結果の値を検査することができます。

プログラムの中で明示範囲終了符号を使用してステートメントを終了させるようにすると、ロジックはより明確であり、したがってトレースしやすくなります。

例えば、特定のルーチンが開始して終了したかどうかを判別する場合は、プログラムに次のようなコードを挿入してみてください。

```
DISPLAY "ENTER CHECK PROCEDURE"  
.  
  . (checking procedure routine)  
.  
DISPLAY "FINISHED CHECK PROCEDURE"
```

ルーチンが正しく作動していることを確認したら、次のいずれかの方法で DISPLAY ステートメントを使用不可にします。

- 各 DISPLAY ステートメントの行の 7 桁目にアスタリスクを置き、コメント行に変換する。
- 各 DISPLAY ステートメントの 7 桁目に D を置き、コメント行に変換する。これらのステートメントを再活動化したい場合は、ENVIRONMENT DIVISION に WITH DEBUGGING MODE 節を含めると、7 桁目の D は無視され、DISPLAY ステートメントが実施されます。

プログラムを実動に移す前に、使用したすべてのデバッグ・エイドを削除するか使用不可にしてから、プログラムを再コンパイルします。プログラムはより効果的に実行され、使用するストレージは小さくなります。

関連概念

22 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。

ファイル状況キーをデバッグ・エイドとして使用するためには、各入出力ステートメントの後で、状況キーの値がゼロ以外かどうか検査します。値がゼロでない (エラー・メッセージで報告される) 場合は、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。その後で、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の、あるいはオープン属性 (INPUT、OUTPUT、I-O、または EXTEND) 用に提供されたものです。

各 USE AFTER STANDARD ERROR ステートメントを、PROCEDURE DIVISION 内の DECLARATIVES キーワードに続くセクションにコーディングします。

関連タスク

276 ページの『ERROR 宣言のコーディング』

277 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。

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

108 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。

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*)

382 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。

例えば、プロシージャが何度実行されるかを決定するには、デバッグ・プロシージャを `USE FOR DEBUGGING` 宣言に組み込み、カウンターを使用して、制御がそのプロシージャに渡される回数の記録をとることができます。カウンター技法を使用して、次のような項目を検査できます。

- `PERFORM` ステートメントが実行される回数。特定のルーチンが使用されているかどうか、および制御構造が正しいかどうか。
- ループが実行される回数、ループが実行されているかどうか、およびループの回数が正確かどうか。

プログラムに、デバッグ行かデバッグ・ステートメント、またはその両方を入れることができます。

デバッグ行 は、桁 7 の D で識別されているステートメントです。プログラムのデバッグ行をアクティブにするには、`ENVIRONMENT DIVISION` の `SOURCE-COMPUTER` 行に `WITH DEBUGGING MODE` 節をコーディングする必要があります。この節が含まれていないと、デバッグ行はコメントとしてしか扱われません。

デバッグ・ステートメント とは、`PROCEDURE DIVISION` の `DECLARATIVES` セクションにコーディングされたステートメントです。それぞれの `USE FOR DEBUGGING` 宣言は別個のセクションにコーディングしなければなりません。デバッグ・ステートメントは、次のようにコーディングしてください。

- `DECLARATIVES` セクション内のみ。
- ヘッダー `USE FOR DEBUGGING` の後に置く。
- 最外部のプログラムだけに置く (ネストされたプログラムでは無効です)。デバッグ・ステートメントが、ネストされたプログラムに含まれているプロシージャによって起動されることはありません。

プログラムでデバッグ・ステートメントを使用するには、`WITH DEBUGGING MODE` 節を指定し、さらに、`DEBUG` ランタイム・オプションを使用する必要があります。

オプションに関する制約事項:

- `THREAD` オプションを指定してコンパイルするプログラムでは、`USE FOR DEBUGGING` 宣言を使用できません。

『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`

この例は、以下のプログラム・セグメントは、`DISPLAY` ステートメントと `USE FOR DEBUGGING` 宣言を使用してプログラムをテストする際に必要となるステートメントの種類を示しています。

DISPLAY ステートメントは、情報を端末または出力データ・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.
```

プロシージャー Some-Routine が実行されるたびに、DECLARATIVES SECTION の DISPLAY ステートメントがこのメッセージを出します。

```
Trace For Procedure-Name : Some-Routine 22
```

メッセージの終わりにある番号 22 は、データ項目 Total の累算された値で、Some-Routine が実行された回数を示しています。 デバッグ宣言内のステートメントは、名前を指定されたプロシージャーが実行される前に実行されます。

DISPLAY ステートメントを使用して、プログラムの実行をトレースし、プログラム中のフローを示すこともできます。 これを行うには、DISPLAY ステートメントから Total を除去し、DECLARATIVES SECTION の USE FOR DEBUGGING を次のように変更します。

```
USE FOR DEBUGGING ON ALL PROCEDURES.
```

これで、最外部プログラムのそれぞれの非デバッグ・プロシージャーが実行される前にメッセージが表示されるようになります。

コンパイラー・オプションを使用したデバッグ

ある種のコンパイラー・オプションは、プログラム内のエラーの検出、プログラム内の各種エレメントの検出、リストの取得、およびデバッグのためのプログラムの準備に役立ちます。

コンパイラー・オプション (括弧内に示されているもの) を使用して、以下のエラーを検出することができます。

- 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)

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

『コーディング・エラーの検出』

458 ページの『行シーケンス問題の検出』

458 ページの『無効な COBOL データや無効な COBOL プログラムの検査』

459 ページの『有効範囲の検査』

460 ページの『診断するエラーのレベルの選択』

462 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』

462 ページの『データ項目のリスト』

464 ページの『リストの入手』

関連参照

345 ページの『第 17 章 コンパイラー・オプション』

コーディング・エラーの検出

条件付きでコンパイルしたり、構文検査のみを行ったりする場合は、NOCOMPILE オプションを使用してください。SOURCE オプションと一緒に使用すると、NOCOMPILE は、コーディングの間違い (欠落している定義、正しく定義されていないデータ項目、重複するデータ名など) を見つけるのに役立つリストを作成します。

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。エラーすべてが *x* より低い重大度である場合に、プログラムはコンパイルされます。使用できる重大度レベルは、S (重大)、E (エラー)、および W (警告) であり、この順序で低くなります。

レベル *x* またはそれ以上のエラーが発生した場合、コンパイルは停止し、プログラムの構文検査のみが行われます。

関連参照

362 ページの『COMPILE』

行シーケンス問題の検出

順序どおりになっていないステートメントを見つけるには、SEQUENCE コンパイラー・オプションを使用してください。シーケンス中断は、ソース・プログラムのセクションが移動または削除されたことを表します。

SEQUENCE を使用すると、コンパイラーはソース・ステートメント番号を検査し、昇順になっているかどうかを調べます。順序どおりになっていないステートメント番号の横には、2 つのアスタリスクが入れられます。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

414 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。

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 の終わりを越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。

APAR PH08642 対応 PTF がインストールされているため、NUMCHECK のパフォーマンスは向上しています。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

382 ページの『INITCHECK』

395 ページの『NUMCHECK』

404 ページの『PARMCHECK』

788 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。

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。(この検査は、UNBOUNDED テーブルおよびグループに対しては行われません。)
- 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。(この検査は、UNBOUNDED グループに対しては行われません。)
- 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

420 ページの『SSRANGE』

788 ページの『パフォーマンスに関連するコンパイラー・オプション』

診断するエラーのレベルの選択

FLAG コンパイラー・オプションを使用すると、コンパイル時に診断するエラーのレベルを指定すること、およびエラー・メッセージをリストに組み込みかどうかを指示することができます。すべてのエラーが通知されるようにするには、FLAG(I) または FLAG(I,I) を使用してください。

最初のパラメーターには、発行される構文エラー・メッセージのうち最も重大度レベルの低いものを指定してください。オプションとして、2 番目のパラメーターには、ソース・リストに組み込む構文メッセージのうち最も重大度レベルの低いものを指定します。この重大度レベルは、最初のパラメーターのレベルと同じかそれ以上でなければなりません。両方のパラメーターを指定する場合は、一緒に SOURCE コンパイラー・オプションも指定する必要があります。

表 49. 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。

『例: 組み込みメッセージ』

関連タスク

317 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

318 ページの『コンパイラー診断メッセージの重大度コード』

317 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』

378 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==> IGYP2052-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==> IGYP2121-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==> IGYP2121-S "WS-DATE" was not defined as a data-name. The statement was discarded.
090696**      accept ws-time from time                                          UND
==090696==> IGYP2121-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==> IGYP2121-S "WS-TIME" was not defined as a data-name. The statement was discarded.
090698**      read location-file                                                193
==090698==> IGYP2053-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
                        the start of the next clause.
193     IGYGR1207-S      The "ASSIGN" clause was missing or invalid in the "SELECT" entry for file "LOCATION-FILE".
                        The file definition was discarded.
269     IGYDS1066-S      "REDEFINES" object "WS-DATE" was not the immediately preceding level-1 data item.
                        The "REDEFINES" clause was discarded.
90602   IGYP2052-S      An error was found in the definition of file "LOCATION-FILE". The reference to this file
                        was discarded. Same message on line: 90684
90694   IGYP2121-S      "WS-DATE" was not defined as a data-name. The statement was discarded.
                        Same message on line: 90695
90696   IGYP2121-S      "WS-TIME" was not defined as a data-name. The statement was discarded.
```

```

Same message on line: 90697
90698 IGYPS2053-S An error was found in the definition of file "LOCATION-FILE". This input/output statement
was discarded. Same message on line: 90709
Messages      Total      Informational      Warning      Error      Severe      Terminating
Printed:       13           1           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・エンティティー定義および参照の検出

XREF(FULL) コンパイラー・オプションを使用すると、データ名、プロシージャ名、またはプログラム名が定義および参照されている場所を見つけることができます。また、コピーブックを取得したデータ・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。

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。

- 491 ページの『例: XREF 出力: データ名相互参照』
- 492 ページの『例: XREF 出力: プログラム名相互参照』
- 493 ページの『例: XREF 出力: COPY/BASIS 相互参照』
- 494 ページの『例: XREF 出力: 組み込み相互参照』

関連タスク

- 464 ページの『リストの入手』

関連参照

- 437 ページの『XREF』

データ項目のリスト

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するには、MAP(HEX|DEC) コンパイラー・オプションを使用します。システム・ダンプのデータ項目の内容を突き止める場合に、MAP 出力を使用してください。

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。

MAP リストおよび組み込み MAP 要約の各部分は、ソース全体を通じ、*CONTROL MAP|NOMAP (または *CBL MAP|NOMAP) ステートメントを使用して、選択または抑制することができます。以下に例を示します。

```
*CONTROL NOMAP
      01  A
      02  B
*CONTROL MAP
```

469 ページの『例: MAP 出力』

関連タスク

464 ページの『リストの入手』

関連参照

388 ページの『MAP』

デバッガーの使用

デバッグ・ツールを使用して、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。

リモート・デバッグの場合、z/OS 下または z/OS UNIX 下で実行されるデバッグ・ツール・エンジンによって提供されるデバッグ情報にアクセスするためのクライアント・グラフィカル・ユーザー・インターフェースを備えた Eclipse プラグインがあります。IBM Debug Tool Plug-in for Eclipse は IBM Developer for z Systems だけでなく IBM Problem Determination Tools Studio にも組み込まれています。

TEST の NOSOURCE サブオプションを指定すると、ディスクに保管されるオブジェクト・プログラムをより小さく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、Debug Tool 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 サブオプションを指定すると、Debug Tool ソース・ウィンドウでソースを確認できなくなります。

最大のデバッグ機能を得るには、OPTIMIZE(0)、NOSTGOPT、および TEST のコンパイラー・オプションを指定します。

デバッグ機能の制限をあまり受けることなくパフォーマンスを向上させるには、ゼロ以外の OPTIMIZE レベル、NOSTGOPT コンパイラー・オプション、および TEST(EJPD) コンパイラー・オプションを指定します。

最高のパフォーマンスを得ながら、引き続き Debug Tool 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

Debug Tool User's Guide (デバッグのためのプログラムの準備)

関連参照

423 ページの『TEST』

リストの入手

コンパイラー・オプションを使用して適切なコンパイラー・リストを要求することによって、デバッグに必要な情報を入手してください。

重要: コンパイラーによって作成されるリストは、プログラミング・インターフェースではなく、容易に変更できるものです。

表 50. コンパイラー・オプションとリストの対応

用途	リスト	内容	コンパイラー・オプション
プログラムに有効なオプションのリスト、プログラムに関する統計、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	<ul style="list-style-type: none">プログラムに有効なオプションのリストプログラムに関する統計コンパイルに関する診断メッセージ¹	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	415 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 項目のマッピング	<p>すべての DATA DIVISION 項目および暗黙的に宣言されたすべての項目</p> <p>組み込みマップ要約 (DATA DIVISION 内の、データ宣言が含まれている行のリストの右マージン)</p> <p>ネストされたプログラム・マップ (ネストされたプログラムが含まれているプログラム)</p>	388 ページの『MAP』 ²

表 50. コンパイラー・オプションとリストの対応 (続き)

用途	リスト	内容	コンパイラー・オプション
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。	名前のソートされた相互参照リスト。COPY/BASIS ステートメントおよびコピーブック・データ・セットまたはファイルのソートされた相互参照リスト	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。	437 ページの『XREF』 ^{2,3}
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・コード ³	生成されたコード	387 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	399 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	433 ページの『VBREF』

1. メッセージを除去するには、コンパイル診断情報のレベルを左右するオプション (例えば、FLAG) をオフにしてください。また、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り、メッセージを選択的に抑制することもできます。

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. ソースに *CONTROL LIST および *CONTROL NOLIST (または *CBL LIST および *CBL NOLIST) ステートメントを入れることによって、生成されるオブジェクト・コードの選択的リストを制御することができます。*CONTROL ステートメントは PROCESS (または CBL) ステートメントと異なることに注意してください。

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。

『例: 短縮リスト』
 468 ページの『例: SOURCE および NUMBER 出力』
 469 ページの『例: MAP 出力』
 471 ページの『例: 組み込みマップ要約』
 474 ページの『例: ネストされたプログラム・マップ』
 491 ページの『例: XREF 出力: データ名相互参照』
 492 ページの『例: XREF 出力: プログラム名相互参照』
 493 ページの『例: XREF 出力: COPY/BASIS 相互参照』
 494 ページの『例: XREF 出力: 組み込み相互参照』
 495 ページの『例: OFFSET コンパイラー出力』
 496 ページの『例: VBREF コンパイラー出力』
 496 ページの『例: 条件付きコンパイル出力』

関連タスク

317 ページの『コンパイラー・メッセージのリストの生成』
 474 ページの『LIST 出力の読み取り』
 言語環境プログラム デバッグのガイド (COBOL プログラムのデバッグ)

関連参照

317 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(1),X,MAP,NOLIST,RENT,OPT(1),SSR
  TEST TRUNC(OPT)
Options in effect:         (4)
  NOADATA
  NOADV
    AFP(VOLATILE)
    QUOTE
    ARCH(7)
    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(1)
  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(NOEJPD,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  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 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 IGYPG3113-W Truncation of high-order digit positions may occur due to precision of intermediate results
                  exceeding 30 digits.

```

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 で定義された項目のマップ・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・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言語拡張を抑制します。

TEST このプログラムは、デバッグ・ツールおよび問題判別ツール (デバッグ・ツールや Fault Analyzer など) で使用するために、また CEEDUMP でリストされた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トで、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) コンパイル単位のプログラム統計。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      ***          D O  M A I N  L O G I C          ***
000872      ***
000873      *** Initialization. Read and process update transactions until ***
000874      *** EOE. Close files and stop run.          ***
000875      *****/
000876      procedure division.
000877      000-do-main-logic.
000878      display "PROGRAM IGYTCARA - Beginning".
000879      perform 050-create-vsam-master-file.
000880      perform 100-initialize-paragraph.

```

```

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。

Data Division Map

(1)
Data Definition Attribute codes (rightmost column) have the following meanings:
D = Object of OCCURS DEPENDING G = GLOBAL S = Spanned file
E = EXTERNAL O = Has OCCURS clause U = Undefined format file
F = Fixed-length file OG= Group has own length definition V = Variable-length file
FB= Fixed-length blocked file R = REDEFINES VB= Variable-length blocked file
X = Unallocated

(2) Source LineID	(3) Hierarchy	(4) Data Name	(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	000000000	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	000000000	DS 0CL121	Group	X
455	2	FILLER	BLL=XXXXX	000000000	DS 36C	Display	X
456	2	PRINT-CITY	BLL=XXXXX	000000002	DS 13C	Display	X
457	2	FILLER	BLL=XXXXX	000000003	DS 3C	Display	X
458	2	PRINT-STATE	BLL=XXXXX	000000007	DS 2C	Display	X
459	2	FILLER	BLL=XXXXX	000000010	DS 1C	Display	X
460	2	PRINT-ZIPCODE	BLL=XXXXX	000000016	DS 5C	Display	X
461	2	FILLER	BLL=XXXXX	000000018	DS 1C	Display	X
462	2	PRINT-WORK-PHONE	BLL=XXXXX	000000014	DS 14C	Display	X
463	2	FILLER	BLL=XXXXX	000000018	DS 1C	Display	X
464	2	PRINT-WORK-JUNCTION	BLL=XXXXX	000000025	DS 25C	Display	X
465	2	FILLER	BLL=XXXXX	000000020	DS 20C	Display	X (10)
467	1	DETAIL2-LINE	BLL=00002	000000000	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	

- (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するデータ名。
- (5) このデータ項目に使用されるベース・ロケーター。
- (6) 収容構造の先頭からの 16 進変位。(MAP(HEX) オプションが有効である場合。 MAP(DEC) オプションが有効である場合は、10 進変位が示されます。)
- (7) 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。 構造に可変長フィールドが含まれている場合、構造の最大長が示されます。
- (8) データ型および使用法。
- (9) データ定義属性コード。 定義は DATA DIVISION マップの先頭で説明されています。
- (10) DETAILX-LINE は PROCEDURE DIVISION で参照されませんでした。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・ロケーターは XXXXX に設定されました。

471 ページの『例: 組み込みマップ要約』

474 ページの『例: ネストされたプログラム・マップ』

関連参照

472 ページの『MAP 出力で使用する用語』

473 ページの『LIST および MAP 出力で使用する記号』

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).
. . .
000105 Working-storage section.
000106 01 Working-storage-for-IGYCARA pic x.
000107
000108 77 comp-code pic $9999 comp.
000109 77 ws-type pic x(3) value spaces.
000135 01 i-f-status-area.
000136 05 i-f-file-status pic x(2).
000137 88 i-o-successful value zeroes.
000138
000139
000140 01 status-area.
000141 05 commuter-file-status pic x(2).
000142 88 i-o-okay value zeroes.
000143 05 commuter-vsam-status.
000144 10 vsam-r15-return-code pic 9(2) comp.
000145 10 vsam-function-code pic 9(1) comp.
000146 10 vsam-feedback-code pic 9(3) comp.
000147
000148 77 update-file-status pic xx.
000149 77 loccode-file-status pic xx.
000150 77 updprint-file-status pic xx.
000151
000216 01 ws-commuter-record EXTERNAL.
000217 05 ws-commuter-key.
000218 10 ws-commuter-generic-key.
000219 15 commuter-shift pic x.
000220 15 commuter-home-code pic xx.
000221 15 commuter-work-code pic xx.
000222 10 commuter-name pic x(9).
000223 10 commuter-initials pic xx.
000224 05 commuter-address pic x(18).
000225 05 commuter-city pic x(13).
000226 05 commuter-state pic xx.
000227 05 commuter-zipcode pic 9(5) comp-3.
. . .
000395 Linkage Section.
000396 01 detail1-line.
000397 05 filler pic xx.
000398 05 print-transaction-code pic x.
000399 05 filler pic x(4).
000400 05 print-record-type pic x(3).
000401 05 filler pic xxx.
000402 05 print-shift pic x.
000403 05 filler pic x.
000404 05 print-home-code pic xx.
000405 05 filler pic x.
000406 05 print-work-code pic xx.
000407 05 filler pic xx.
000408 05 print-name pic x(9).
000409 05 filler pic xx.
000410 05 print-initials pic xx.
. . .
000487 procedure division.
000488 000-do-main-logic.
000489 display "PROGRAM IGYTCARA - Beginning".

```

(1)	(2)	(3)
BLF=00001		0CL80
BLF=00001,000000000		16C
BLF=00001,000000016		64C
		1C
		2C
		3C
		0CL2
	000000000	2C
		0CL8
	000000000	2C
	000000002	0CL6
	000000002	2C
	000000004	2C
	000000006	2C
		2C
		2C
		2C
BLX=00001		0CL81
BLX=00001,000000000		0CL16
BLX=00001,000000000		0CL5
BLX=00001,000000000		1C
BLX=00001,000000001		2C
BLX=00001,000000003		2C
BLX=00001,000000005		9C
BLX=00001,000000014		2C
BLX=00001,000000016		18C
BLX=00001,000000034		13C
BLX=00001,000000047		2C
BLX=00001,000000049		3P
BLL=00001		0CL121
BLL=00001,000000000		2C
BLL=00001,000000002		1C
BLL=00001,000000003		4C
BLL=00001,000000007		3C
BLL=00001,000000010		3C
BLL=00001,000000013		1C
BLL=00001,000000014		1C
BLL=00001,000000015		2C
BLL=00001,000000017		1C
BLL=00001,000000018		2C
BLL=00001,000000020		2C
BLL=00001,000000022		9C
BLL=00001,000000031		2C
BLL=00001,000000033		2C

(1) このデータ項目に使用されるベース・ロケーター。

(2) 収容構造の先頭からの 10 進変位。MAP(DEC) オプションが有効であること

を示します。MAP(HEX) オプション、またはサブオプションなしの MAP を指定した場合は、16 進変位が示されます。

- (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

473 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 51. MAP 出力で使用される用語

用語	定義	説明
ALPHABETIC	DS <i>n</i> C	英字データ項目 (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	表示浮動小数点データ項目 (USAGE DISPLAY)
DISPLAY	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 10 進数データ項目 (USAGE DISPLAY)
DISP-NUM-EDIT	DS <i>n</i> C	数字編集データ項目 (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)
GROUP	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
INDEX	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	国別グループ (GROUP-USAGE NATIONAL)
NAT-GRP-VARLEN	DS 0CL <i>n</i> ¹	国別可変長グループ (GROUP-USAGE NATIONAL)
NAT-NUM	DS <i>n</i> C	国別 10 進数データ項目 (USAGE NATIONAL)
NAT-NUM-EDIT	DS <i>n</i> C	国別数字編集データ項目 (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)

表 51. MAP 出力で使用する用語 (続き)

用語	定義	説明
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. n 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 出力で使用する記号

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 52. LIST および MAP 出力で使用する記号

記号	定義
BLF_ n^1	ファイルのベース・ロケーター
BLL_ n^1	LINKAGE SECTION のベース・ロケーター
BLO_ n^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_ n^1	XML-TEXT および XML-NTEXT のベース・ロケーター
BLV_ n^1	位置が変更できるデータのベース・ロケーター
BLX_ n^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
Pfm _{sv} _cell	保管セル番号を実行する
TSN=N	コンパイラー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
#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	Language Environment Common Anchor Area の開始アドレス
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) 書き込み可能静的域のアドレス
_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

表 52. LIST および MAP 出力で使用される記号 (続き)

記号	定義
_QCON	シンボルのオフセット
_returnValue	プロシーチャーの戻り値
_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・ロケーターの場合は XXXXX になることもあります。これは、STGOPT 処理によって削除されたデータ項目を表します。	

例: ネストされたプログラム・マップ

この例は、MAP コンパイラー・オプションを指定することによって作成される、ネストされたプロシーチャーのマップを示しています。括弧内の番号は、後続の注釈に対応しています。

```

Nested Program Map
Program Attribute codes (rightmost column) have the following meanings:
  C = COMMON
  I = INITIAL (1)
  U = PROCEDURE DIVISION USING... (5)

Source Nesting
LineID Level Program Name from PROGRAM-ID paragraph Program
(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 コンパイラー出力の各部分は、プログラムのデバッグに有用である場合があります。

LIST コンパイラー・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。
 - 有効なコンパイラー・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。

- WORKING-STORAGE に関する情報。この情報は、「Data Division Map」および「Static Map」に含まれています。
- 書き込み可能静的領域 (WSA) の説明。この説明は、リストの「Static Map」セクションまたは「WSA24 Map」セクションにあります。ソースの WORKING-STORAGE 域にあるシンボルは、「Static Map」に示される書き込み可能静的領域にマップされます。

「Data Division Map」セクションと、「Static Map」セクションを使用すると、WORKING-STORAGE で定義されているデータ項目の位置がわかります。これらのデータ項目は書き込み可能静的領域 (WSA または WSA24) にあります。

「Static Map」には、各レベル 1 データ項目の、書き込み可能静的領域の先頭に対する相対オフセットが示されます。「Data Division Map」セクションには、各レベル n データ項目の、レベル 1 メンバーに対する相対オフセットが示されます。これらの両方の情報を使用することにより、書き込み可能静的領域内の任意のデータ・メンバーのオフセットを判別できます。

DATA24 オプションを指定してコンパイルする場合、16 MB 境界より下にマップされるデータ項目は、「WSA24 Map」に示されます。同じプロセスに従って、それらの位置を判別できます。

- プログラムで使用される定数およびリテラルに関する情報。「Constant Area」には、プログラム内の定数およびリテラルに関する情報と、コンパイラによって作成される定数およびリテラルに関する情報があります。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。

483 ページの『例: プログラム初期化コード』

484 ページの『例: MD5 シグニチャー』

485 ページの『例: タイム・スタンプおよびバージョン情報』

485 ページの『例: コンパイラ・オプションおよびプログラム情報』

485 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』

486 ページの『例: プログラム・プロローグ域』

488 ページの『例: 静的マップ』

- 489 ページの『例: 定数域』
- 489 ページの『例: ベース・ロケーター・テーブル』
- 490 ページの『例: 外部シンボル』
- 491 ページの『例: DSA メモリー・マップ (自動マップ)』

関連参照
 『シグニチャー情報バイト』
 469 ページの『例: MAP 出力』
 言語環境プログラム 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

このトピックの表には、LIST コンパイラー・オプション使用時に提供されるプログラム初期化コードのリストの一部である、プログラム・シグニチャー情報が示されています。

表 53. **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	7
		8
		9
		10
		11
		12
03	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 26-31 はコンパイラー・オプションを参照しています。
- シグニチャー・バイト 6-7 は DATA DIVISION 項目を参照しています。
- シグニチャー・バイト 8 は ENVIRONMENT DIVISION 項目を参照しています。
- 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しています。

表 54. 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	項目	
			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	プログラムで RANDOM 関数を使用	
		2	プログラムで NATIONAL データ (Unicode) を使用	
		3	スキーマ妥当性検査による XML PARSE	
		4	STGOPT	NOSTGOPT
		5	AFP(VOLATILE)	AFP(NOVOLATILE)
		6	HGPR(PRESERVE)	HGPR(NOPRESERVE)
		7	NOTEST(DWARF)	NOTEST(DWARF) ではない
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	ZONEDATA(MIG)	ZONEDATA(MIG) ではない
		5	ZONEDATA(NOPFD)	ZONEDATA(NOPFD) ではない
07	31	0	NUMCHECK(ZON[(ALPHNUM)])	NUMCHECK(ZON[(ALPHNUM)]) ではない
		1	NUMCHECK(PAC)	Not NUMCHECK(PAC)
		2	NUMCHECK(BIN)	NUMCHECK(BIN) ではない
				ビット 0、1、および 2 がオフであれば NONUMCHECK は有効です
		3	NUMCHECK(ABD)	NUMCHECK(MSG) (ビット 0、1、または 2 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(NOALPHNUM)) ではない

表 54. 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	項目	
			オン	オフ
08	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
09	2	0	LIB (常にオン)	
		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
10	3	0	NUMPROC(PFD)	NUMPROC(NOPFD)
		1	RENT	NORENT
		2	RESIDENT (常にオン)	
		3	SEQUENCE	NOSEQUENCE
		4	予約済み	
		5	SOURCE	NOSOURCE
		6	NOSSRANGE ではない	NOSSRANGE
		7	TERM	NOTERM
11	4	0	TEST	NOTEST
		1	TRUNC(STD)	TRUNC(STD) ではない
		2	WORD	NOWORD
		3	VBREF	NOVBREF
		4	XREF	NOXREF
		5	ZWB	NOZWB
		6	NAME	NONAME
		7		NOCMPR2 (常にオフ)

表 54. 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	項目	
			オン	オフ
12	5	0	予約済み	
		1	NUMCLS=ALT	NUMCLS=PRIM
		2	DBCS	NODBCS
		3	AWO	NOAWO
		4	TRUNC(BIN)	TRUNC(BIN) ではない
		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	データ記述記入項目の OCCURS DEPENDING ON 節	
14	7	0	データ記述記入項目の SYNCHRONIZED 節	
		1	データ記述記入項目の JUSTIFIED 節	
		2	USAGE IS POINTER 項目	
		3	複合 OCCURS DEPENDING ON 節	
		4	DATA DIVISION の外部浮動小数点項目	
		5	DATA DIVISION の内部浮動小数点項目	
		6	行順次ファイル	
		7	USAGE IS PROCEDURE-POINTER 項目または 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で使用される方式を示すビット。詳しくは、「z/OS Language Environment Vendor Interfaces」の『COBOL-specific vendor interfaces』を参照。	
		4	VSAMOPENFS	
16	9	0	ACCEPT	
		1	ADD	
		2	ALTER	
		3	CALL	
		4	CANCEL	
		6	CLOSE	

表 54. 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	項目	
			オン	オフ
17	10	0	COMPUTE	
		2	DELETE	
		4	DISPLAY	
		5	DIVIDE	
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 54. 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	項目	
			オン	オフ
22	15	0	USE	
		1	WRITE	
		2	CONTINUE	
		3	END-ADD	
		4	END-CALL	
		5	END-COMPUTE	
		6	END-DELETE	
		7	END-DIVIDE	
23	16	0	END-EVALUATE	
		1	END-IF	
		2	END-MULTIPLY	
		3	END-READ	
		4	END-RETURN	
		5	END-REWRITE	
		6	END-SEARCH	
		7	END-START	
24	17	0	END-STRING	
		1	END-SUBTRACT	
		2	END-UNSTRING	
		3	END-WRITE	
		4	GOBACK	
		5	EVALUATE	
		7	SERVICE	
25	18	0	END-INVOKE	
		1	END-EXEC	
		2	XML	
		3	END-XML	
		4	ALLOCATE	
		5	FREE	
		6	JSON	
		7	END-JSON	
26	19	0 から 7	予約済み	
27	20	0 から 7	予約済み	

表 54. 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	項目	
			オン	オフ
28	21	0	16 進数リテラル	
		1	更新 GO TO	
		2	I-O ERROR 宣言	
		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 ID	
		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	
30	23	7	使用されている浮動小数点項目	
		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)
40	24	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	
		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 のどちらか)	
		5	COMMON	
		6	SELECT . . . OPTIONAL	
		7	EXTERNAL	

表 54. 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	項目	
			オン	オフ
41	25	0	GLOBAL	
		1	RECORD IS VARYING	
		2	VOLATILE	
		5	組み込み関数を使用した	
		6	z リテラルが検出された	
		7	RECURSIVE	
42	26	0	RMODE(ANY)	RMODE(ANY) ではない
		1-3	予約済み	
		4	予約済み	
		5	INTDATE(LILIAN)	INTDATE(ANSI)
		6	TEST NOTEST(SEPARATE)	TEST NOTEST(NOSEPARATE)
		7	TEST NOTEST(SEPARATE(DSNAME))	TEST NOTEST(SEPARATE(NODSNAME))
43	27	0	PGMNAME(LONGUPPER)	PGMNAME(LONGUPPER) ではない
		1	PGMNAME(LONGMIXED)	PGMNAME(LONGMIXED) ではない
		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	

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

387 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 V4 以前の初期化コードに含まれていた PROGRAM-ID、COMPILED TIME、COMPILED DATE などの情報は、Enterprise COBOL V5 以降の初期化コードには含まれていないため、そのコードが依存するプログラム

は、Enterprise COBOL V5 および V6 では異なる動作をす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。

(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	L3282: 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)
000020	07FF	000003	BR	R15
000022	0000	000003	DC	X'0000'
000024		000003	L3284: EQU	*
000024	0000 0000	000003	DC	X'00000000'
000028	0000 0000	000003	DC	X'00000000'
00002C	0000 8A0C	000003	DC	X'00008A0C'
000030	0000 8948	000003	DC	X'00008948'
000034	0000 0054	000003	DC	X'00000054'
000038	0000 0000	000003	DC	X'00000000'
00003C		000003	L3285: EQU	*
00003C	0000 0024	000003	DC	X'00000024'
000040	0000 8A1C	000003	DC	X'00008A1C'
000044		000003	L3280: EQU	*
000044	58F0 C31C	000003	L	R15,796(,R12)
000048	184E	000003	LR	R4,R14
00004A	05EF	000003	BALR	R14,R15
00004C	0000 0000	000003	DC	X'00000000'
000050	A7F4 0009	000003	J	L3281
000054		000003	MAINENT DS	0H
000054		000003	L3283: EQU	*
000054	18EF	000003	LR	R14,R15
000056	4100 E978	000003	LA	R0,2424(,R14)
00005A	5500 C314	000003	CL	R0,788(,R12)
00005E	A724 FFF3	000003	JH	L3280
000062		000003	L3281: EQU	*
000062	5000 E04C	000003	ST	R0,76(,R14)
000066	A708 0010	000003	LHI	R0,16
00006A	8900 0010	000003	SLL	R0,16
00006E	A70A 0301	000003	AHI	R0,769
000072	5000 E000	000003	ST	R0,0(,R14)
000076	5000 E004	000003	ST	R13,4(,R14)
00007A	18DE	000003	LR	R13,R14
00007C	4100 D600	000003	LA	R0,1744(,R13)
000080	5000 D074	000003	ST	R0,116(,R13)
000084	4100 0000	000003	LA	R0,0(,R0)
000088	5000 D070	000003	ST	R0,112(,R13)
				# Skip over constant area
				# Eyecatcher: CEE
				# Stack Size
				# Offset to PPA1
				# Wrong Entry Point: cause exception
				# Save GPRs Used
				# Args for boot strap routine
				#
				# Branch to boot strap routine
				# Available half-word
				# Boot Strap Info Block
				# address of entry label
				# WSA24 allocation size
				# address of Saved Option String
				# address of entry point name
				# A(Label L3283)
				# address of boot strap routine(IGZXBST)
				# CEE Parameter Block
				# address of infoBlockLabel
				# A(PARMCEE-CEEPPARMBlock)
				# Handle growing stack
				# Load CEECAA0GETS
				# Required NAB
				# Extend Stack
				# Argument list size = 0
				# Branch back
				# PRIMARY ENTRY POINT ADDRESS
				# User Code Entry Point
				# Load NAB
				# New NAB Address
				# Exceed current storage segment?
				# Yes: branch to recovery code
				# Stack now has sufficient room
				# Update NAB
				# COBOL Language Word upper half
				# shift to upper half of register
				# add COBOL Language Word lower half
				# Save Language Word
				# Save Back Chain
				# Set new DSA
				# Address of COBOSACB
				# Saved in member slot1
				#
				# 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

関連参照

476 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: TEST オプションが指定されている場合、または z/OS UNIX で -g が cob2 とともに指定されている場合に限り、MD5 シグニチャーが表示されます。

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

486 ページの『例: プログラム・プロログ域』

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 F1F3          =C'2017'          Compiled Year
0029CC F0F3 F2F7          =C'0717'          Compiled Date MMDD
0029D0 F1F2 F3F1 F2F2    =C'123122'        Compiled Time HHMMSS
0029D6 F0F5 F0F1 F0F0    =C'060200'        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 09/08/2017 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'P170724'      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-master-file.
```

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-MASTER-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 (コンパイラー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

473 ページの『LIST および MAP 出力で使用する記号』

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。

コンパイラー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詳しくは、『z/OS Language Environment Vendor Interfaces』を参照してください。

```

DATA VALIDATION AND UPDATE PROGRAM      IGYTCARA Date 09/08/2017 Time 10:48:16

      1      2      3      4
      PPA1: Entry Point Constants
0081E0 1CCEA506      =F'483304710'      Flags
0081E4 00008310      =A(PPA2-IGYTCARA)
0081E8 00008378      =A(PPA3-IGYTCARA)
0081EC 00000000      =F'0'
0081F0 FFFE0000      =F'-131072'      No EPD
0081F4 40000000      =F'1073741824'      Register Save Mask
0081F8 90            =AL1(144)      Member Flags
0081F9 000978        =AL3(2424)      Flags
0081FC 0000          =AL1(0)      Callee's DSA use/8
0081FE 0012          =H'18'      Flags
008200 00000600      =F'-805304624'      Offset/2 to CDL
008204 00000000      =F'0'      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 epilg
008218 0000 ***** AL2(8),C'IGYTCARA'      CDL end
      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。PPA2 には、PPA4 セクションに対するオフセットとともに、リストの Timestamp and Version Information セクションに対するオフセットが含まれています。

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(CCEESTART-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 FFFFFFF700    =A(CCEESTART-PPA2)
000838 00000058      =F'88'            A(PPA4-PPA2)
00083C FFFFFFFB00    =A(TIMESTAMP-PPA2)
000840 FFFFFFF700    =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 00000008      =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。ここには、書き込み可能静的域 (Static Map セクションおよび WSA24 セクション) など、コンパイラーによって生成された各種テーブルに対するオフセットが入っています。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 00000760      =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 UsrWrkStrg
000748 00000000      =X'0'           Length UsrWrkStrg
00074C 00           =X'0'           Has Externals
00074D 0000          =X'0'           A(SYSDBUGName-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 変更
(Enterprise COBOL for z/OS 移行ガイド)
z/OS Language Environment Vendor Interfaces

例: 静的マップ

リストにある 3 つのマップ・セクション (STATIC MAP、WORKING-STORAGE MAP、および WSA 24 MAP) はまとめてプログラムの静的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 STATIC 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 STATIC 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。

***** S T A T I C M A P *****		
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 WSA 24 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、STATIC MAP セクションも表示されます。

***** W S A 2 4 M A P *****		
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。

コンパイラーは、Constant Area の開始アドレスをロードし、固定オフセットをそれぞれの定数またはリテラルに追加することによって、Constant Area からロードを生成し、Constant Area へ格納します。

(1) (2)		CONSTANT AREA:		(3)	(4)
006A98 (+0)	00CCDDFF 00000000 C9C7E8E3 C3C1D9C1 00000000 00000000 C9C7E9E2 D9E3C3C4			IGYTCARA.....IGZSRICD
006AB8 (+32)	40000A00 40000000 00000008 00000000 E2E8E2D6 E4E34040 00100000 00000000			SYSOUT
006AD8 (+64)	0E000000 00000001 0F000000 0000001E 00000000 40000000 00000003 0064003C			TRANSACTION CODE.....
006AF8 (+96)	000FE800 9F0F0000 00000011 00000000 E3D9C1D5 E2C1C3E3 4B40C3D6 C4C50000			SHIFT CODE ..HOME LOC
006B18 (+128)	0000000E 00000000 E2C8C9C6 E340C3D6 C4C54040 40400000 C8D6D4C5 40D3D6C3				CODE..WORK LOC. CODE..LAST NAM
006B38 (+160)	4B40C3D6 C4C50000 E6D6D9D2 40D3D6C3 4B40C3D6 C4C50000 D3C1E2E3 40D5C1D4				E ..INITIALS ..DUPLICAT
006B58 (+192)	C5404040 40400000 C9D5C9E3 C9C1D3E2 40404040 40400000 C4E4D7D3 C9C3C1E3				E REC...REC. NOT FOUND..ADDRESS
006B78 (+224)	C540D9C5 C34B0000 D9C5C34B 40D5D6E3 40C6D6E4 D5C40000 C1C4C4D9 C5E2E240				..CITY ..STATE CO
006B98 (+256)	40404040 40400000 C3C9E3E8 40404040 40404040 40400000 E2E3C1E3 C540C3D6				DE ..ZIPCODE ..HOME PHO
006BB8 (+288)	C4C54040 40400000 E9C9D7C3 D6C4C540 40404040 40400000 C8D6D4C5 40D7C8D6				NE ..WORK PHONE ..HOME JUN
006BD8 (+320)	D5C54040 40400000 E6D6D9D2 40D7C8D6 D5C54040 40400000 C8D6D4C5 40D1E4D5				CTION ..WORK JUNCTION ..DRIVING
006BF8 (+352)	C3E3C9D6 D5400000 E6D6D9D2 40D1E4D5 C3E3C9D6 D5400000 C4D9C9E5 C9D5C740				STATUS.. REPORT #: IGYTCARA
006C18 (+384)	E2E3C1E3 E4E20000 40D9C5D7 D6D9E340 407B7A40 C9C7E8E3 C3C1D9C1 40404040			
006C38 (+416)	40404040 40404040 40404040 40404040 40404040 40404040 40404000 00000033				COMMUTER FILE UPDATE LIST
006C58 (+448)	C3D6D4D4 E4E3C5D9 40C6C9D3 C540E4D7 C4C1E3C5 40D3C9E2 E3404040 40404040			
006C78 (+480)	40404040 40404040 40404040 40404040 40400000 00000032 40404040 40404040				PAGE #: PROGRAM #: IGYT
006C98 (+512)	D7C1C7C5 407B7A40 00000000 00000010 40D7D9D6 C7D9C1D4 407B7A40 C9C7E8E3				CARA RUN TIME:
006CB8 (+544)	C3C1D9C1 40404040 404040D9 E4D540E3 C9D4C57A 40000000 00000025 7A000000			RUN DATE:/.....
006CD8 (+576)	00000030 00000000 D9E4D540 C4C1E3C5 7A400000 0000000A 61000000 00000008				

- (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。

関連参照

z/OS Language Environment Vendor Interfaces (ベース・ロケーター・テーブル)

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・ロケーター・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 (自動マップ)

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。

```

* * * * * A U T O M A T I C   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。

An "M" preceding a data-name reference indicates that the data-name is modified by this reference.

(1) 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

(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-MASTER-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

データ名の相互参照:

- (1) その名前が定義されている行番号。
- (2) データ名。
- (3) その名前が使用されている行番号。M が行番号の前に置かれている場合は、データ項目がその位置で明示的に変更されたことを意味します。

プロシージャ参照の相互参照:

- (4) プロシージャ参照のコンテキスト取扱コードの説明。
- (5) そのプロシージャ名が定義されている行番号。
- (6) プロシージャ名。
- (7) そのプロシージャが参照されている行番号およびそのプロシージャのコンテキスト取扱コード。

『例: XREF 出力: プログラム名相互参照』

493 ページの『例: XREF 出力: COPY/BASIS 相互参照』

494 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ス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、z/OS の XREF コンパイラー・オプションで生成された関連コピーブックのライブラリー名およびデータ・セット名に対するコピーブックのソート済み相互参照を示しています。括弧内の番号は、後続の注釈に対応しています。

COPY/BASIS cross-reference of text-names, library names

(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	DDNAME	(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: z/OSでは、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。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 UNIX 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) ソースの COPY ステートメントからのものです。例えば、上の相互参照の 3 番目の項目に対応する COPY ステートメントは次のようなものです。
- COPY '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

437 ページの『XREF』

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-master-file.	932 (1)
000882			perform 100-initialize-paragraph.	984
000883			read update-transaction-file into ws-transaction-record	204 340
000884			at end	
000885	1		set transaction-eof to true	254
000886			end-read.	
. . .				
000984			100-initialize-paragraph.	
000985			move spaces to ws-transaction-record	IMP 340 (2)
000986			move spaces to ws-commuter-record	IMP 316
000987			move zeroes to commuter-zipcode	IMP 327
000988			move zeroes to commuter-home-phone	IMP 328
000989			move zeroes to commuter-work-phone	IMP 329
000990			move zeroes to commuter-update-date	IMP 333
000991			open input update-transaction-file	204
000992			location-file	193
000993			i-o commuter-file	181
000994			output print-file	217
. . .				

```

001442          1100-print-i-f-headings.
001443
001444          open output print-file.                217
001445
001446          move function when-compiled to when-comp.      IFN 698 (2)
001447          move when-comp (5:2) to compile-month.        698 640
001448          move when-comp (7:2) to compile-day.          698 642
001449          move when-comp (3:2) to compile-year.          698 644
001450
001451          move function current-date (5:2) to current-month.      IFN 649
001452          move function current-date (7:2) to current-day.      IFN 651
001453          move function current-date (3:2) to current-year.      IFN 653
001454
001455          write print-record from i-f-header-line-1          222 635
001456          after new-page.                                    138
. . .

```

- (1) プログラム内のデータ名またはプロシージャ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 09/08/2017 Time 10:48:16

```

. . .
(1)  (2)  (3)
LINE #  HEXLOC  VERB      LINE #  HEXLOC  VERB      LINE #  HEXLOC  VERB
000880  0026F0  DISPLAY    000881  002702  PERFORM    000933  002702  OPEN
000934  002722  IF          000935  00272C  DISPLAY    000936  002736  PERFORM
001389  002736  DISPLAY    001390  002740  DISPLAY    001391  00274A  DISPLAY
001392  002754  DISPLAY    001393  00275E  DISPLAY    001394  002768  DISPLAY
001395  002772  DISPLAY    000937  00277C  PERFORM    001434  00277C  DISPLAY
001435  002786  STOP       000939  0027A2  MOVE       000940  0027AC  WRITE
000941  0027D6  IF          000942  0027E0  DISPLAY    000943  0027EA  PERFORM
001389  0027EA  DISPLAY    001390  0027F4  DISPLAY    001391  0027FE  DISPLAY
001392  002808  DISPLAY    001393  002812  DISPLAY    001394  00281C  DISPLAY
001395  002826  DISPLAY    000944  002830  DISPLAY    000945  00283A  PERFORM
001403  00283A  DISPLAY    001404  002844  DISPLAY    001405  00284E  DISPLAY
001406  002858  DISPLAY    001407  002862  CALL       000947  002888  CLOSE

```

- (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) このステートメント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)。
ステートメントは発生順にリストされ、使用されるたびに一度ずつリストされます。
- (3) 使用されるステートメント。

注:

- 最適化プログラムは段落をインライン化するか、コードを周囲に移動させるか、またはあまり使用されないコード (エラー・メッセージ・フォーマット設定コードなど) であれば、そのコードを実際にはプログラム本文の後に置く場合があります

ます。これで、以前のコンパイラーで作成されたものよりも役立たない、OFFSET レポートが作成される可能性があります。代わりに、LIST 出力を調べてください (OFFSET および LIST は相互に排他的なオプションであることに注意してください)。詳しくは、474 ページの『LIST 出力の読み取り』を参照してください。

- エラー・メッセージ・フォーマット設定において調和しないコードが原因で、「From compile unit {name} at entry point {name} at compile unit offset {offset}...」に示されている 言語環境プログラム 生成のオフセットが、プログラムのオフセット範囲からはみ出してしまう可能性があります。このような場合、問題を見つけるためには、COBOL メッセージ (IGZnnnns) でステートメント番号を調べてください。

関連参照

399 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・ステートメントが入っているプログラムのリストを示しています。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。

```

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。

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抑止

TEST(DWARF) が有効になっている場合は、CEEDUMP 処理において、WORKING-STORAGE SECTION のサイズによってはダンプに大量の情報が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。

1. 言語環境プログラム・サンプル・データ・セット .SCEESAMP でサンプル JCL IGZ10PT を使用してロード・モジュール IGZUOPT を作成します。ジョブ・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 を作成します。
2. このモジュールを STEPLIB 連結内のデータ・セット内に配置します。CEEDUMP 処理中に DWARF 情報が抑止されるようになります。これにより、CEEDUMP で COBOL プログラムの出力の量が減る可能性があります。

JCL のジョブ STEP1 で、マクロ IGZXOPT を呼び出すアセンブラー・プログラムがアセンブルされます。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 SKIPDWARF オプションのみがサポートされています。

```
IGZXOPT SKIPDWARF=ON | OFF
```

SKIPDWARF の設定を ON または OFF にできます。デフォルト値は OFF です。

- ON が指定された場合、CEEDUMP での DWARF 処理は抑止されます。
- OFF が指定された場合、DWARF 処理は通常どおりに進行します。OFF を指定することは、STEPLIB から IGZUOPT を除外することと同じです。

関連参照

423 ページの『TEST』

第 3 部 特定の環境に合わせた **COBOL** プログラムの目標

第 20 章 COBOL プログラムの開発 (CICS の場合)

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 コンパイラー・オプションを使用すると、Enterprise COBOL コンパイラーは、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。

実行時エラーの処理方法を判別するには、CBLPSHPOP ランタイム・オプションを設定します。CICS の HANDLE と CBLPSHPOP については、関連タスクを参照してください。

関連概念

508 ページの『組み込みの CICS 変換プログラム』

関連タスク

『CICS のもとで実行する COBOL プログラムのコーディング』

506 ページの『CICS オプションを使用したコンパイル』

509 ページの『分離型の CICS 変換プログラムの使用』

512 ページの『CICS HANDLE を使用したエラー処理』

言語環境プログラム プログラミング・ガイド

(CICS 環境での条件処理: CBLPSHPOP ランタイム・オプションの使用)

CICS Application Programming Guide

関連参照

358 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

CICS 下で実行されるようにプログラムをコーディングするには、EXEC CICS コマンド・フォーマットを使用して CICS コマンドを PROCEDURE DIVISION 内にコーディングしてください。

```
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。EXEC CICS コマンド内に COBOL ステートメントをコーディングしないでください。

制約事項: CICS では、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が含まれる COBOL プログラムを実行することはできません。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。

- ENVIRONMENT DIVISION での FILE-CONTROL 記入項目 (FILE-CONTROL 記入項目が SORT ステートメントで使用される場合を除く)。
- 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 を使用できます)
 - CLOSE
 - DELETE
 - DISPLAY UPON CONSOLE
 - DISPLAY UPON SYSPUNCH
 - MERGE
 - OPEN
 - READ
 - RERUN
 - REWRITE
 - START
 - STOP *literal*
 - WRITE

分離型の CICS 変換プログラムを使用する予定である場合、EXEC コマンドを含んでいる REPLACE ステートメントはいずれも、プログラムの PROCEDURE DIVISION ヘッダーの後に配置する必要があります。そうしないとコマンドは変換されません。

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、呼び出しで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) コンパイラー・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) コンパイラー・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。

DL/I CALL ステートメントではなく EXEC DLI を使用する場合は、IMS 製品のレベルに関係なく、DATA(31) を指定することができます。

- 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。

データ項目の内容の表示: システム論理出力装置 (SYSOUT、SYSLIST、SYSLST) への DISPLAY は CICS のもとでサポートされます。DISPLAY 出力は、言語環境プログラムのメッセージ・ファイル (一時データ・キュー CESE) に書きこまれます。ただし、DISPLAY . . . UPON CONSOLE と DISPLAY . . . UPON SYSPUNCH はサポートされません。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

508 ページの『組み込みの CICS 変換プログラム』

関連タスク

268 ページの『CICS のもとでのソート』

『CICS のもとでのシステム日付の取得』

504 ページの『COBOL プログラムとの間の呼び出し』

506 ページの『ECI 呼び出しの成否の判断』

509 ページの『分離型の CICS 変換プログラムの使用』

関連参照

268 ページの『CICS SORT アプリケーションの制約事項』

370 ページの『DISPSIGN』

CICS のもとでのシステム日付の取得

CICS プログラムでシステム日付を検索するには、形式 2 の ACCEPT ステートメントまたは CURRENT-DATE 組み込み関数を使用してください。

以下の形式 2 の ACCEPT ステートメントを CICS で使用すると、システム日付を入手することができます。

- ACCEPT *identifier-2* FROM DATE (2 桁年)
- ACCEPT *identifier-2* FROM DATE YYYYMMDD
- ACCEPT *identifier-2* FROM DAY (2 桁年)
- ACCEPT *identifier-2* FROM DAY YYYYDDD
- ACCEPT *identifier-2* FROM DAY-OF-WEEK (1 桁の整数。1 は月曜日を表します。)

次に示す形式 2 の ACCEPT ステートメントを CICS で使用すると、システム時刻を入手することができます。

- ACCEPT *identifier-2* FROM TIME

あるいは、CURRENT-DATE 組み込み関数を使用できます。この組み込み関数も時刻を提供できます。

これらの方法は、CICS 環境と非 CICS 環境の両方で使用できます。

CICS プログラムでは、形式 1 の ACCEPT ステートメントは使用しないでください。

関連タスク

38 ページの『画面またはファイルからの入力割り当て (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

CALL を使用して、VS COBOL II、COBOL for MVS & VM、COBOL for OS/390 & VM、および Enterprise COBOL のプログラムとの間で呼び出しを行うことができます。

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 プログラムが EXEC CICS ステートメントを含んでいる場合、または EXEC SQL ステートメントを含んでいる場合、NODYNAM コンパイラー・オプションが必要です。この場合、CALL *identifier* を NODYNAM コンパイラー・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 プログラムが EXEC CICS ステートメントを含んでおらず、EXEC SQL ステートメントも含んでいない場合、NODYNAM を使用してコンパイルする必要はありません。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。CICS 環境では、COBOL プログラムは CICS PROGRAM 定義に RELOAD=YES オプションをコーディングするサブプログラムに対する動的呼び出しをサポートします。

ん。 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、代わりに CICS の LINK、XCTL、および RETURN を使用できます。

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、「言語環境プログラム 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プログラム準拠 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アセンブラー言語プログラムは、非 LE 準拠 アセンブラー・プログラムと呼んでいます。

表 55. 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	言語環境プログラム準拠アセンブラー・プログラム	非言語環境プログラム準拠アセンブラー・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、分離型の CICS 変換プログラムを使用する予定である場合、EXEC コマンドまたは EXEC インターフェース・ブロック (EIB) への参照を含んでいるネストされたプログラムへのパラメーターとして、DFHEIBLK および DFHCOMMAREA を渡してください。制御階層内で、このようなネストされたプログラムとその最上位のプログラムとの間にあるプログラムにも、同じパラメーターを渡す必要があります。

関連概念

508 ページの『組み込みの CICS 変換プログラム』

関連タスク

509 ページの『分離型の CICS 変換プログラムの使用』

527 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』

283 ページの『プログラム呼び出し時のエラーの処理』

言語環境プログラム ILC (言語間通信) アプリケーションの作成 (CICS のもとでの ILC)

関連参照

373 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。

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 特殊レジスタを設定します。

関連タスク

CICS External Interfaces Guide

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。

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。CICS オプションが COBOL インストール・デフォルトである場合、CICS サブオプションを指定することはできません。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。

Enterprise COBOL の規則に従い、CBL ステートメントや PROCESS ステートメントはすべて、コメント行より前に置く必要があります。

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 56. 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 プロシーチャー・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 V6R1 用のデータ・セットのデフォルト名は CICS61.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=CICS61.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

508 ページの『組み込みの CICS 変換プログラム』

関連タスク

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

508 ページの『CICS サブオプションの分離』

CICS Application Programming Guide

関連参照

358 ページの『CICS』

349 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。

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。

関連参照

358 ページの『CICS』

組み込みの CICS 変換プログラム

CICS コンパイラー・オプションを指定して COBOL プログラムをコンパイルする場合、COBOL コンパイラーは組み込みの CICS 変換プログラムと連動して、ソース・プログラム内のネイティブ COBOL ステートメントと組み込みの CICS ステートメントの両方を処理します。

コンパイラーは、CICS ステートメントを検出したとき、およびソース・プログラム内の重要な地点で、組み込みの CICS 変換プログラムとインターフェースをとります。EXEC CICS ステートメントと END-EXEC ステートメントの間のテキストはすべて変換プログラムに渡されます。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ーに制御を戻します。

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。

- デバッグ・ツール を使用して、分離型の CICS 変換プログラムによって生成される拡張ソースではなく、元のソースをデバッグすることができます。
- コピーブック内の EXEC CICS ステートメントや EXEC DLI ステートメントを個別に変換する必要がありません。
- 変換済みで未コンパイル・バージョンのソース・プログラムに対応する中間データ・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。
- EXEC CICS が含まれているネストされたプログラムの使用が簡単です。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これらを引数として指定する必要はなく、ネストされたプログラムの PROCEDURE DIVISION ヘッダーの USING 句で指定する必要もありません。
- EXEC CICS ステートメントが含まれているネストされたプログラムを個別のファイルに保管し、COPY ステートメントを使用して、それらのネストされたプログラムを組み込むことができます。
- REPLACE ステートメントを EXEC CICS ステートメントに影響させることができます。
- CICS ステートメントが含まれているプログラムをバッチ・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。
- CICS 制御ブロックの 2 進数フィールドは BINARY ではなく COMP-5 の形式で生成されるようになったので、TRUNC コンパイラー・オプションの設定との依存関係がありません。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: CICS の資料では、組み込みの CICS 変換プログラムでコンパイルされたプログラムに対して EXCI 変換プログラム・オプションはサポートされていないことになっていますが、CICS はこの立場を翻しました。現在は、EXCI 変換プログラム・オプションでコンパイルを実行でき、警告メッセージ DFH7006I を無視することができます。

関連概念

CICS Application Programming Guide (組み込みの CICS 変換プログラム)

関連タスク

501 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』

506 ページの『CICS オプションを使用したコンパイル』

関連参照

358 ページの『CICS』

430 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。

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 57. 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 58. 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。

例えば、分離型の CICS 変換プログラムを使用する場合、データ項目を、8 桁より大きな値を受け取る可能性のある PIC S9(8) BINARY として定義するときは、TRUNC(BIN) コンパイラー・オプションを使用するか、項目を USAGE COMP-5 に変更するか、PICTURE 節を変更してください。

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

508 ページの『組み込みの CICS 変換プログラム』

関連タスク

506 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。コンパイラー・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
- SD^{1, 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

506 ページの『CICS オプションを使用したコンパイル』

268 ページの『CICS のもとでのソート』

関連参照

435 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。

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。

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)。

CICS HANDLE CONDITION または CICS HANDLE AID コマンドを使用する場合は、CICS HANDLE コマンドで指定された LABEL が、CICS HANDLE ラベルへの分岐を引き起こす CICS コマンドと同じ PROCEDURE DIVISION に入っていなければなりません。

LABEL オプションを指定した CICS HANDLE コマンドでは、COBOL CALL ステートメントを使用して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、援助機能、または異常終了を処理することはできません。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と、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。

VS COBOL II ランタイムから言語環境プログラム・ランタイムにアプリケーションを移行する場合は、関連参照で CBLPSHPOP オプションについての追加考慮事項を参照してください。

『例: CICS HANDLE を使用したエラー処理』

関連タスク

793 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('MASTER')
                                INTO(RECORD)
                                RIDFLD(ACCTNO)
                                END-EXEC.
. . .
END PROGRAM B.

```

第 21 章 Db2 環境用のプログラミング

一般に、COBOL プログラムのコーディングは、プログラムから Db2 データベースにアクセスするかどうかに関係なく同じになります。しかし、Db2 データの検索、更新、挿入、および削除を行い、さらにその他の Db2 サービスを使用するためには、SQL ステートメントを使用しなければなりません。

Db2 と通信するには、以下のステップを実行します。

- EXEC SQL および END-EXEC ステートメントで区切って、必要なすべての SQL ステートメントをコーディングします。
- Db2 独立型プリコンパイラーを使用するか、あるいは 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

『Db2 コプロセッサ』

522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

516 ページの『分離型 Db2 プリコンパイラーの使用』

517 ページの『SQL ステートメントのコーディング』

520 ページの『SQL オプションを使用したコンパイル』

527 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

524 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。EXEC SQL ステートメントと END-EXEC ステートメントの間のテキストはすべて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と、以下のようになります。
 - 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

522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

『分離型 Db2 プリコンパイラーの使用』

520 ページの『SQL オプションを使用したコンパイル』

関連参照

524 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』

416 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。

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、分離型 Db2 プリコンパイラーは、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 Db2 コプロセッサを使用してください。

関連概念

515 ページの『Db2 コプロセッサ』

関連参照

524 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。EXEC SQL および END-EXEC 区切り文字はそれぞれ 1 行の中で完結している必要があります。複数行にわたって継続させることはできません。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。

さらに、以下の特別なステップを実行する必要があります。

- EXEC SQL INCLUDE ステートメントをコーディングして、最外部プログラムの WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION に SQL 通信域 (SQLCA) を組み込んでください。再帰的プログラムや THREAD コンパイラー・オプションを使用するプログラムの場合には、LOCAL-STORAGE をお勧めします。
- SQL ステートメントで使用するすべてのホスト変数を WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、または LINKAGE SECTION に定義する。ただし、EXEC SQL BEGIN DECLARE SECTION および EXEC SQL END DECLARE SECTION を指定する必要はありません。

制約事項: オブジェクト指向クラスまたはメソッドで SQL ステートメントを使用することはできません。

関連タスク

『Db2 コプロセッサを用いた SQL INCLUDE の使用』

518 ページの『SQL ステートメントでの文字データの使用』

519 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』

519 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』

519 ページの『SQL ステートメントでのバイナリー項目の使用』

520 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

523 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 コンパイラー・オプションを使用すると、SQL INCLUDE ステートメントは、ネイティブの COBOL COPY ステートメントとまったく同様に扱われます。

したがって、次の 2 行は同様に扱われます。(EXEC SQL INCLUDE ステートメントの終了を示すピリオドが必要です。)

```
EXEC SQL INCLUDE name END-EXEC.  
COPY "name".
```

SQL INCLUDE ステートメント内の *name* 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 INCLUDE ステートメントのライブラリー探索順序は、ライブラリー名を指定しない COBOL COPY ステートメントを解決するためにコンパイラーが使用するのと同じ SYSLIB 連結です。

関連参照

445 ページの『第 18 章 コンパイラー指示ステートメント』

524 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』

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

EXEC SQL ステートメントで使用する文字データのホスト変数を記述するのに、以下の USAGE 節のいずれかをコーディングできます。すなわち、1 バイトまたは UTF-8 データの場合は USAGE DISPLAY、DBCS データの場合は USAGE DISPLAY-1、または UTF-16 データの場合は USAGE NATIONAL です。

独立型 Db2 プリコンパイラーを使用する場合、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。USAGE DISPLAY または DISPLAY-1 で宣言されたホスト変数のコード・ページを指定しなければならないのは、COBOL CODEPAGE コンパイラー・オプションで有効な CCSID が、文字および図形データ用に Db2 が使用する CCSID と一致しない場合のみです。

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 コンパイラー・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.
```

EXEC SQL DECLARE *variable-name* VARIABLE CCSID *nnnn* END-EXEC を指定した場合は、その指定によって暗黙の CCSID がオーバーライドされます。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

522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

516 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

524 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』
359 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。

EXEC SQL ステートメントで指定する国別 10 進数ホスト変数はいずれも次のような特性を持っている必要があります。

- 符号付きでなければなりません。
- SIGN LEADING SEPARATE 節で指定する必要があります。
- 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

54 ページの『数値データの形式』

関連タスク

151 ページの『国別数値データ項目の定義』

関連参照

524 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。

国別グループ内のすべての従属項目は USAGE NATIONAL を持っている必要があるので、国別グループ項目では可変長ストリングを記述できません。

関連タスク

152 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

EXEC SQL ステートメントで指定するバイナリー・データ項目の場合は、USAGE COMP-5 としてか、USAGE BINARY、COMP、または COMP-4 として定義することができます。

バイナリー・データ項目を USAGE BINARY、COMP、または COMP-4 として定義する場合は、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場合があります。) 代わりに、TRUNC(OPT) または TRUNC(STD) が有効な場合は、コンパイラーはその項目を受け入れますが、10 進数切り捨て規則のため、そのデータは無効なことがあります。切り捨てがデータの妥当性に影響を与えないようにしなければなりません。

関連概念

54 ページの『数値データの形式』

関連参照

430 ページの『TRUNC』

SQL ステートメントの成否の判断

Db2 は、SQL ステートメントの実行を終了すると、戻りコードを SQLCA 構造体に入れて送り (例外が 1 つあります)、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。

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使用された後で 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 コンパイラー・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。

SQL 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、OPTFILE、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。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 コンパイラー・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。上記のコンパイラー・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。

- 生成済みデータベース要求モジュール (DBRM) の位置を指定した DBRMLIB DD 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 xxxxx.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、以下の行が JCL に含まれる場合があります。

```
//DBRMLIB DD DSN=PAYROLL.MONTHLY.DBRMLIB.DATA(MASTER),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だけに有効である必要があります。コンパイラーの呼び出しで 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、コンパイラー診断メッセージを受け取ります。

関連概念

515 ページの『Db2 コプロセッサ』

522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

『Db2 サブオプションの分離』

527 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

373 ページの『DYNAM』

416 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。

サブオプション・ストリングに組み込まれるオプションは累積されます。コンパイラーは、複数のソースからこれらのサブオプションを、これらのサブオプションが指定された順序で連結します。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、連結ストリング内の当該サブオプションの最後の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。

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

515 ページの『Db2 コプロセッサ』

関連タスク

523 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

『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』

359 ページの『CODEPAGE』

416 ページの『SQL』

417 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。次に例を示します。

```
01 hostvariable pic n(10) usage national.
```

- 明示的な FOR BIT DATA 宣言を持つ英数字ホスト変数は、Db2 によって CCSID 66535 に設定されます。これは、変数がエンコードされた文字を表さないことを示します。次に例を示します。

```
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。次に例を示します。

```
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 ホスト変数は、SQLCCSID オプションが有効であると、マップされた値 *mmmmmm* で処理されます。これは、CODEPAGE(*nnnnn*) コンパイラー・オプションからの混合 (MBCS) CCSID *nnnnn* の純粋 DBCS CCSID コンポーネントです。
- 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

359 ページの『CODEPAGE』

417 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。既存のプリコンパイラー・ベ

ースのアプリケーションを、統合 Db2 コプロセッサを使用するように移行するためのメカニズムとしては、NOSQLCCSID オプションが推奨されます。

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

Db2 プリコンパイラーの動作との最高の互換性を必要とするアプリケーションの場合は、NOSQLCCSID オプションが推奨されます。

COBOL 英数字データ項目を、FOR BIT DATA サブタイプで定義される Db2 スtring・データとやり取りするホスト変数として使用する場合は、次のいずれかを行う必要があります。

- 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

- Db2 DCLGEN コマンドを使用してテーブルの COBOL 宣言を生成する場合は、必要なら、FOR BIT DATA 宣言を自動的に作成させることもできます。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

515 ページの『Db2 コプロセッサ』

関連タスク

516 ページの『分離型 Db2 プリコンパイラーの使用』

関連参照

417 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、522 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

プリコンパイラー: Db2 プリコンパイラーでは、各 EXEC SQL INCLUDE ステートメントをピリオドで終了する必要がありません。ピリオドが指定されている場合、プリコンパイラーはそれ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 コプロセッサでは、各 EXEC SQL INCLUDE ステートメントが COPY ステートメントのように扱われるため、ピリオドでステートメントを終了する必要があります。以下に例を示します。

```
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

プリコンパイラー: Db2 プリコンパイラーを使用して、COBOL REPLACE ステートメントおよび COPY ステートメントの REPLACING 句は、EXEC SQL ステートメントで作成された拡張ソースに作用します。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
```

プリコンパイラーでは、G.ABC に対する参照は拡張ソース内で ABC of G として示され、XYZ of G に置き換えられます。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

プリコンパイラー: Db2 プリコンパイラーは、同じ行で END-EXEC ステートメントより後にあるコードをすべて無視します。

コプロセッサ: Db2 コプロセッサは、同じ行で END-EXEC ステートメントより後にあるコードを処理します。

ホスト変数の複数定義

プリコンパイラー: Db2 プリコンパイラーでは、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、 408 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: Db2 プリコンパイラでは、EXEC SQL ステートメントの開始桁が 12 から 72 の間でなければなりません。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: Db2 コプロセッサでは、継続行も含め EXEC SQL ステートメントのすべての行を 12 から 72 桁目の間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: Db2 プリコンパイラでは、COBOL の英数字データ項目を、サブタイプ FOR BIT DATA を持つ Db2 文字データを保持するホスト変数として使用することができます。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: Db2 コプロセッサでは、COBOL の英数字データ項目を、サブタイプ FOR BIT DATA を持つ Db2 文字データを保持するホスト変数として使用できますが、その場合は、COBOL プログラム内でそのホスト変数に対して明示の EXEC SQL DECLARE VARIABLE ステートメントが指定されていなければなりません。以下に例を示します。

```
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・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: Db2 プリコンパイラでは、プログラムが複数回呼び出されたときに異なるアドレスに存在する可能性のあるホスト変数を渡す場合は、呼び出し先プログラムが SQL-INIT-FLAG をリセットする必要があります。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: Db2 コプロセッサでは、呼び出し先プログラムが SQL-INIT-FLAG をリセットする必要はありません。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。ただし、SQL-INIT-FLAG を変更するステートメント (MOVE ZERO TO SQL-INIT-FLAG など) は、プログラム内の SQL 処理に影響を及ぼしません。

関連概念

- 515 ページの『Db2 コプロセッサ』
- 522 ページの『COBOL および Db2 CCSID の決定』

関連タスク

- 516 ページの『分離型 Db2 プリコンパイラーの使用』

関連参照

- 523 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』
- 417 ページの『SQLCCSID』

DYNAM または NODYNAM コンパイラー・オプションの選択

EXEC SQL ステートメントを含んでいる COBOL プログラムの場合、コンパイラー・オプションの DYNAM を選ぶか NODYNAM を選ぶかは、稼働環境によって異なります。

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、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。

ストアード・プロシージャでは CAF を使用するので、COBOL ストアード・プロシージャも 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

- 520 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

- 373 ページの『DYNAM』

第 22 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。

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
- 言語環境プログラム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いくつかあります。

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

『IMS SQL コプロセッサ』

関連タスク

530 ページの『SQLIMS ステートメントのコーディング』

532 ページの『SQLIMS オプションを使用したコンパイル』

534 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』

535 ページの『IMS のもとでのオブジェクト指向 COBOL と Java の使用』

535 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』

536 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』

537 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。EXEC SQLIMS ステートメントと END-EXEC ステートメントの間のテキストはすべて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 プログラムには、Db2 SQL データベースにアクセスするための EXEC SQL ステートメントと、IMS DLI データベースにアクセスするための EXEC SQLIMS ステートメントの一方または両方が含まれている可能性があります。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、以下のように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する。

関連タスク

『SQLIMS ステートメントのコーディング』

532 ページの『SQLIMS オプションを使用したコンパイル』

534 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

418 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。EXEC SQLIMS 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。

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

531 ページの『IMS SQL コプロセッサで SQLIMS INCLUDE を使用』

531 ページの『SQLIMS ステートメントでの文字データの使用』

531 ページの『SQLIMS ステートメントでのバイナリー項目の使用』

532 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。

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 INCLUDE ステートメントのライブラリー探索順序は、ライブラリー名を指定しない COBOL COPY ステートメントを解決するためにコンパイラーが使用するのと同じ SYSLIB 連結です。

関連参照

445 ページの『第 18 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。

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

529 ページの『IMS SQL コプロセッサ』

関連参照

359 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

EXEC SQLIMS ステートメントで指定するバイナリー・データ項目の場合は、USAGE COMP-5 としてか、USAGE BINARY、COMP、または COMP-4 として定義することができます。

バイナリー・データ項目を USAGE BINARY、COMP、または COMP-4 として定義する場合は、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

54 ページの『数値データの形式』

SQLIMS ステートメントの成否の判断

IMS は SQLIMS ステートメントの実行を終了すると、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。

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。

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、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いければなりません。

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。上記のコンパイラー・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 LNKST 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。コンパイラーの呼び出しで 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、コンパイラー診断メッセージが発行されます。

関連概念

529 ページの『IMS SQL コプロセッサ』

関連タスク

『IMS サブオプションの分離』

関連参照

416 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。

サブオプション・ストリングに組み込まれるオプションは累積されます。コンパイラーは、複数のソースからこれらのサブオプションを、これらのサブオプションが指定された順序で連結します。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、コンパイラー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。コンパイラーが同じ 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。コンパイラーは、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

529 ページの『IMS SQL コプロセッサ』

関連タスク

532 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。

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。さらに、COBOL プログラムを含むプログラム・オブジェクトをプリロードするときは、そのプログラム・オブジェクト内のすべての 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なければなりません。

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。

16 MB 境界より上で実行するには、RENT を指定してアプリケーション・プログラムをコンパイルする必要があります。IMS アプリケーション・プログラムのデータは 16 MB 境界より上に常駐させることができ、IMS サービスを使用するプログラムでは DATA(31) RENT を使用できます。

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

43 ページの『ストレージとそのアドレス可能性』

関連タスク

527 ページの『DYNAM または NODYNAM コンパイラー・オプションの選択』
言語環境プログラム プログラミング・ガイド (IMS での条件処理)

関連参照

366 ページの『DATA』

409 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。

例えば、次のようなことが可能です。

- 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

728 ページの『ファクトリー・セクションの定義』

693 ページの『第 33 章 オブジェクト指向プログラムの作成』

743 ページの『第 34 章 Java メソッドとの通信』

333 ページの『第 16 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。

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。

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. マスター 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。

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

333 ページの『第 16 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』

739 ページの『OO アプリケーションの構造化』

738 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

Java 従属領域で稼働するアプリケーションは、クラスの main メソッドで開始する必要があります。

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 は、main メソッドを備えた Java クラスを生成します。Java 従属領域は、このメソッドを検索し、インスタンス化し、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、メッセージの処理 (GU および GN) やトランザクションの同期 (CHKP) のために、DL/I 呼び出しを使用する必要があります。

関連タスク

739 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。

関連タスク

『STOP RUN ステートメントの使用』

『メッセージの処理とトランザクションの同期』

『データベースへのアクセス』

538 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。

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。

COBOL コンポーネントでは、メッセージ処理用 (GU と GN) およびトランザクション同期用 (CHKP) の DL/I サービスに対して、CALL ステートメントを使用します。Java コンポーネントは、IMS 用の Java クラスを使用して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。

制約事項: Java 従属領域で稼働する COBOL ルーチンでは、Db2 データベース・アクセス用の EXEC SQL ステートメントはサポートされていません。

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。

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) 定義内のすべての PCB に名前が付いている必要があります。) 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 リスト内の PCB の相対的な位置を知る必要はありません。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。)

『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 "DFSAIB" 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 23 章 z/OS UNIX のもとでの COBOL プログラムの実行

z/OS UNIX 環境で COBOL プログラムを実行するには、Enterprise COBOL または COBOL for OS/390 & VM を使用して COBOL 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。

z/OS UNIX ファイル・システム からプログラムを実行する予定であれば、リンカー・オプション AMODE 31 を使用してください。z/OS UNIX アプリケーション内から呼び出す AMODE 24 プログラムは、MVS PDSE に常駐させる必要があります。

制約事項: z/OS UNIX 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。
- NOTHREAD 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。NOTHREAD 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 Developer for z Systems 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 UNIX プログラムをデバッグできます。

関連タスク

- 321 ページの『第 15 章 z/OS UNIX のもとでのコンパイル』
 - 336 ページの『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実行』
 - 542 ページの『z/OS UNIX 環境での実行』
 - 542 ページの『環境変数の設定およびアクセス』
 - 545 ページの『UNIX/POSIX API の呼び出し』
 - 547 ページの『z/OS UNIX 下でのメインプログラム・パラメーターへのアクセス』
- 言語環境プログラム プログラミング・ガイド

関連参照

- 409 ページの『RENT』

z/OS UNIX 環境での実行

COBOL プログラムは、z/OS UNIX シェルまたはシェル以外から、z/OS UNIX 実行環境のいずれかで実行できます。

- 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。以下に例を示します。

```
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 UNIX ファイル に送信します。

- シェル以外からは、TSO/E またはバッチでプログラムを実行できます。

z/OS UNIX 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。

z/OS UNIX ファイル に常駐する COBOL プログラムを EXEC JCL ステートメントを使用して呼び出すには、BPXBATCH ユーティリティを使用してください。

関連タスク

336 ページの『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実行』
『環境変数の設定およびアクセス』

545 ページの『UNIX/POSIX API の呼び出し』

547 ページの『z/OS UNIX 下でのメインプログラム・パラメーターへのアクセス』

197 ページの『QSAM ファイルの定義および割り振り』

243 ページの『行順次ファイルの割り振り』

233 ページの『VSAM ファイルの割り振り』

39 ページの『画面上またはファイル内での値の表示 (DISPLAY)』

言語環境プログラム 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

423 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)
言語環境プログラム 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

z/OS UNIX COBOL プログラムの環境変数は、`export` および `set` を使用してシェルから、またはプログラムから、設定できます。

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状態にリセットするには、z/OS UNIX シェル・コマンド `unset` を使用してください。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います。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。

544 ページの『例: 環境変数の設定とアクセス』

関連タスク

542 ページの『z/OS UNIX 環境での実行』

『実行に影響を与える環境変数の設定』

547 ページの『z/OS UNIX 下でのメインプログラム・パラメーターへのアクセス』

336 ページの『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実行』

321 ページの『z/OS UNIX のもとでの環境変数の設定』

関連参照

『ランタイム環境変数』

言語環境プログラム 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

実行に影響を与える環境変数の設定

シェルから z/OS UNIX COBOL プログラムの環境変数を設定するには、`export` または `set` コマンドを使用してください。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。

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
```

544 ページの『例: 環境変数の設定とアクセス』

関連タスク

545 ページの『UNIX/POSIX API の呼び出し』

321 ページの『z/OS UNIX のもとでの環境変数の設定』

関連参照

『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

40 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)

言語環境プログラム 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()` および `putenv()` は C 関数であるため、引数 BY VALUE を渡す必要があります。文字ストリングは、ヌル終了ストリングを指す BY VALUE ポインターとして渡します。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=ABCDEFGF".
01 rc pic 9(9) binary.
Linkage section.
```

```

01 var pic x(5000).
Procedure division.
* Retrieve and display the PATH environment variable
  Set P to address of PATH
  Call "getenv" using by value P returning var-ptr
  If var-ptr = null then
    Display "PATH not set"
  Else
    Set address of var to var-ptr
    Move 0 to var-len
    Inspect var tallying var-len
      for characters before initial X"00"
    Display "PATH = " var(1:var-len)
  End-if
* Set environment variable MYVAR to ABCDEFG
  Set P to address of putenv-arg
  Call "putenv" using by value P returning rc
  If rc not = 0 then
    Display "putenv failed"
  Stop run
End-if
Goback.

```

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 UNIX 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、言語環境プログラム の一部です。

これらは C 関数なので、BY VALUE によって引数を渡さなければなりません。文字ストリングは、ヌル終了ストリングを指す BY VALUE ポインターとして渡します。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、コンパイラー・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: 当該 API とともに >>CALLINTERFACE DYNAM ディレクティブを使用することはできません。

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 言語環境プログラム・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 VS COBOL II プログラムを実行したことがある。

- 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 言語環境プログラム・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 59. POSIX 関数呼び出しを使用したサンプル

目的	サンプル	使用される関数
一部のファイルおよびディレクトリー・ルーチンの用法を示す	IGYTFL1	<ul style="list-style-type: none"> • getcwd() • mkdir() • rmdir() • access()
iconv ルーチンを使用してデータを変換する方法を示す	IGYTENV	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • fork() • getpid() • getppid() • execl() • perror() • wait()
errno 値の入手方法を示す	IGYTERNO、IGYTGETE	<ul style="list-style-type: none"> • perror() • fopen()
プロセス間通信メッセージ・ルーチンの用法を示す	IGYTMSQ、IGYTMSQ2	<ul style="list-style-type: none"> • ftok() • msgget() • msgsnd() • perror() • fopen() • fclose() • msgrcv() • msgctl() • perror()

関連タスク

542 ページの『z/OS UNIX 環境での実行』

542 ページの『環境変数の設定およびアクセス』

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 UNIX 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 UNIX 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。

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。

『例: z/OS UNIX 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

542 ページの『z/OS UNIX 環境での実行』

542 ページの『環境変数の設定およびアクセス』

545 ページの『UNIX/POSIX API の呼び出し』

588 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 UNIX 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 UNIX *
*****
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
エレメント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 番目の文字ストリングへのポインターが含まれていま
す。

第 4 部 複雑なアプリケーションの構造化

第 24 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 (言語環境プログラム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、他の言語環境プログラム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。

言語環境プログラム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、言語環境プログラム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ー・ルーチン呼び出し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

『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』

552 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』

554 ページの『別のプログラムへの制御権移動』

567 ページの『再帰呼び出しの実行』

567 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』

568 ページの『プロシーチャー・ポインターと関数ポインターの使用』

570 ページの『プログラムを再入可能にする』

609 ページの『マルチスレッド化による COBOL 制限の処理』

言語環境プログラム ILC (言語間通信) アプリケーションの作成

関連参照

言語環境プログラム プログラミング・ガイド (規則の登録)

メインプログラム、サブプログラム、および呼び出し

COBOL プログラムが実行単位の最初のプログラムであれば、その COBOL プログラムはメインプログラム です。それ以外の場合は、そのプログラムも実行単位内の他のすべての COBOL プログラムも、サブプログラム です。特定のソース・コードのステートメントまたはオプションが、COBOL プログラムをメインプログラムまたはサブプログラムとして識別することはありません。

COBOL プログラムがメインプログラムであるかサブプログラムであるかは、次の 2 つの理由により重要になることもあります。

- プログラム終了処理ステートメントの影響

- 戻った後に再入する際のそのプログラムの状態

PROCEDURE DIVISION で、あるプログラムから別のプログラム (通常サブプログラムと呼ばれる) を呼び出すことができ、この呼び出し先プログラム自体からも別のプログラムを呼び出すことができます。別のプログラムを呼び出すプログラムは呼び出し側 プログラムと呼ばれ、そのプログラムが呼び出すプログラムは呼び出し先プログラムと呼ばれます。呼び出し先プログラムの処理が完了すると、そのプログラムは制御権を呼び出し側プログラムに戻すか、実行単位を終了することができます。

呼び出し先 COBOL プログラムは、PROCEDURE DIVISION の先頭で実行を開始します。

関連タスク

『メインプログラムまたはサブプログラムの終了と再入』

554 ページの『別のプログラムへの制御権移動』

567 ページの『再帰呼び出しの実行』

関連参照

言語環境プログラム 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。

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 60. 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。呼び出されるプログラムの中に次に実行可能なステートメントがない場合、暗黙の EXIT PROGRAM ステートメントが生成されます。 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。

表 60. 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
STOP RUN	<p>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。)</p> <p>STOP RUN は実行単位を終了し、実行単位内で動的に呼び出されたプログラムと、それらにリンク・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。)</p> <p>スレッド化された環境では、言語環境プログラム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。</p>	<p>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。)</p> <p>STOP RUN は実行単位を終了し、実行単位内で動的に呼び出されたプログラムと、それらにリンク・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。)</p> <p>スレッド化された環境では、言語環境プログラム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。</p>
GOBACK	<p>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。)</p> <p>GOBACK は実行単位を終了し、実行単位内で動的に呼び出されたプログラムと、それらにリンク・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。)</p> <p>スレッド化された環境では、スレッドが終了します。²</p>	<p>呼び出し側プログラムに戻ります。</p> <p>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と、スレッドが終了します。²</p>
<p>1. メインプログラムが、言語環境プログラム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。</p> <p>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。</p>		

サブプログラムは通常、EXIT PROGRAM または GOBACK で終了されると、最後に使用された状態になります。次回にサブプログラムが実行単位内で呼び出されると、PERFORM ステートメントの戻り値が初期値にリセットされることを除けば、その内部値は終了時のまま残されます (それに対して、メインプログラムは呼び出されるたびに初期化されます)。

プログラムが初期状態になるのは、次のような場合です。

- 動的に呼び出され、その後取り消されるサブプログラムは、次に呼び出されると初期状態になります。

- PROGRAM-ID 段落に INITIAL 節を持つプログラムは、呼び出されるたびに初期状態になります。
- LOCAL-STORAGE SECTION で定義されているデータ項目は、プログラムが呼び出されるたびに、VALUE 節で指定されている初期状態にリセットされます。

関連概念

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

言語環境プログラム 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

563 ページの『ネストされた COBOL プログラムの呼び出し』

567 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。

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。

旧レベルのプログラムで呼び出しを行う場合の制約事項については、「Enterprise COBOL for z/OS 移行ガイド」の『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』を参照してください。

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 プロシージャの代わりに、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 の代替方法として使用することができ、オブジェクト指向 COBOL アプリケーション、z/OS UNIX 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。このような個々の名前は、そのサブプログラムを含むプログラム・オブジェクトの NAME または ALIAS として、バインダー (リンケージ・エディター) 制御ステートメントで指定する必要があります。

関連概念

558 ページの『AMODE 切り替え』

- 560 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』
564 ページの『ネストされたプログラム』

関連タスク

- 『静的呼び出しの作成』
『動的呼び出しの作成』
561 ページの『静的呼び出しと動的呼び出しの両方の作成』
563 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)

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。

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。

- 561 ページの『例: 静的および動的 *CALL* ステートメント』

関連概念

- 560 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

- 『動的呼び出しの作成』
561 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『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

- 371 ページの『DLL』
373 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

プシオ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。

動的 `CALL` ステートメントで呼び出される各サブプログラムは、別のプログラム・オブジェクト (システム・リンク・ライブラリーまたはユーザー指定の専用ライブラリーのメンバーである) の一部である場合があります。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)

561 ページの『例: 静的および動的 `CALL` ステートメント』

関連概念

557 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』

560 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

557 ページの『サブプログラムの取り消し』

555 ページの『静的呼び出しの作成』

561 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

371 ページの『DLL』

373 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)

言語環境プログラム 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。

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。

561 ページの『例: 静的および動的 CALL ステートメント』

関連概念

560 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。

AMODE 24 プログラムが、31 ビット・アドレッシング・モードで実行する Enterprise COBOL プログラムと同じ実行単位にあるとします。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、言語環境プログラムのランタイム・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プログラムのライブラリー・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。

561 ページの『例: 静的および動的 CALL ステートメント』

関連概念

『AMODE 切り替え』

560 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

555 ページの『動的呼び出しの作成』

関連参照

373 ページの『DYNAM』

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

言語環境プログラム プログラミング・リファレンス

AMODE 切り替え

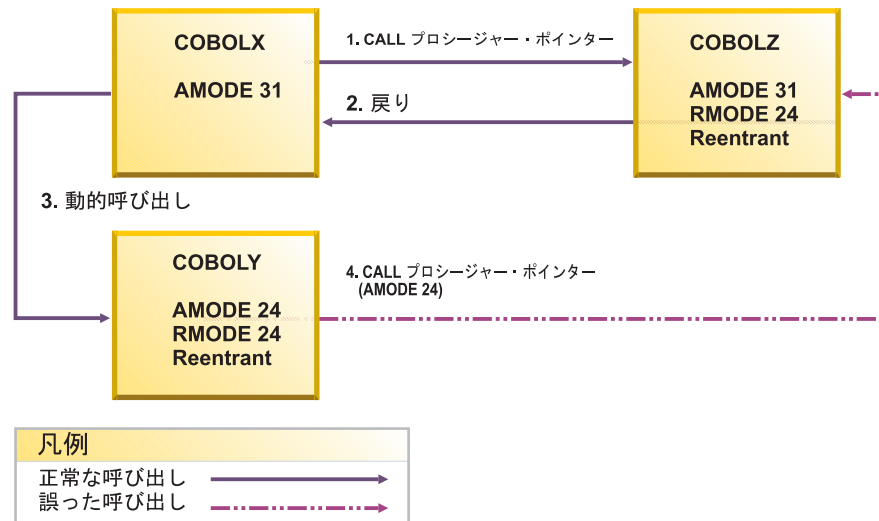
複数の COBOL サブプログラムを持つアプリケーションを使用する場合、その COBOL サブプログラムの一部を AMODE 31 に設定し、その他を AMODE 24 に設定

することができます。この混合 AMODE サポートを使用するには、言語環境プログラムのランタイム・オプション ALL31(OFF) および STACK(,,BELOW) を有効にして、呼び出しを動的に行う必要があります。

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。例えば、AMODE 31 COBOL プログラムから AMODE 24 COBOL プログラムへの動的呼び出しを行っている場合、AMODE は自動的に切り替えられます。

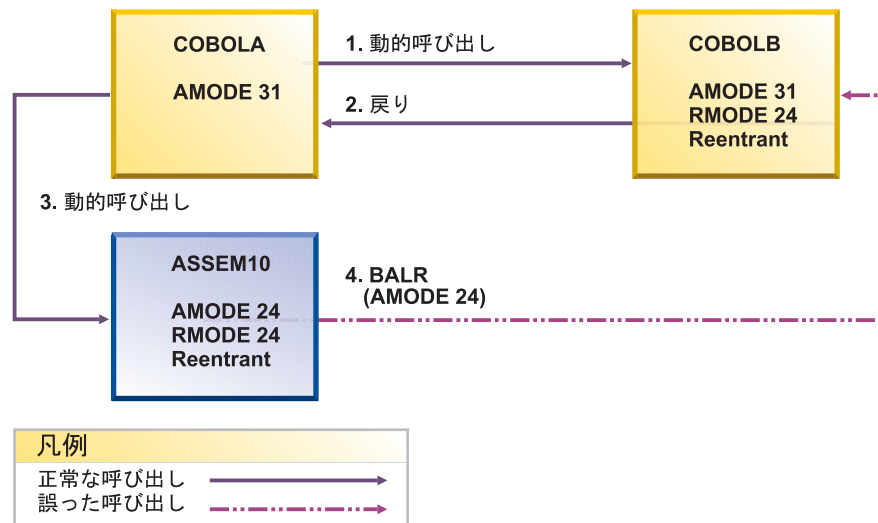
ただし、プロシージャ・ポインター、関数ポインター、または COBOL サブプログラムを呼び出す他の言語を使用する場合は、COBOL プログラムがエンクレープで複数回呼び出されたときに、呼び出されるたびに同じ AMODE になることを確認する必要があります。この場合、AMODE は自動的に切り替えられません。

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・ポインターを COBOLZ に設定します。COBOLZ は再入可能プログラム・オブジェクトであり、AMODE 31 と RMODE 24 です。COBOLX は、プロシージャ・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・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・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 COBOLA 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 はアセンブラ言語である ASSEM10 を動的に呼び出します。ASSEM10 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。ASSEM10 は AMODE 24 になります。
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

43 ページの『ストレージとそのアドレス可能度』

557 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

555 ページの『動的呼び出しの作成』

関連参照

言語環境プログラム 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。

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済みます。

関連概念

557 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

555 ページの『静的呼び出しの作成』

555 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

373 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).
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.
. . .
    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 ステートメントが実行されると、SUBPROG (呼び出されるプログラム) の PROCEDURE DIVISION の先頭のステートメントに制御権が移動します。
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) SUBPROG 中の処理が EXIT PROGRAM ステートメントに達すると、呼び出し側プログラムに制御権が戻されます。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。

呼び出し先プログラムと 2 つのいずれかの呼び出し側プログラムの特定の実行において、最初の CALL と 2 番目の CALL の間で RECORD-1 内の値が変更された場合は、2 番目の CALL ステートメントの実行時に渡される値は、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。

ネストされたプログラムの呼び出しには、CALL *literal* ステートメントまたは CALL *identifier* ステートメントを使用します。

含まれている プログラムは、それを直接収容するプログラムからのみ呼び出すことができます。ただし、含まれている プログラムをその PROGRAM-ID 段落で COMMON として識別している場合は別です。その場合、共通プログラム は、共通プログラムと同じプログラム内の任意の (直接的または間接的) 含まれている プログラムから呼び出すことができます。COMMON として識別できるのは、含まれている プログラムのみです。再帰呼び出しはできません。

ネストされたプログラム構造を使用する場合は、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。
- END PROGRAM マーカーを使用して、含まれているプログラムと収容しているプログラムを終了させます。

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

『ネストされたプログラム』

関連参照

566 ページの『名前のスコープ』

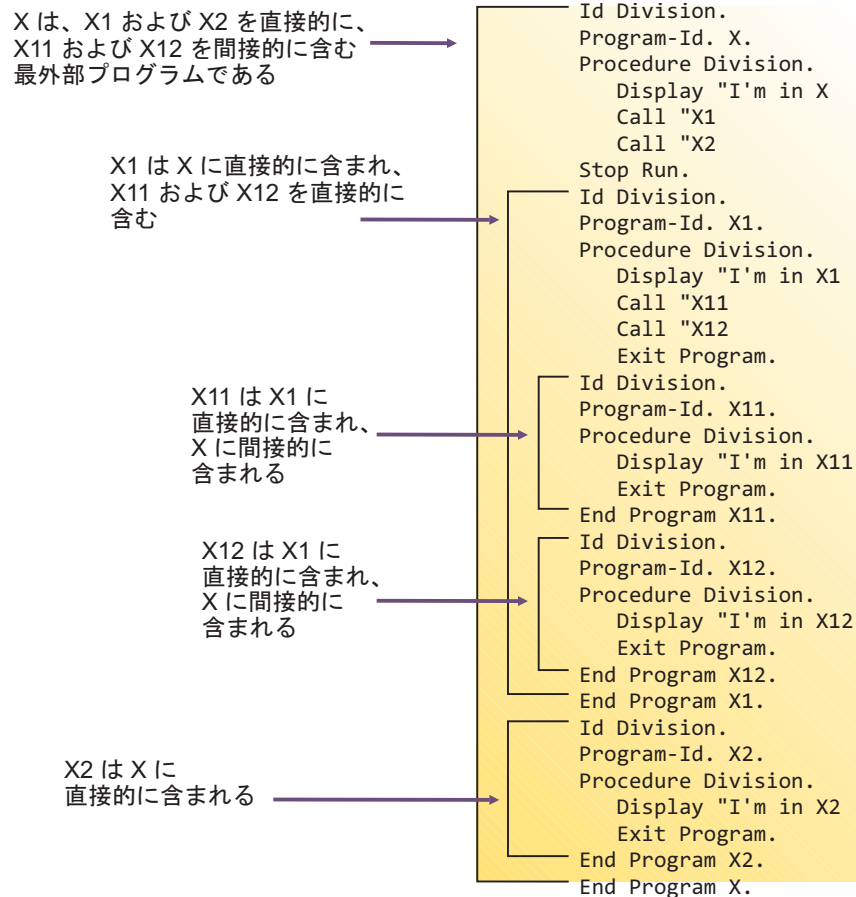
ネストされたプログラム

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、直接的および間接的に含まれたプログラムのあるネストされた構造を説明したものです。



『例: ネストされたプログラムの構造』

関連タスク

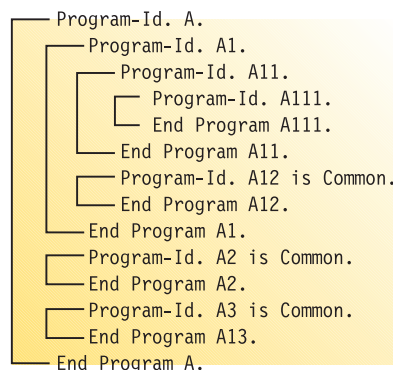
563 ページの『ネストされた COBOL プログラムの呼び出し』

関連参照

566 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認識されているかどうか、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、宣言されているプログラム内からのみ可視またはアクセス可能です。収容されているプログラムおよび収容プログラムでは、可視ではなく、アクセス可能でもありません。

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。

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が、これは各宣言が異なるプログラム内に現れる場合に限られます。同じ収容構造内の異なるプログラムで同じ名前を設定することにより、ネストされた構造内で名前のマスキング (非表示) 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。検索は、参照が含まれるプログラムから始まり、一致する名前が見つかるまで、収容プログラムに対して外側へ続行されます。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対するものであり、データ項目やファイル結合子などの名前に関連付けられた特定タイプのオブジェクトに対するものではありません。オブジェクトのタイプとは無関係に、何らかの一致する名前が見つかったら検索は停止します。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。このような呼び出しを再帰的と言います。

再帰呼び出しを行うには、再帰的に呼び出されるプログラムの PROGRAM-ID 段落で RECURSIVE 節をコーディングする必要があります。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。

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。

プロシージャ・ポインターとは、USAGE IS PROCEDURE-POINTER 節によって定義されるデータ項目です。関数ポインターとは、USAGE IS FUNCTION-POINTER 節によって定義されるデータ項目です。ここでは、「ポインター」は、プロシージャ・ポインター・データ項目または関数ポインター・データ項目のどちらかを指します。プロシージャ・ポインター・データ項目または関数ポインター・データ項目は、以下の入り口点の入り口アドレス (ポインター) が入るように設定することができます。

- ネストされていない別の COBOL プログラム。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 言語環境プログラム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。例えば、C 関数の入り口アドレスを受け取るためには、CALL RETURNING ステートメントを使用して関数を呼び出してください。この結果、SET ステートメントの形式を使用して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。

- NODYNAM オプションおよび 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、ポインター項目をリテラル値 (入り口点の実際の名前) に設定する場合、その値は、同じプログラム・オブジェクト内の入り口点を参照している必要があります。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。
- 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、ポインター項目をリテラル値 (入り口点の実際の名前) に設定する場合、その値は、同じプログラム・オブジェクト内の入り口点、または 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。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。

AMODE ANY 属性を持つ **COBOL** エントリー・ポイントには、**AMODE 31** または **AMODE 24** で入ることができます。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。

関連タスク

『使用するポインター・タイプの決定』

570 ページの『代替入り口点の呼び出し』

597 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

371 ページの『DLL』

373 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、言語環境プログラム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。

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、言語環境プログラム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

597 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』
743 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

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。

代替入り口点への動的呼び出しには次のエレメントが必要です。

- 明示的に指定された NAME 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生成する NAME コンパイラー・オプションの使用。
- 異なる入り口点での同じモジュールに対する動的呼び出しに対する、介在的 CANCEL。CANCEL を使用すると、プログラムは、新しい入り口点で呼び出されたとき初期状態で呼び出されます。

呼び出し先プログラムで ENTRY ラベルを使用することにより、プログラムが実行を開始する別の入り口点を指定できます。ただし、この方法は、構造化プログラムではお勧めしません。

561 ページの『例: 静的および動的 CALL ステートメント』

関連参照

392 ページの『NAME』

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・バック域にある 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。

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 UNIX 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

43 ページの『ストレージとそのアドレス可能度』

関連タスク

592 ページの『DLL を作成するためのプログラムのコンパイル』

333 ページの『第 16 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

409 ページの『RENT』

366 ページの『DATA』

言語環境プログラム プログラミング・リファレンス (ALL31、HEAP)

第 25 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とは、別のプログラムから呼び出される任意のプログラムのことです。

関連タスク

17 ページの『別のプログラムからのデータの使用』

748 ページの『Java とのデータ共用』

『データの受け渡し』

578 ページの『LINKAGE SECTION のコーディング』

578 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

583 ページの『戻りコード情報の引き渡し』

584 ページの『EXTERNAL 節によるデータの共用』

585 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)』

588 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。

BY REFERENCE

サブプログラムは、データのコピーを処理するのではなく、呼び出し側プログラムのストレージ内のデータ項目を参照して処理します。BY REFERENCE は、パラメーターに関して 3 つの方法のどれも指定されておらず、暗黙指定されてもいない場合の、パラメーターの想定引き渡しメカニズムです。

BY CONTENT

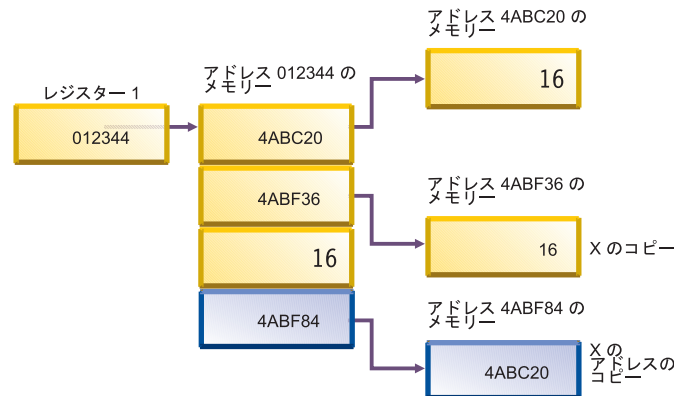
呼び出し側プログラムは、*literal* または *identifier* の内容だけを渡します。呼び出し先プログラムは、呼び出し側プログラム内の *literal* または *identifier* の値を変更できません。たとえ、*literal* または *identifier* を受け取ったデータ項目を変更する場合でも変更できません。

BY VALUE

呼び出し側プログラムまたはメソッドは、送り出しデータ項目を参照せず、*literal* または *identifier* の値を渡します。呼び出されるプログラムまたはメソッドは、その中のパラメーターを変更できます。しかし、サブプログラムまたはメソッドは送り出しデータ項目の一時コピーへのアクセス権しか持っていないので、どの変更内容も呼び出し側プログラムの引数に影響を与えません。

次の図に、BY REFERENCE、BY CONTENT、および BY VALUE によって渡される値の違いを示します。

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 61. CALL ステートメントでデータを渡す方法

コード	目的	コメント
CALL . . . BY REFERENCE <i>identifier</i>	呼び出し側プログラムの CALL ステートメントの引数の定義と呼び出されるプログラムのパラメーターの定義に、同じメモリーを共用させる。	サブプログラムがパラメーターに対して行う変更は、呼び出し側プログラムの引数に影響を与えます。
CALL . . . BY REFERENCE ADDRESS OF <i>identifier</i>	<i>identifier</i> のアドレスを呼び出し先プログラムに渡します。ここで、 <i>identifier</i> は LINKAGE SECTION 内の項目です。	サブプログラムがアドレスに対して行う変更は、呼び出し側プログラム内のアドレスに影響を与えます。
CALL . . . BY REFERENCE <i>file-name</i>	データ制御ブロック (DCB) をアセンブラ・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 <i>identifier</i>	<i>identifier</i> のアドレスのコピーを、呼び出し先プログラムに渡します。	アドレスのコピーに任意の変更を加えても <i>identifier</i> のアドレスには影響しませんが、アドレスのコピーを使用する <i>identifier</i> を変更すると <i>identifier</i> が変更されます。
CALL . . . BY CONTENT ID	ID のコピーをサブプログラムに渡す。	サブプログラムによってパラメーターを変更しても、呼び出し側の ID には影響しません。
CALL . . . BY CONTENT <i>literal</i>	呼び出し先プログラムにリテラル値のコピーを渡す。	
CALL . . . BY CONTENT LENGTH OF <i>identifier</i>	データ項目の長さのコピーを渡す。	呼び出し側プログラムは、その LENGTH 特殊レジスターから <i>identifier</i> の長さを渡します。
次のような BY REFERENCE と BY CONTENT の組み合わせ: CALL 'ERRPROC' USING BY REFERENCE A BY CONTENT LENGTH OF A.	データ項目とその長さのコピーの両方をサブプログラムに渡す。	

表 61. 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 <i>identifier</i>	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用するプログラムにデータを渡す。	ID のコピーがパラメーター・リストとして直接渡されます。
CALL . . . BY VALUE <i>literal</i>	C/C++ プログラムなど、BY VALUE パラメーター・リンケージ規約を使用するプログラムにデータを渡す。	リテラルのコピーがパラメーター・リストとして直接渡されます。
CALL . . . BY VALUE ADDRESS OF <i>identifier</i>	呼び出し先プログラムに <i>identifier</i> のアドレスを渡す。データに対するポインターを必要とする C/C++ プログラムにデータを渡すのに推奨される方法。	アドレスのコピーに任意の変更を加えても <i>identifier</i> のアドレスには影響しませんが、アドレスのコピーを使用する <i>identifier</i> を変更すると <i>identifier</i> が変更されます。
CALL . . . RETURNING	関数戻り値を使用して C/C++ 関数を呼び出す。	
1. CALL オペランドとしてのファイル名は、COBOL への IBM 拡張部分として許可されます。拡張部分の使用は一般的に、コンパイラーの特定の内部実装によって異なります。制御ブロック・フィールド設定は、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

43 ページの『ストレージとそのアドレス可能性』

関連タスク

『呼び出し側プログラムの中での引数の記述』

576 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』

577 ページの『OMITTED 引数に関するテスト』

584 ページの『CALL . . . RETURNING の指定』

584 ページの『EXTERNAL 節によるデータの共用』

585 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)』

748 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

呼び出し側プログラムでは、DATA DIVISION の他のデータ項目と同じ方法で、DATA DIVISION で引数を記述します。

引数のためのストレージは、最外部プログラムにおいてのみ割り振られます。例えば、プログラム A がプログラム B を呼び出し、プログラム B がプログラム C を呼び出すとします。データ項目はプログラム A で割り振られ、プログラム B と C の LINKAGE SECTION で記述され、そのデータの集合を 3 つのすべてのプログラムで使用できます。

ファイルのデータを参照する場合、データが参照される時には、そのファイルはオープンされていなければなりません。

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、43 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

43 ページの『ストレージとそのアドレス可能性』

関連タスク

578 ページの『LINKAGE SECTION のコーディング』

578 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。

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

578 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 引数に関するテスト

CALL ステートメント内の引数の代わりに OMITTED キーワードをコーディングして、1 つ以上の BY REFERENCE 引数が、呼び出し先プログラムに渡されないように指定することができます。

例えば、プログラム sub1 を呼び出すときに、2 番目の引数を省略するには、次のステートメントをコーディングします。

```
Call 'sub1' Using PARM1, OMITTED, PARM3
```

CALL ステートメントの USING 句の引数は、数および位置において呼び出し先プログラムのパラメーターと一致しなければなりません。

呼び出し先プログラムで、対応するパラメーターのアドレスを NULL と比較して、引数が OMITTED として渡されたかどうかをテストすることができます。以下に例を示します。

```
Program-ID. sub1.  
.  
.  
Procedure Division Using RPARM1, RPARM2, RPARM3.  
    If Address Of RPARM2 = Null Then  
        Display 'No 2nd argument was passed this time'  
    Else  
        Perform Process-Parm-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)

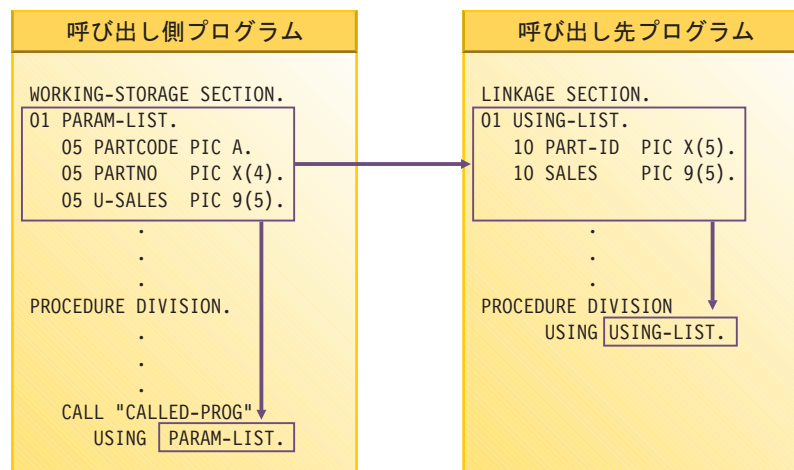
USING 句 (*Enterprise COBOL for z/OS* 言語解説書)

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。

呼び出し先プログラムの identifier リストのデータ名の数、呼び出し側プログラムから渡される引数の数よりも大きいと、エラーになります。コンパイラーは引数とパラメーターの突き合わせを試行しません。

次の図は、あるプログラムから別のプログラムにデータ項目が渡される様子を示しています (暗黙的に BY REFERENCE)。



呼び出し側プログラムでは、パーツ (PARTCODE) とパーツ・ナンバー (PARTNO) のコードは別のデータ項目です。それに対して、呼び出し先プログラムでは、パーツのコードとパーツ・ナンバーのコードが 1 つのデータ項目 (PART-ID) に結合されています。呼び出し先プログラムでは、PART-ID への参照は、これらの項目に対する唯一有効な参照です。

関連タスク

588 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。

```
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

プログラム間で渡す必要があるすべてのデータ項目をグループ化し、それらを 1 つのレベル 01 項目に入れることを考慮してください。 そうすると、1 個のレベル 01 レコードを渡すことができます。

データ項目を BY VALUE で渡す場合、データ項目は基本項目でなければならないことに注意してください。

レコード突き合わせの間違いの可能性をより少なくするためには、レベル 01 レコードをコピー・ライブラリーの中に置き、両方のプログラムにそれをコピーするようにします。すなわち、呼び出し側プログラムの WORKING-STORAGE SECTION と呼び出し先プログラムの LINKAGE SECTION の中でコピーします。

関連タスク

578 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。

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

123 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

レコード域のアドレスを渡したり受信する必要がある場合、ポインター・データ項目を使用できます。これは、USAGE IS POINTER 節を使用して定義されるデータ項目、または ADDRESS OF 特殊レジスターであるデータ項目のいずれかです。

ポインター・データ項目の代表的な適用は、チェーン・リスト (各レコードがそれぞれ次のレコードを指し示す一連のレコード) の処理です。

プログラム相互間のアドレスをチェーン・リストに入れて渡す場合は、NULL を使用して、以下の 2 つの方法のいずれかで、無効なアドレスの値 (非数値 0) をポインター項目に割り当てることができます。

- データ定義の中で VALUE IS NULL 節を使用する。
- SET ステートメントの中で送信フィールドとして NULL を使用する。

最後のレコードのポインター・データ項目がヌル値を含んでいるチェーン・リストの場合、このコードを使用して、リストの終わりを検査できます。

```
IF PTR-NEXT-REC = NULL  
...  
(logic for end of chain)
```

リストの終わりに達していない場合、プログラムはレコードを処理して次のレコードに移ることができます。

呼び出し側プログラムから渡されるデータには、無視したいヘッダー情報が含まれていることがあります。ポインター・データ項目は数値ではないので、この項目に関する演算を直接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。

581 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

578 ページの『LINKAGE SECTION のコーディング』

578 ページの『引数を受け渡すための PROCEDURE DIVISION のコーディング』

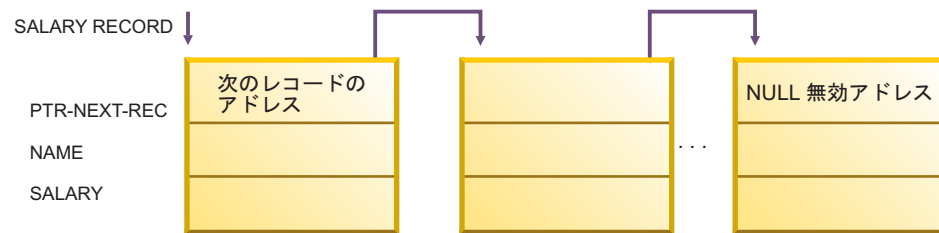
関連参照

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・リストを示しています。下図は、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き、各レコードの最初の項目は次のレコードを指しています。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) ノ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疑似コードは、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
```

以下のコードには、このチェーン・リスト処理例で使用される呼び出し側プログラム `LISTS` の概要が含まれています。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LISTS.
ENVIRONMENT DIVISION.
DATA DIVISION.
*****
WORKING-STORAGE SECTION.
77 PTR-FIRST          POINTER VALUE IS NULL.
77 DEPT-TOTAL         PIC 9(4) VALUE IS 0.
*****
LINKAGE SECTION.
01 SALARY-REC.
   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
```

```

      SET ADDRESS OF SALARY-REC TO PTR-FIRST              (4)
*****
      PERFORM WITH TEST BEFORE UNTIL ADDRESS OF SALARY-REC = NULL (5)

      IF DEPT = DEPT-X
        THEN ADD SALARY TO DEPT-TOTAL
        ELSE CONTINUE
      END-IF
      SET ADDRESS OF SALARY-REC TO PTR-NEXT-REC            (6)

      END-PERFORM
*****
      DISPLAY DEPT-TOTAL
      GOBACK.

```

- (1) PTR-FIRST は、NULL の初期値を持つポインター・データ項目として定義されます。CHAIN-ANCH への呼び出しから正常に戻ると、PTR-FIRST には、チェーン・リスト内の最初のレコードのアドレスが入れられます。呼び出しで何か間違いが起これ、PTR-FIRST がチェーンの最初のレコードのアドレスの値を受け取っていないと、PTR-FIRST はヌル値のままであり、プログラムのロジックに従って、レコードは処理されません。
- (2) 呼び出し側プログラムの LINKAGE SECTION には、チェーン・リストのレコードの記述が入っています。さらに、CALL ステートメントの USING 節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。

別のプログラムから受け取ったアドレスを増分するには、LINKAGE SECTION および PROCEDURE DIVISION を次のようにセットアップすることができます。

```

LINKAGE SECTION.
01 RECORD-A.
   02 HEADER          PIC X(12).
   02 REAL-SALARY-REC PIC X(30).
. . .
01 SALARY-REC.
   02 PTR-NEXT-REC     POINTER.
   02 NAME             PIC X(20).
   02 DEPT             PIC 9(4).
   02 SALARY           PIC 9(6).
. . .
PROCEDURE DIVISION USING DEPT-X.
. . .
      SET ADDRESS OF SALARY-REC TO ADDRESS OF REAL-SALARY-REC

```

この時点では、SALARY-REC のアドレスは、REAL-SALARY-REC、または RECORD-A + 12 のアドレスを基底にしています。

関連タスク

580 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。)

また、メソッドの PROCEDURE DIVISION ヘッダーで RETURNING 句を使用して、起動プログラムまたはメソッドに情報を戻すこともできます。PROCEDURE DIVISION . . . RETURNING で CALL . . . RETURNING を指定しても、RETURN-CODE レジスターは設定されません。

RETURN-CODE 特殊レジスターの使用

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あることがあります。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。

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: 言語環境プログラム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 . . . RETURNING の指定

C/C++ 関数または COBOL サブルーチンへの呼び出しでは、CALL ステートメントの RETURNING 句を指定することができます。

RETURNING 句のフォーマットは次のとおりです。

CALL . . . RETURNING *dataname2*

呼び出し先プログラムの戻り値は *dataname2* に保管されます。 *dataname2* は、呼び出し側プログラムの DATA DIVISION で定義しなければなりません。 ターゲット関数で宣言される戻り値のデータ型は、*dataname2* のデータ型と同じでなければなりません。

EXTERNAL 節によるデータの共用

EXTERNAL 節を使用すると、別々にコンパイルされたプログラムやメソッド (バッチ・シーケンスのプログラムを含む) がデータ項目を共用できるようになります。 WORKING-STORAGE SECTION のレベル 01 データ記述に EXTERNAL をコーディングします。

次の規則が適用されます。

- EXTERNAL グループ項目に従属する項目はそれ自体が 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。
- VALUE 文節は、EXTERNAL であるいずれのグループ項目、従属項目についてもコーディングできません。

実行単位内で、ある COBOL プログラムまたはメソッドの項目のデータ記述が、その項目を含むプログラムのデータ記述と同じ場合は、そのプログラムまたはメソッドはその項目にアクセスして処理できます。例えば、プログラム A に次のデータ記述があるとして。

```
01 EXT-ITEM1      EXTERNAL      PIC 99.
```

プログラム B は、WORKING-STORAGE SECTION に同じデータ記述が存在する場合、そのデータ項目にアクセスすることができます。

EXTERNAL データ項目にアクセス権を持っているプログラムはすべて、その項目の値を変更できます。そのため、保護しなければならないデータ項目には、この節を使用しないでください。

プログラム間でのファイルの共用 (外部ファイル)

実行単位の別々にコンパイルされたプログラムまたはメソッドが共用ファイルとしてファイルにアクセスできるようにするには、そのファイルに `EXTERNAL` 節を使用します。

以下の指針に従うことをお勧めします。

- ファイル状況コードを検査するすべてのプログラムの `FILE STATUS` 節で、同じデータ名を使用する。
- 同じファイル状況フィールドを検査するすべてのプログラムについて、ファイル状況フィールドのレベル 01 データ定義で `EXTERNAL` 節をコーディングする。

外部ファイルを使用すると、次のような利点があります。

- メインプログラムに入出力ステートメントが含まれていなくても、ファイルのレコード域を参照することができます。
- それぞれのサブプログラムが、`OPEN` や `READ` のような単一の入出力機能を制御することができます。
- それぞれのプログラムがファイルにアクセスすることができます。

『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	機能
efl	メインプログラム。すべてのサブプログラムを呼び出し、レコード域の内容を検査する。
eflopeno	出力用に外部ファイルをオープンし、ファイル状況コードを検査する。
eflwrite	外部ファイルにレコードを書き込み、ファイル状況コードを検査する。
eflopeni	入力用に外部ファイルをオープンし、ファイル状況コードを検査する。
eflread	外部ファイルからレコードを読み取り、ファイル状況コードを検査する。
eflclose	外部ファイルをクローズし、ファイル状況コードを検査する。

各プログラムは、次の 3 つのコピーブックを使用します。

- `efselect` は `FILE-CONTROL` 段落に入れられます。

- ```

Select ef1
Assign To ef1
File Status Is efs1
Organization Is Sequential.

```
- `effile` は `FILE SECTION` に入れます。

```

Fd ef1 Is External
 Record Contains 80 Characters
 Recording Mode F.
01 ef-record-1.
 02 ef-item-1 Pic X(80).

```
  - `efwrkstg` は `WORKING-STORAGE SECTION` に入れます。

```

01 efs1 Pic 99 External.

```

## 外部ファイルを使用する入出力

```

IDENTIFICATION DIVISION.
Program-Id.
 ef1.
*
* This main program controls external file processing.
*
ENVIRONMENT DIVISION.
Input-Output Section.
File-Control.
 Copy efselect.
DATA DIVISION.
FILE SECTION.
 Copy effile.
WORKING-STORAGE SECTION.
 Copy efwrkstg.
PROCEDURE DIVISION.
 Call "eflopeno"
 Call "eflwrite"
 Call "eflclose"
 Call "eflopeni"
 Call "eflread"
 If ef-record-1 = "First record" Then
 Display "First record correct"
 Else
 Display "First record incorrect"
 Display "Expected: " "First record"
 Display "Found : " ef-record-1
 End-If
 Call "eflclose"
 Goback.
End Program 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 effile.
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.
 eflwrite.
*
* This program writes a record to the external file.
*
ENVIRONMENT DIVISION.
Input-Output Section.
File-Control.
 Copy efselect.
DATA DIVISION.
FILE SECTION.
 Copy effile.
WORKING-STORAGE SECTION.
 Copy efwrkstg.
PROCEDURE DIVISION.
 Move "First record" to ef-record-1
 Write ef-record-1
 If efs1 Not = 0
 Display "file status " efs1 " on write"
 Stop Run
 End-If
 Goback.
End Program eflwrite.
Identification Division.
Program-Id.
 eflopeni.
*
* This program opens the external file for input.
*
ENVIRONMENT DIVISION.
Input-Output Section.
File-Control.
 Copy efselect.
DATA DIVISION.
FILE SECTION.
 Copy effile.
WORKING-STORAGE SECTION.
 Copy efwrkstg.
PROCEDURE DIVISION.
 Open Input ef1
 If efs1 Not = 0
 Display "file status " efs1 " on open input"
 Stop Run
 End-If
 Goback.
End Program eflopeni.
Identification Division.
Program-Id.
 eflread.
*
* This program reads a record from the external file.
*
ENVIRONMENT DIVISION.
Input-Output Section.
File-Control.
 Copy efselect.
DATA DIVISION.
FILE SECTION.
 Copy effile.
WORKING-STORAGE SECTION.
 Copy efwrkstg.

```

```

PROCEDURE DIVISION.
 Read ef1
 If efs1 Not = 0
 Display "file status " efs1 " on read"
 Stop Run
 End-If
 Goback.
End Program eflread.
Identification Division.
Program-Id.
 eflclose.
*
* This program closes the external file.
*
ENVIRONMENT DIVISION.
Input-Output Section.
File-Control.
 Copy efselect.
DATA DIVISION.
FILE SECTION.
 Copy effile.
WORKING-STORAGE SECTION.
 Copy efwrkstg.
PROCEDURE DIVISION.
 Close ef1
 If efs1 Not = 0
 Display "file status " efs1 " on close"
 Stop Run
 End-If
 Goback.
End Program eflclose.

```

---

## 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。

下の例に示されているように、パラメーター・ストリングには、LINKAGE SECTION および標準的な COBOL コーディングを使用してアクセスできます。

589 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 言語環境プログラム 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*

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

578 ページの『LINKAGE SECTION のコーディング』

547 ページの『z/OS UNIX 下でのメインプログラム・パラメーターへのアクセス』

言語環境プログラム プログラミング・ガイド (『ランタイムオプションおよびプログラム引数の指定』、『パラメーターを受け取るメインルーチンの作成』)

#### 関連参照

言語環境プログラム カスタマイズ (CBLOPTS (COBOL のみ))

言語環境プログラム プログラミング・リファレンス (『CEE3PRM』、『CEE3PR2』)

## 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します。

'95/'

結果の出力は、次のとおりです。

```
parm-len=002
parm-string='95'
case ninety-five
```



---

## 第 26 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。

DLL または DLL アプリケーションを作成する際には、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ーおよびバインダー (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

『ダイナミック・リンク・ライブラリー (DLL)』

### 関連タスク

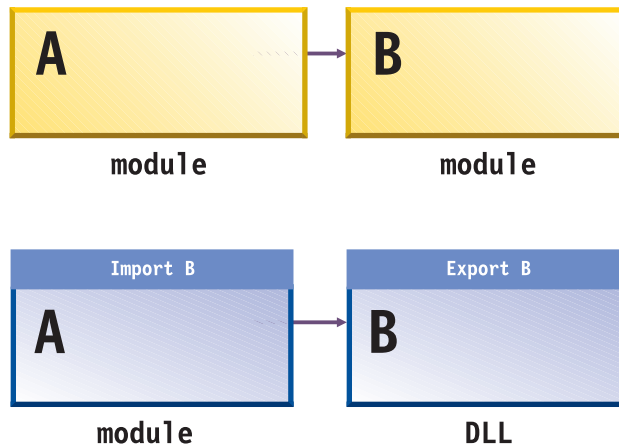
- 325 ページの『z/OS UNIX のもとでの DLL の作成』
- 592 ページの『DLL を作成するためのプログラムのコンパイル』
- 593 ページの『DLL のリンク』
- 595 ページの『DLL での CALL ID の使用』
- 596 ページの『DLL リンケージと動的呼び出しの併用』
- 600 ページの『C/C++ プログラムでの COBOL DLL の使用』
- 601 ページの『OO COBOL アプリケーションでの DLL の使用』
- 597 ページの『DLL でのプロシージャ・ポインターまたは関数ポインターの使用』

---

## 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 へエクスポート 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。

z/OS DLL のいくつかの機能は COBOL の動的 CALL ステートメントで提供される機能とオーバーラップしますが、DLL の方が、次のように、通常の z/OS プログラム・オブジェクトおよび動的呼び出しよりも利点があります。

- 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 言語環境プログラム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 UNIX、または IMS 環境で使用可能です。

関連参照

405 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 コンパイラー・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。

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合に応じて以下のコンパイラー・オプションを使用してください。

表 62. DLL アプリケーションのコンパイラー・オプション

|                                          |                      |
|------------------------------------------|----------------------|
| プログラムまたはクラスの場合                           | 使用するオプション            |
| ルート・プログラム・オブジェクト                         | DLL、RENT、NOEXPORTALL |
| 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   |

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。

594 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

325 ページの『z/OS UNIX のもとでの DLL の作成』

『DLL のリンク』

591 ページの『第 26 章 DLL または DLL アプリケーションの作成』

#### 関連参照

371 ページの『DLL』

377 ページの『EXPORTALL』

409 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。

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、PDS ではなく、PDSE または z/OS UNIX ファイル。

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 63. 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 | CASE(MIXED)          |
| クラス定義または INVOKE ステートメント             |                      |

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。

『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

325 ページの『z/OS UNIX のもとでの DLL の作成』

591 ページの『第 26 章 DLL または DLL アプリケーションの作成』

592 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**

以下の例は、DLL サブプログラムを呼び出すメインプログラムから構成されるアプリケーションの作成方法を示しています。

最初のステップでは、サブプログラム `DemoDLLSubprogram` を含む DLL プログラム・オブジェクトを作成します。2 番目のステップでは、プログラム `MainProgram` を含むメイン・プログラム・オブジェクトを作成します。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'
//*-----
//* Compile COBOL subprogram, bind to form a DLL.
//*-----
//STEP1 EXEC IGYWCL,REGION=80M,GOPGM=DEMODLL,
// PARM.COBOLE='RENT,PGMN(LM),DLL,EXPORTALL',
// PARM.LKED='RENT,LIST,XREF,LET,MAP,DYNAM(DLL),CASE(MIXED)'
//COBOL.SYSIN DD *
 Identification division.
 Program-id. "DemoDLLSubprogram".
 Procedure division.
 Display "Hello from DemoDLLSubprogram!".
 End program "DemoDLLSubprogram".
/*
//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"
 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=*
//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。

*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・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。

ランタイム環境が上記のいずれかの CALL ステートメントを解決できない場合、制御権は CALL ステートメントの ON EXCEPTION または ON OVERFLOW 句に移動します。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、言語環境プログラムは重大度 3 の条件を発生させます。

### 関連タスク

- 596 ページの『DLL リンケージと動的呼び出しの併用』
- 592 ページの『DLL を作成するためのプログラムのコンパイル』
- 593 ページの『DLL のリンク』

#### 関連参照

371 ページの『DLL』

405 ページの『PGMNAME』

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

『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、言語環境プログラムの 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

595 ページの『DLL での CALL ID の使用』

#### 関連参照

言語環境プログラム 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 言語環境プログラム エンクレープ) 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、呼び出し元では、さまざまなモジュールを呼び出すために、どちらのリンケージ・タイプを持つ CALL ステートメントも使用できます。

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、あるいは DLL を指定する CALLINTERFACE コンパイラ指示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、コンパイラー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 コンパイラー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

591 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

592 ページの『DLL を作成するためのプログラムのコンパイル』

593 ページの『DLL のリンク』

『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』

598 ページの『非 DLL からの DLL の呼び出し』

#### 関連参照

371 ページの『DLL』

377 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。

SET *procedure-pointer-1* TO ENTRY *entry-name* ステートメント、SET *function-pointer-1* TO ENTRY *entry-name* ステートメント、および CALL ステートメントには、それらに関連付けられた呼び出しリンケージ・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ー・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ー・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) コンパイラー・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

- 595 ページの『DLL での CALL ID の使用』
- 568 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』
- 592 ページの『DLL を作成するためのプログラムのコンパイル』
- 593 ページの『DLL のリンク』

#### 関連参照

- 371 ページの『DLL』
- 377 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。

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 ルーチンを呼び出します。

『例: 非 DLL からの DLL の呼び出し』

関連タスク

568 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'OOC05R' 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

DLL の COBOL サポートは、COBOL EXTERNAL データを除いて、z/OS C/C++ 製品の DLL サポートと相互協調で処理します。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。

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、および EXPORTALL は、C/C++ DLL、RENT、および EXPORTALL オプションとほとんど同じように機能します。(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

- 592 ページの『DLL を作成するためのプログラムのコンパイル』
- 593 ページの『DLL のリンク』

#### 関連参照

- 371 ページの『DLL』
- 377 ページの『EXPORTALL』
- 409 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。

#### 関連タスク

- 333 ページの『第 16 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』
- 592 ページの『DLL を作成するためのプログラムのコンパイル』
- 593 ページの『DLL のリンク』

#### 関連参照

- 371 ページの『DLL』
- 428 ページの『THREAD』
- 409 ページの『RENT』
- 367 ページの『DBCS』





---

## 第 27 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 UNIX 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。

マルチスレッド化による実行のための明示的な COBOL 言語は存在しません。正しくは、THREAD コンパイラー・オプションを使用してコンパイルを行います。

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、マルチスレッド・アプリケーション・サーバー、スレッドの作成に C/C++ ドライバー・プログラムを使用するアプリケーション、Java と連携して動作する Java スレッドを使用するプログラム、および PL/I タスクを使用するアプリケーションで、THREAD コンパイラー・オプションを使用してコンパイルする COBOL プログラムを実行することができます。つまり、他のプログラムは、プロセス内の複数スレッドで、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 言語環境プログラム・エンクレープ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** 内のデータ項目は、プログラムごとに一度割り振られるため、最後に使われた状態がプログラムのすべての起動において使用可能です。

個々のプログラム起動インスタンスに分離するデータの場合、そのデータを **LOCAL-STORAGE SECTION** で定義します。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

604 ページの『マルチスレッド化』

### 関連タスク

605 ページの『マルチスレッド化サポートのための THREAD の選択』

606 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』

606 ページの『マルチスレッド化されたプログラムの終了』

607 ページの『マルチスレッド化によるファイルの処理』  
609 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

428 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。

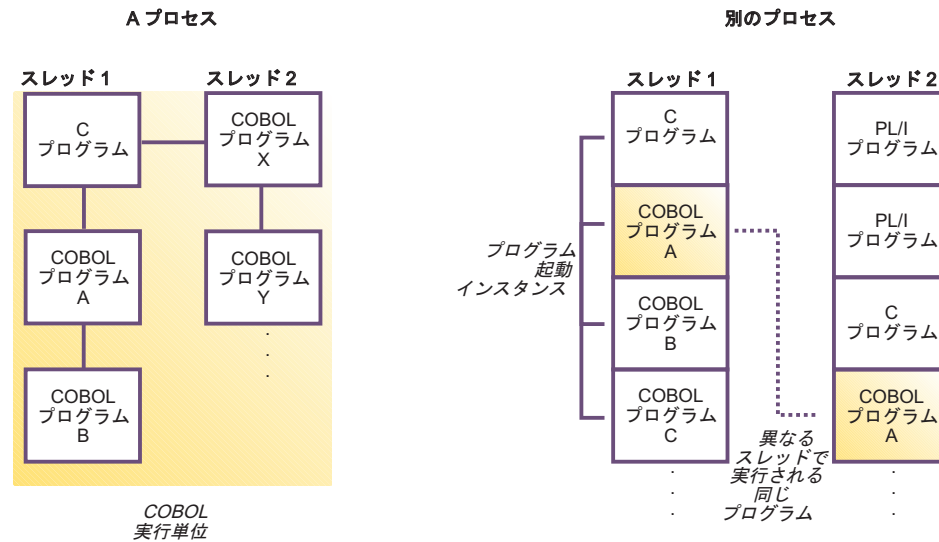
オペレーティング・システムおよびマルチスレッド化アプリケーションは、プロセス 内の実行フローを処理することができます。実行フローとは、プログラムのすべて、または一部が実行時に発生する一連のイベントです。1 つのプロセス内で実行されるプログラムは、リソースを共用できます。プロセスは操作することができます。例えば、システムがプロセスの実行に使用する時間において、プロセスに高い、または低い優先順位を持たせることができます。

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 つの命令のストリーム (1 つのスレッド) 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できます。これらの複数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、実行中のプログラム間で制御権が移動します。

マルチスレッド化環境で、COBOL 実行単位 は、実行中のアクティブな COBOL プログラムが含まれているスレッドを含むプロセスの部分です。COBOL 実行単位は、どのスレッドの実行スタックにもアクティブな COBOL プログラムがなくなるまで続行されます。例えば、呼び出される COBOL プログラムには GOBACK ステートメントが含まれていて、C プログラムに制御を戻します。実行単位内では、COBOL プログラムは非 COBOL プログラムを呼び出すことができ、逆に、非 COBOL プログラムは COBOL プログラムを呼び出すことができます。

スレッド内では、別々の COBOL および非 COBOL プログラムの間で制御が移動します。例えば、COBOL プログラムは別の COBOL プログラムまたは C プログラムを呼び出すことができます。別々に呼び出されたプログラムはそれぞれ、プログラム起動インスタンス です。特定プログラムのプログラム起動インスタンスは、指定したプロセス内で複数のスレッドに存在することができます。

次の図に、プロセス、スレッド、実行単位、およびプログラム起動インスタンスの間の関係を示します。



## 関連概念

言語環境プログラム プログラミング・ガイド (プログラム管理モデル、基礎知識: スレッド)

## 関連タスク

『マルチスレッド化サポートのための THREAD の選択』

606 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』

606 ページの『マルチスレッド化されたプログラムの終了』

607 ページの『マルチスレッド化によるファイルの処理』

609 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

428 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。プログラムが 1 つのアプリケーション によって単一プロセス内の複数のスレッドで呼び出される場合には、THREAD を使用します。ただし、THREAD は、シリアライゼーション・ロジックが自動的に生成されるため、パフォーマンスに悪影響を与える可能性があります。

COBOL プログラムを複数のスレッドで実行するには、THREAD コンパイラー・オプションを使用して、アプリケーション内の COBOL プログラムのすべてをコンパイルしなければなりません。また、RENT コンパイラー・オプションを使用してプログラムをコンパイルし、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する必要があります。

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: THREAD コンパイラー・オプションを使用してプログラムをコンパイルする前に、PROGRAM-ID 段落で RECURSIVE 句を指定する必要があります。指定しなければ、エラーが発生します。

#### 関連タスク

18 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』

333 ページの『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コンパイル』

#### 関連参照

428 ページの『THREAD』

---

## マルチスレッド化されたプログラムへの制御権移動

マルチスレッド化環境用に COBOL プログラムを書くときは、適切なプログラム・リンケージ・ステートメントを選択します。

単一スレッド環境の場合のように、実行単位内で最初に呼び出されるとき、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 の言語環境プログラム条件が発生します。

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、言語環境プログラム・サービス (CEEPIPI インターフェース) を使用します。事前初期設定のための COBOL 固有のインターフェース (ランタイム・オプション RTEREUS) を使用して、THREAD オプションでコンパイルしたプログラムから再使用可能環境を設定することはできません。

#### 関連概念

言語環境プログラム 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

『マルチスレッド化されたプログラムの終了』

552 ページの『メインプログラムまたはサブプログラムの終了と再入』

---

## マルチスレッド化されたプログラムの終了

GOBACK、EXIT PROGRAM、または STOP RUN を使用して、マルチスレッド・プログラムを終了させることができます。

プログラムの呼び出し側に戻るには、GOBACK を使用します。あるスレッド内の最初のプログラムから GOBACK を使用するとき、そのスレッドは終了します。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。

メインプログラムからの場合を除き、GOBACK と同様に、EXIT PROGRAM を使用します。メインプログラムでは EXIT PROGRAM は無効です。

言語環境プログラム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

言語環境プログラム 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

552 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。

各ファイル定義 (FD) には暗黙のシリアルイゼーション・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、自動シリアルイゼーション・ロジックとともに使用されます。

- OPEN
- CLOSE
- READ
- WRITE
- REWRITE
- START
- DELETE

また、自動シリアルイゼーションは、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
```

自動シリアルイゼーションは、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

608 ページの『ファイル定義 (FD) ストレージ』

#### 関連タスク

195 ページの『QSAM ファイルのクローズ』

227 ページの『VSAM ファイルのクローズ』

## 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

『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。

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際にシリアライゼーション問題の発生を避けるには、LOCAL-STORAGE SECTION で、ファイルに関連付けられたデータ項目(ファイル状況データ項目やキー引数) を定義します。

『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

545 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。

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:** マルチスレッド化アプリケーションは CICS において実行することはできません。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、言語環境プログラム・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、言語環境プログラムのエンクレーブ全体 (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 および言語環境プログラム準拠アセンブラー:** 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プログラム準拠アセンブラー・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:** マルチスレッド化アプリケーションの複数のスレッド上で **COBOL** プログラムを実行するには、それらを **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

567 ページの『再帰呼び出しの実行』

608 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)

言語環境プログラム **ILC** (言語間通信) アプリケーションの作成



---

## 第 5 部 **Web** サービスに **COBOL** を使用



---

## 第 28 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ーを使用して 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、任意のソースにある JSON テキストを使用するための組み込み z/OS JSON パーサー (z/OS V2.2、または APAR OA46575 の PTF 適用済み z/OS V2.1 で提供) を備えているため、アプリケーションはクライアント/サーバーのスペースにアクセスできます。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。詳しくは、「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

- 617 ページの『第 29 章 JSON 入力の処理』
- 625 ページの『第 30 章 JSON 出力の生成』
- 627 ページの『第 31 章 XML 入力の処理』
- 675 ページの『第 32 章 XML 出力の生成』



## 第 29 章 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。

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
- 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
- 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 UTF-8 (CCSID 1208) でエンコードされているとみなされるため、英数字グループ項目、またはカテゴリー「英数字」の基本データ項目に入っている必要があります。

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。<sup>1</sup>

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 suppl.
```

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. occl.
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 occl.

```

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. odol.
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 od01.

```

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 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。

```

CBL CODEPAGE(1047)
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.
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

```



---

## 第 30 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。

- 生成された文字のカウントを受け取るデータ項目。
- 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。
- 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用のリソースを表すために使用できます。JSON テキストは、出力データ項目が英数字データ項目の場合は UTF-8 でエンコードされ、出力データ項目が国別データ項目の場合は UTF-16 でエンコードされます。

### JSON GENERATE ステートメントの使用

以下の例を考えてください。

```
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]}}}
```



---

## 第 31 章 XML 入力の処理

XML PARSE ステートメントを使用すると、COBOL プログラム内で XML 入力を処理できます。

XML PARSE ステートメントは、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 コンパイラー・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 バージョン 3 およびバージョン 4 での XML 構文解析との互換性が提供されます。

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ー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。

- XML PARSE ステートメントは、XML 構文解析を開始して、ソース XML 文書および処理プロシージャーを識別します。

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
- 処理プロシージャーは、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ーの間で次のように情報を交換します。
  - XML-CODE は、XML 構文解析の状況を受け取り、時として、情報をパーサーに返します。
  - XML-EVENT は、各 XML イベントの名前をパーサーから受け取ります。
  - XML-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。

XML PARSE ステートメントの ENCODING 句、VALIDATING 句、および RETURNING NATIONAL 句は、XMLPARSE(XMLSS) が有効な場合にのみ使用できます。

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

『COBOL での XML パーサー』

#### 関連タスク

- 630 ページの『XML 文書へのアクセス』
- 630 ページの『XML 文書の構文解析』
- 654 ページの『XML PARSE の例外処理』
- 659 ページの『XML 構文解析の終了』

#### 関連参照

- 436 ページの『XMLPARSE』 (コンパイラ・オプション)
- 648 ページの『XML 文書のエンコード方式』
- 823 ページの『付録 C. XML 参照資料』

*Extensible Markup Language (XML)*

---

## COBOL での XML パーサー

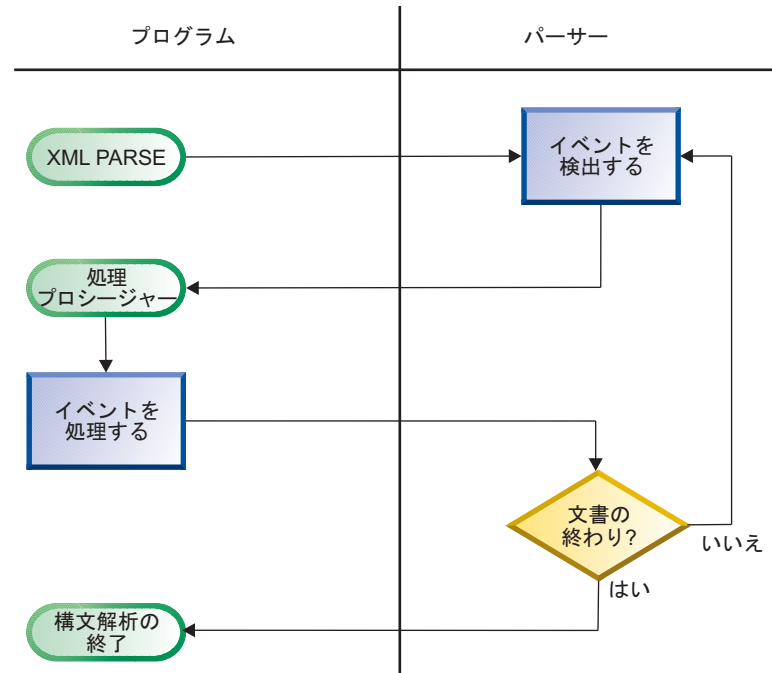
Enterprise COBOL ではイベント・ベースのインターフェースが提供されるため、これを使用して XML 文書を構文解析し、さらに COBOL データ構造に変換することができます。

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。

XML PARSE ステートメントを実行すると、構文解析が開始されてパーサーでの処理プロシージャが確立されます。パーサーは、文書の処理中に検出した XML イベントごとに、処理プロシージャに制御を渡します。イベントの処理後、処理プロシージャは自動的に制御をパーサーに返します。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継続されます。

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

- 643 ページの『XML スキーマ』
- 649 ページの『XML 入力文書エンコード』

#### 関連タスク

- 630 ページの『XML 文書の構文解析』
- 641 ページの『検証を伴う XML 文書の構文解析』
- 654 ページの『XML PARSE の例外処理』
- 659 ページの『XML 構文解析の終了』

#### 関連参照

648 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 PARSE ステートメントを使用して XML 文書を構文解析する前に、その文書をプログラムで使用できるようにしておく必要があります。XML 文書を取得する一般的な方法としては、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。

構文解析する XML 文書がファイル内に保管されている場合は、以下に示す通常の COBOL 機能を使用して文書をプログラムのデータ項目に入れてください。

- FILE-CONTROL 記入項目でプログラムに対してファイルを定義します。
- OPEN ステートメントでファイルをオープンします。
- READ ステートメントで、ファイルからすべてのレコードを読み取って、データ項目 (カテゴリー英数字またはカテゴリー国別の基本項目、あるいは英数字グループまたは国別グループ) に入れます。WORKING-STORAGE SECTION または LOCAL-STORAGE SECTION でデータ項目を定義できます。
- (オプション) STRING ステートメントで、個別レコードすべてを 1 つの連続ストリームに結合したり、無関係なブランクを除去したり、可変長レコードを処理したりします。

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

501 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』

529 ページの『第 22 章 COBOL プログラムの開発 (IMS の場合)』

644 ページの『XML 文書を 1 セグメントずつ構文解析』

#### 関連参照

436 ページの『XMLPARSE』 (コンパイラ・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。

```
XML PARSE xml-document
PROCESSING PROCEDURE xml-event-handler
ON EXCEPTION
 DISPLAY 'XML document error ' XML-CODE
 STOP RUN
NOT ON EXCEPTION
 DISPLAY 'XML document was successfully parsed.'
END-XML
```

XML PARSE ステートメントで、まず XML 文書文字ストリームを含む構文データ項目 (上の例では `xml-document`) を識別します。DATA DIVISION には、文書のエンコードが Unicode UTF-16 である場合には、カテゴリー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、以下のオプションで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行われるアクションを指示できます。

- ON EXCEPTION は、構文解析中に未処理の例外が発生した場合に制御を受け取ります。
- 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。END-XML を使用して、条件ステートメント内で ON EXCEPTION 句または NOT ON EXCEPTION 句を使用する XML PARSE ステートメントをネストできます。

パーサーは、XML イベントごとに処理プロシージャに制御を渡します。処理プロシージャの終わりに到達すると、制御はパーサーに返されます。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

- 634 ページの『XML イベント』
- 635 ページの『XML-CODE』
- 643 ページの『XML スキーマ』
- 637 ページの『XML-INFORMATION』

#### 関連タスク

- 『XML を処理するためのプロシージャの作成』
- 641 ページの『検証を伴う XML 文書の構文解析』
- 644 ページの『XML 文書を 1 セグメントずつ構文解析』
- 653 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 436 ページの『XMLPARSE』 (コンパイラ・オプション)

- 648 ページの『XML 文書のエンコード方式』
- 823 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』
- 825 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。

パーサーは、イベントを検出すると、特殊レジスター内の処理プロシージャに情報を渡します。これらの特殊レジスターのコンテンツは、COBOL データ構造の取り込みと、処理の制御に使用します。

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 64. 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 <sup>1</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | PICTURE X(30) USAGE DISPLAY VALUE SPACE | XML イベントの名前                                                                                                                                             |
| XML-CODE <sup>2</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 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 <sup>1</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 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 <sup>1</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 可変長基本カテゴリー英数字項目。                        | XML PARSE ID <sup>3</sup> 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 <sup>1</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | 可変長基本カテゴリー国別項目。                         | XML PARSE ID <sup>3</sup> 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 <sup>1, 4</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 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 <sup>3</sup>                                                        |
| XML-NNAMESPACE <sup>1, 4</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 可変長基本カテゴリー国別項目。                         | XML 文書が国別データ項目である場合、または RETURNING NATIONAL 句が XML PARSE ステートメントで指定されている場合の、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                |
| XML-NAMESPACE-PREFIX <sup>1, 4</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 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) <sup>3</sup>                                         |
| XML-NNAMESPACE-PREFIX <sup>1, 4</sup>                                                                                                                                                                                                                                                                                                                                                                                                                                                        | 可変長基本カテゴリー国別項目。                         | XML 文書が国別データ項目である場合、または RETURNING NATIONAL 句が XML PARSE ステートメントに指定されている場合の、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) |
| <ol style="list-style-type: none"> <li>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。</li> <li>XML GENERATE ステートメントでも XML-CODE が使用されます。したがって、処理プロシーチャー内に XML GENERATE ステートメントがある場合、XML GENERATE ステートメントの前に XML-CODE の値を保存し、XML GENERATE ステートメントの後に保存した値をリストアします。</li> <li>英数字データ項目で、RETURNING NATIONAL 句を XML PARSE ステートメントに指定した場合、テキストは対応する国別特殊レジスターで返されます。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。</li> <li>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。</li> </ol> |                                         |                                                                                                                                                         |

#### 制約事項:

- 処理プロシーチャーで、XML PARSE ステートメントを直接実行してはなりません。ただし、INVOKE または CALL ステートメントを使用して、処理プロシーチャーからメソッドまたは最外部のプログラムに制御が渡る場合、ターゲットとなるメソッドまたはプログラムは同一または別の XML PARSE ステートメントを実行できます。複数のスレッドで実行されているプログラムから、同一または別の XML ステートメントを同時に実行することもできます。
- 処理プロシーチャーの範囲で、GOBACK または EXIT PROGRAM ステートメントを実行してはいけません。ただし、制御がそれぞれ INVOKE または CALL ステートメント

メントで渡されたメソッドまたはプログラムから制御を返す場合を除きます。この場合は、処理プロシージャーの範囲で実行されます。

処理プロシージャーに 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、各処理プロシージ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。

660 ページの『例: XML の処理用プログラム』

#### 関連概念

『XML イベント』

635 ページの『XML-CODE』

638 ページの『XML-TEXT および XML-NTEXT』

638 ページの『XML-NAMESPACE および XML-NNAMESPACE』

639 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』

#### 関連タスク

644 ページの『XML 文書を 1 セグメントずつ構文解析』

641 ページの『検証を伴う XML 文書の構文解析』

659 ページの『XML 構文解析の終了』

#### 関連参照

436 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。
- XMLPARSE (XMLSS) が有効であれば、RETURNING NATIONAL 句が XML PARSE ステートメントに指定されている場合、または XML 文書が国別データ項目である場合には、パーサーは XML-NTEXT を設定します。それ以外の場合には、XML-TEXT を設定します。



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。

パーサーは、文書内でエンコード競合や整形式性または妥当性エラーを検出すると、XML-EVENT を 'EXCEPTION' に設定し、XML-CODE 特殊レジスターで例外に関する情報を追加します。(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。

660 ページの『例: 単純な文書の構文解析』

#### 関連概念

628 ページの『COBOL での XML パーサー』

『XML-CODE』

637 ページの『XML-INFORMATION』

638 ページの『XML-TEXT および XML-NTEXT』

638 ページの『XML-NAMESPACE および XML-NNAMESPACE』

639 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』

#### 関連タスク

632 ページの『XML を処理するためのプロシーチャーの作成』

641 ページの『検証を伴う XML 文書の構文解析』

#### 関連参照

436 ページの『XMLPARSE』 (コンパイラー・オプション)

823 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』

825 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チャーから 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、処理プロシーチャーで 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。

XMLPARSE(COMPAT) が有効な場合の EXCEPTION XML イベントでは、制御がパーサーに戻る前に、処理プロシーチャーが XML-CODE を意味がある値に設定できるケースがあります。(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プログラム・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 65. 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 <sup>1</sup> | 次のイベントは END-OF-DOCUMENT <sup>2</sup>        | 次のイベントは入力に依存 <sup>2</sup> | 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 <sup>1</sup> | 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 66. XMLPARSE(COMPAT) 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                   | -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 <sup>1</sup>       | 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 <sup>2</sup>        | 差 (上記) をエンコード値として使用して構文解析 <sup>2</sup> | 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 <sup>3</sup> | 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 <sup>4</sup> | [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    |

表 66. XMLPARSE(COMPAT) 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 (続き)

| XML イベント・タイプ                                                                                                                                                                       | -1                               | 0                     | XML-CODE-100,000    | その他のゼロ以外の値                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|-----------------------|---------------------|----------------------------------------------------------------------------------------------------------|
| START-OF-DOCUMENT                                                                                                                                                                  | 即時に終了。XML-CODE = -1 <sup>4</sup> | [XML-CODE に明らかな変更はなし] | 即時に終了。XML-CODE = -1 | <ul style="list-style-type: none"> <li>XML-CODE = 1</li> <li>そうでない場合は即時に終了。<br/>XML-CODE = -1</li> </ul> |
| 1. XMLPARSE(COMPAT) が有効な場合の XML PARSE 例外に関する関連参照の例外コードを参照してください。<br>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。<br>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。<br>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、XML GENERATE 例外の処理に関する関連タスクを参照してください。

#### 関連概念

656 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

632 ページの『XML を処理するためのプロシージャの作成』

644 ページの『XML 文書を 1 セグメントずつ構文解析』

654 ページの『XML PARSE の例外処理』

659 ページの『XML 構文解析の終了』

682 ページの『XML GENERATE 例外の処理』

#### 関連参照

823 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』

825 ページの『XMLPARSE(COMPAT) が有効な場合の XML PARSE 例外』

832 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

634 ページの『XML イベント』

635 ページの『XML-CODE』

#### 関連タスク

632 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

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
- ATTRIBUTE-NATIONAL-CHARACTER または CONTENT-NATIONAL-CHARACTER イベントが発生した

特殊レジスター XML-TEXT と XML-NTEXT は、相互に排他的です。パーサーが XML-TEXT を設定した場合、XML-NTEXT は空で長さゼロになります。パーサーが XML-NTEXT を設定した場合、XML-TEXT は空で長さゼロになります。

XML-NTEXT 内の文字エンコード・ユニットの数を決定するには、LENGTH 組み込み関数 (例: FUNCTION LENGTH(XML-NTEXT)) を使用します。XML-NTEXT 内のバイト数を決定するには、特殊レジスター LENGTH OF XML-NTEXT を使用します。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、処理プロシージャー外では未定義です。

#### 関連概念

634 ページの『XML イベント』

635 ページの『XML-CODE』

#### 関連タスク

632 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 436 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。

XML-NNAMESPACE 内の文字エンコード・ユニットの数を決定するには、LENGTH 組み込み関数 (例: FUNCTION LENGTH(XML-NNAMESPACE)) を使用します。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、処理プロシージャ外では未定義です。

#### 関連概念

634 ページの『XML イベント』

635 ページの『XML-CODE』

『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』

638 ページの『XML-TEXT および XML-NTEXT』

#### 関連タスク

632 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

436 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。

XML-NNAMESPACE-PREFIX 内の文字エンコード・ユニットの数を決定するには、LENGTH 組み込み関数 (例: FUNCTION LENGTH(XML-NNAMESPACE-PREFIX)) を使用しま

す。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、処理プロシージャ外では未定義です。

#### 関連概念

634 ページの『XML イベント』

638 ページの『XML-NAMESPACE および XML-NNAMESPACE』

#### 関連タスク

632 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

436 ページの『XMLPARSE』 (コンパイラ・オプション)

## XML テキストの COBOL データ項目への変換

XML データは固定長ではなく、固定形式でもないので、XML データを COBOL データ項目に移動するときは、特別な技法を使用する必要があります。

英数字項目の場合、XML データを COBOL 項目の左端 (デフォルト) に配置するか、右端に配置するかを決める必要があります。データが右端に配置する場合は、項目の定義に JUSTIFIED RIGHT 節を指定します。

数字の XML 値、特に、'\$1,234.00' または '\$1234' といった「修飾」された通貨値には特別な配慮が必要です。この 2 つのストリングは、XML で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

131 ページの『数値への変換 (NUMVAL、NUMVAL-C、NUMVAL-F)』

147 ページの『COBOL での国別データ (Unicode) の使用』

632 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。

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 フォーマットのスキーマを生成するには、z/OS UNIX コマンドの `xsdosrg` を使用します。これは、z/OS システム・サービスが提供する OSR 生成プログラムを起動します。(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 UNIX 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依じて、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 UNIX 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 `DDSCHEMA` 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 `DDSCHEMA` を z/OS UNIX 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 `DDSCHEMA` は、MVS データ・セットを指定した `DSN` オプション、または z/OS UNIX 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: `FILE` キーワードを使用した XML 検証は、CICS 下ではサポートされません。

`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が、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。以下に例を示します。

```
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形式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、処理プロシージャに制御を渡します。

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。

671 ページの『例: 検証を伴う XML 文書の構文解析』

#### 関連概念

635 ページの『XML-CODE』  
『XML スキーマ』

#### 関連タスク

654 ページの『XML PARSE の例外処理』

#### 関連参照

436 ページの『XMLPARSE』 (コンパイラー・オプション)  
823 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します。

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 タイプの必須の `itemNumber` 属性があり (行 16)、その他の次の順序のエレメント (行 6 から 15) が含まれています。

- 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)。

行 9 から 13 のように、タイプ宣言は、インラインで名前がないものにすることができます。ここでは、`maxExclusive` ファセットで `quantityOnHand` エレメントに有効な値を指定しています。

一方、`itemNumber` 属性については、名前付きタイプの SKU が行 19 から 23 に独立して宣言されています。ここでは、正規表現構文を使用するパターン・ファセットによって、そのタイプに有効な値が、3 つの数字、ハイフン、2 つの大文字から (この順序で) 構成されることを指定しています。

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。

671 ページの『例: 検証を伴う XML 文書の構文解析』

関連タスク

641 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ジャに返します。

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、処理プロシージャで 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。

- OCCURS DEPENDING ON 節を含んだ、可変長グループ項目
- 参照変更された項目
- RECORD IS VARYING DEPENDING ON 節を指定した FD レコード。ここで、依存データ項目は FD レコードの参照修飾子または ODO オブジェクトで長さとして使用されます)

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、100 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ジャ・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 (各行が XML 文書の 1 セグメント) では、テキスト `This comment ends this segment` が入っているセグメントが構文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-->

```

669 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

634 ページの『XML イベント』

635 ページの『XML-CODE』

#### 関連タスク

644 ページの『XML 文書を 1 セグメントずつ構文解析』

635 ページの『XML-CODE』

#### 関連参照

436 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 660 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ー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 ステートメントによって、"filling" ID に割り当てるための完全な文字値が適切に累積され、記述されています。

```
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

634 ページの『XML イベント』

635 ページの『XML-CODE』

#### 関連参照

436 ページの『XMLPARSE』 (コンパイラー・オプション)

660 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

XMLPARSE(XMLSS) が有効である場合、XML PARSE ステートメントの ENCODING 句には、(上述の XML PARSE での説明したとおり) サポートされた任意の CCSID を使用することができます。

表 67. XML 文書のコード化文字セット

CCSID	説明
1208	UTF-8 <sup>1</sup>
1047	Latin 1 / オープン・システム
1140, 37	USA、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. XMLPARSE(XMLSS) が有効である場合に、ENCODING 句の XML PARSE ステートメントでサポートされます	

#### 関連概念

『XML 入力文書エンコード』

#### 関連タスク

651 ページの『エンコードの指定』

653 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

675 ページの『第 32 章 XML 出力の生成』

#### 関連参照

359 ページの『CODEPAGE』

436 ページの『XMLPARSE』 (コンパイラー・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 コンパイラー・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) コンパイラー・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。

- RETURNING NATIONAL 句が XML PARSE ステートメントで指定されている場合:  
UTF-8、または 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 68. 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する必要があります。

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
- XML PARSE ステートメントの ENCODING 句 (使用されている場合)
- 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

- 156 ページの『国別 (Unicode) 表現との間の変換』
- 『エンコードの指定』
- 653 ページの『UTF-8 でエンコードされた XML 文書の構文解析』
- 654 ページの『XML PARSE の例外処理』

#### 関連参照

- 436 ページの『XMLPARSE』 (コンパイラー・オプション)
- 648 ページの『XML 文書のエンコード方式』
- 652 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。

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。以下に例を示します。

```
<?xml version="1.0" encoding="ibm-1140"?>
```

XML パーサーは、先頭バイトが XML 宣言で開始されていない XML 文書を検出すると例外を生成します。

エンコード宣言は、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の混合は自由) 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 69. 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
1208	UTF-8

XML 構文解析にサポートされている CCSID については、XML 文書のエンコードに関する関連参照を参照してください。

**関連概念**

649 ページの『XML 入力文書エンコード』

**関連タスク**

653 ページの『UTF-8 でエンコードされた XML 文書の構文解析』

658 ページの『エンコード競合の処理』

**関連参照**

648 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 70. 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'

表 70. さまざまな EBCDIC CCSID 用特殊文字の 16 進値 (続き)

文字	1047	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149
]	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

XMLPARSE(XMLSS) コンパイラー・オプションが有効である場合、Unicode UTF-8 でエンコードされた XML 文書を、他の XML 文書と同じように構文解析することができます。ただし、追加でいくつかの要件が適用されます。

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。

XML PARSE ステートメントで RETURNING NATIONAL 句をコーディングしないと、パーサーは、英数字特殊レジスター XML-TEXT、XML-NAMESPACE、および XML-NAMESPACE-PREFIX で XML 文書のフラグメントを返します。

UTF-8 文字 1 文字は、可変個のバイト数でエンコードされます。英数字データに対するほとんどの COBOL 操作では、1 バイトのエンコード (1 文字が 1 バイトでエンコードされるエンコード) を想定しています。UTF-8 文字を英数字データとして操作する場合、データが正常に処理されるようにする必要があります。マルチバイト文字のバイトを分割する可能性がある操作 (参照変更や切り捨てを呼び出す移動操作など) は避けてください。英数字データのマルチバイト文字を処理するために、INSPECT などのステートメントを信頼して使用することはできません。

UTF-8 文書フラグメントの処理は、XML PARSE ステートメントで RETURNING NATIONAL 句を指定することによって信頼性を高めることができます。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。)

次のコードの断片では、UTF-8 XML 文書の構文解析における ENCODING 句と RETURNING NATIONAL 句、両方の使用方法を説明します。

```
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

- 638 ページの『XML-TEXT および XML-NTEXT』
- 638 ページの『XML-NAMESPACE および XML-NNAMESPACE』
- 639 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』

#### 関連タスク

- 159 ページの『UTF-8 データの処理』
- 630 ページの『XML 文書の構文解析』
- 651 ページの『エンコードの指定』

#### 関連参照

- 436 ページの『XMLPARSE』 (コンパイラー・オプション)
- 648 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。発生する個々の例外コードと、それに続いて実行できるアクションは、XMLPARSE コンパイラー・オプションの設定によって異なります。

#### 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` と一致していないため、正しい構成ではありません。

```
<msg>Hello</mmsg>
```

検証を行わずに文書を構文解析した場合、戻りコードは 16 進数の 000C (XRC\_NOT\_WELL\_FORMED) で、理由コードは 16 進数の 3035 (XRSN\_ENDTAG\_NAME\_MISMATCH) です。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、整形式エラーに関して 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、XML PARSE ステートメントの ON EXCEPTION 句で指定したステートメントに制御を渡します。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。

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。

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。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渡されます。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。

例外コードが上記の範囲内にないすべての例外で、パーサーはこれ以上のイベントをシグナル通知しませんが、ON EXCEPTION 句に指定されたステートメントに制御を渡します。パーサーに制御を返す前に処理プロシージャで XML-CODE を設定したとしても、XML-CODE には元の例外コードが含まれます。

例外を処理する必要がない場合は、XML-CODE の値を変更しないでパーサーに制御を戻します。パーサーは、ON EXCEPTION 句で指定されたステートメントに制御を移します。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

- 635 ページの『XML-CODE』
- 649 ページの『XML 入力文書エンコード』
- 『XML パーサーによるエラーの処理方法』

#### 関連タスク

- 632 ページの『XML を処理するためのプロシージャの作成』
- 641 ページの『検証を伴う XML 文書の構文解析』
- 658 ページの『エンコード競合の処理』

#### 関連参照 436 ページの『XMLPARSE』 (コンパイラ・オプション)

- 648 ページの『XML 文書のエンコード方式』
- 823 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』
- 825 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。

例外コードは、XML PARSE 例外に関する関連参照に記載されています。

- 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。

すべての他の XML 特殊レジスターは空で、長さゼロです。

#### 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。処理プロシージャからパーサーに戻ると、パーサーは、ON EXCEPTION 句が指定されている場合はその句に制御を渡します。指定されていない場合には、パーサーは XML PARSE ステートメントの最後に制御を渡します。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。

#### 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

- 634 ページの『XML イベント』
- 635 ページの『XML-CODE』
- 649 ページの『XML 入力文書エンコード』

#### 関連タスク

- 644 ページの『XML 文書を 1 セグメントずつ構文解析』
- 654 ページの『XML PARSE の例外処理』
- 658 ページの『エンコード競合の処理』
- 659 ページの『XML 構文解析の終了』

関連参照 436 ページの『XMLPARSE』 (コンパイラー・オプション)

- 648 ページの『XML 文書のエンコード方式』

823 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』  
825 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。

### 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。構文解析データ項目が英数字であり、XML-CODE の例外コードが 100,001 から 165,535 の範囲にある例外イベントは、(エンコード宣言で指定された) 文書のコード・ページが、外部コード・ページ情報と矛盾していることを示しています。

この特殊ケースでは、XML-CODE の値から 100,000 を減算することで、文書のコード・ページを使用して構文解析を行うようにすることができます。例えば、XML-CODE が 101,140 に設定されている場合、文書のコード・ページは 1140 です。別の方法では、パーサーに戻る前に XML-CODE をゼロに設定して、外部コード・ページを使用して構文解析することもできます。

パーサーは、コード・ページ矛盾の例外イベント用の処理プロシージャから戻ると、以下の 3 つの処置のいずれかをとります。

- XML-CODE をゼロに設定した場合、パーサーは、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

- 635 ページの『XML-CODE』
- 649 ページの『XML 入力文書エンコード』
- 656 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

- 654 ページの『XML PARSE の例外処理』



#### 関連参照

- 436 ページの『XMLPARSE』 (コンパイラー・オプション)
  - 648 ページの『XML 文書のエンコード方式』
  - 823 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』
  - 825 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

プロシージャーが通常の 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、処理プロシージャーで 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。

この技法は、処理プロシージャー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ー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。制御は、XML PARSE ステートメントの ON EXCEPTION 句が指定されている場合にはその句に渡されます。

ON EXCEPTION 句の命令ステートメントでは、XML-CODE 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。

また、XMLPARSE(COMPAT) コンパイラー・オプションが有効である場合には、XML-CODE の値を変更せずに処理プロシージャーからパーサーに戻ることによって、XML EXCEPTION イベントの発生後に構文解析を終了することができます。この場合、結果は意図的に終了した場合と似ていますが、XML-CODE に元の例外コードが入った状態でパーサーが XML PARSE ステートメントに戻る点は除きます。

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

- 635 ページの『XML-CODE』
- 656 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

- 632 ページの『XML を処理するためのプロシージャーの作成』
- 654 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。

以下の例を使用して、XML PARSE の基本的な使用方法、および XMLPARSE(XMLSS) の特殊な使用方法 (名前空間を含む文書の構文解析、文書の 1 セグメントずつの構文解析、およびスキーマと照合する検証を伴った文書の構文解析など) を示します。

- 『例: 単純な文書の構文解析』
- 『例: XML の処理用プログラム』
- 665 ページの『例: 名前空間を使用する XML 文書の構文解析』
- 669 ページの『例: XML 文書を 1 セグメントずつ構文解析』
- 671 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>
```

次のコードの断片では、Doc を構文解析する XML PARSE ステートメント、および XML イベントを処理する処理プロシージャ P を示します。

```
XML Parse Doc
Processing procedure P
. . .
P. Display XML-Event XML-Text.
```

処理プロシージャでは、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 71. 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

- 634 ページの『XML イベント』
- 638 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ジャを示しています。

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。例の後に、XMLPARSE(XMLSS) と XMLPARSE(COMPAT) が有効な場合のプログラムの出力が示されています。

パーサーと処理プロシージャ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'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 '*****+-----'
 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.

```

## XMLPARSE(XMLSS) が有効な構文解析の出力

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'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 & 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!

```

## XMLPARSE (COMPAT) が有効な構文解析の出力

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

634 ページの『XML イベント』

#### 関連タスク

647 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

436 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 사용할 수 있습니다. author 의 title (作家的称号: Mr) 以及 book 의 title (本의 타이틀: Writing COBOL for Fun and Profit) として title 가 사용されています.

## 샘플 XML 문서

次の XML 문서에는, 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lement comment で、デフォルト名前空間が空字符串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、処理プロシージャラー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 72. 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	섹션		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
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



表 72. 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
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	セクション		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:localE1Name1>'.
 2 pic x(20) value '<pfx2:localE1Name2/>'.
 2 pic x(40) value '<pfx3:localE1Name3 pfx4:localAtName4="">'.
 2 pic x(02) value 'c1'.
 2 pic x(41) value '<pfx5:localE1Name5 pfx6:localAtName6=""/>'.
 2 pic x(24) value 'c2</pfx3:localE1Name3>c3'.
 2 pic x(32) value '</pfx1:localE1Name1></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 '|'
 xml-namespace-prefix '|'
 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、処理プロシージャラー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 73. 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		
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

634 ページの『XML イベント』

638 ページの『XML-TEXT および XML-NTEXT』

638 ページの『XML-NAMESPACE および XML-NNAMESPACE』

639 ページの『XML-NAMESPACE-PREFIX および XML-NNAMESPACE-PREFIX』

関連参照

436 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に制御を渡します。処理プロシージャ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に渡します。処理プロシージャは次の 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、処理プロシーダは入力各レコードを表示し、その後一連の 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

436 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ジャを示します。すべての XML 文書は整形形式の文書ですが、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。

item.xsd ファイルには、スキーマがテキスト形式で含まれ、次の z/OS UNIX コマンドによって、前処理済みのスキーマが item.osr ファイルに生成されているとします。

```
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 ステートメントでは、スキーマが入っている外部 z/OS UNIX ファイルに DD 名を関連付けます。

```
//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 rsn 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、処理プロシーチャー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

635 ページの『XML-CODE』

643 ページの『XML スキーマ』

### 関連タスク

641 ページの『検証を伴う XML 文書の構文解析』

654 ページの『XML PARSE の例外処理』

### 関連参照

823 ページの『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』





---

## 第 32 章 XML 出力の生成

XML GENERATE ステートメントを使用して、COBOL プログラムから XML 出力を生成させることができます。

XML GENERATE ステートメントでは、ソースおよび出力データ項目を指定します。オプションとして、次のものも指定できます。

- 生成された XML 文字のカウントを受け取るフィールド
- 生成された XML 文書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、メッセージとして WebSphere MQ へ渡したり、後で変換するために CICS 通信域へ伝送したりできます。

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

『XML 出力の生成』

681 ページの『生成される XML 出力のエンコードの制御』

682 ページの『XML GENERATE 例外の処理』

687 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。

```

XML GENERATE XML-OUTPUT FROM SOURCE-REC
COUNT IN XML-CHAR-COUNT
ON EXCEPTION
 DISPLAY 'XML generation error ' XML-CODE
 STOP RUN
NOT ON EXCEPTION
 DISPLAY 'XML document was successfully generated.'
END-XML

```

XML GENERATE ステートメントでは、XML 出力を受け取るデータ項目 (上の例では XML-OUTPUT) をまず識別します。データ項目は、生成された XML 出力を格納できる十分な大きさに定義します。通常、データ名の長さに応じて、COBOL ソース・データ・サイズの 5 倍から 10 倍に定義します。

DATA DIVISION では、受信 ID を、英数字 (英数字グループ項目またはカテゴリ・英数字の基本項目のどちらか) として、あるいは国別 (国別グループ項目またはカテゴリ・国別の基本項目のどちらか) として定義できます。

次に、XML フォーマットに変換されるソース・データ項目 (この例では SOURCE-REC) を識別します。ソース・データ項目は、英数字グループ項目、国別グループ項目、あるいはクラス英数字または国別の基本データ項目にすることができます。

COBOL データ項目の中には、XML に変換されず無視されるものがあります。XML に変換する英数字グループ項目または国別グループ項目の従属データ項目は、次のような場合は無視されます。

- REDEFINES 節を指定しているか、またはそのような再定義項目に従属しているもの。
- RENAMES 節を指定しているもの。

ソース・データ項目の中の次のような項目も、XML の生成時に無視されます。

- 基本 FILLER (または名前なしの) データ項目
- SYNCHRONIZED データ項目で挿入されている遊びバイト

XML を読みやすくするために余分な空白文字 (例えば、改行または字下げ) が挿入されることはありません。

必要に応じて、COUNT IN 句をコーディングして、XML 出力の生成時に充てんされる XML 文字エンコード・ユニット数を取得できます。受け取る ID のカテゴリが国別である場合、カウントは、UTF-16 文字エンコード・ユニット数です。すべての他のエンコード (UTF-8 を含む) では、カウントはバイト数です。

カウント・フィールドを参照変更長として使用して、生成された XML 出力を含む受け取りデータ項目の一部のみを取得できます。例えば、XML-OUTPUT(1:XML-CHAR-COUNT) は、XML-OUTPUT の最初の XML-CHAR-COUNT 文字位置を参照します。

次のプログラムの抜粋を検討します。

```

01 doc pic x(512).
01 docSize pic 9(9) binary.
01 G.
 05 A pic x(3) value "aaa".
 05 B.
 10 C pic x(3) value "ccc".

```

```

 10 D pic x(3) value "ddd".
05 E pic x(3) value "eee".
 .
 .
 .
XML Generate Doc from G

```

上のコードによって、次の XML 文書が生成されます。ここで、A、B、および E は、エレメント G の子エレメントとして表され、C および D は、エレメント B の子エレメントになります。

```
<G><A>aaa<C>ccc</C><D>ddd</D><E>eee</E></G>
```

また、XML GENERATE ステートメントの ATTRIBUTES 句を指定することもできます。ATTRIBUTES 句を使用すると、生成された XML 文書に含まれる適格なデータ項目はすべて、それを含む XML エレメントの子エレメントとしてではなく、それを含む XML エレメント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ない、データ項目の名前は FILLER 以外でなければならない、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。オプションで、TYPE OF 句を使用して、どのデータ項目を属性またはエレメントとして表現するか、より詳細な制御を指定できます。

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"></G>
```

オプションとして、XML GENERATE ステートメントの ENCODING 句をコーディングして、生成される XML 文書の CCSID を指定できます。ENCODING 句を使用しなかった場合、文書エンコードは受け取りデータ項目のカテゴリおよび CODEPAGE コンパイラー・オプションによって決まります。詳細については、生成された XML 出力のエンコードの制御に関する下記の関連タスクを参照してください。

オプションとして、XML-DECLARATION 句をコーディングして、生成された XML 文書にバージョン情報およびエンコード宣言を含んだ XML 宣言を組み込むことができます。受け取りデータ項目のカテゴリによって次のようになります。

- 国別の場合: エンコード宣言には値 UTF-16 が含まれます (encoding="UTF-16")。
- 英数字の場合: エンコード宣言は、ENCODING 句 (指定されている場合) またはプログラムで有効になっている CODEPAGE コンパイラー・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、下記のプログラムの抜粋では、XML GENERATE の XML-DECLARATION 句を指定して、エンコードを CCSID 1208 (UTF-8) で指定しています。

```

01 Greeting.
05 msg pic x(80) value 'Hello, world!'.
 .
 .
 .
XML Generate Doc from Greeting
 with Encoding 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、カテゴリーが国別または英数字の ID またはリテラルで指定します。

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、次のデータ定義および XML GENERATE ステートメントを検討してみましょう。

```
01 Greeting.
 05 msg pic x(80) value 'Hello, world!'.
01 NS pic x(20) value 'http://example'.
 . . .
 XML Generate Doc from Greeting
 namespace is NS
```

次に示すとおり、結果として得られる XML 文書は、デフォルト名前空間 (<http://example>) を持ちます。

```
<Greeting xmlns="http://example"><msg>Hello, world!</msg></Greeting>
```

名前空間を指定しなければ、生成される XML 文書内のエレメント名は、どの名前空間にも属しません。

オプションとして、NAMESPACE-PREFIX 句をコーディングして、生成される文書内の各エレメントの開始および終了タグに適用する接頭部を指定することもできます。接頭部は、上述のように名前空間を指定した場合にのみ指定できます。

XML GENERATE ステートメントを実行する場合、接頭部の値はコロン (:) なしの有効な XML 名でなければなりません。詳しくは、名前空間に関する下記の関連参照を参照してください。値には末尾スペースを含めることができますが、接頭部の使用前に除去されます。

名前空間接頭部は、カテゴリーが国別または英数字の ID またはリテラルで指定します。

接頭部は、各エレメントの開始および終了タグを修飾するため、短くすることを推奨します。

例えば、次のデータ定義および XML GENERATE ステートメントを検討してみましょう。

```
01 Greeting.
 05 msg pic x(80) value 'Hello, world!'.
01 NS pic x(20) value 'http://example'.
```

```
01 NP pic x(5) value 'pre'.
```

```
XML Generate Doc from Greeting
namespace is NS
namespace-prefix is NP
```

次のように、結果として得られる XML 文書は明示的な名前空間 (<http://example>) を持ち、接頭部 `pre` がエレメント `Greeting` および `msg` の開始および終了タグに適用されます。

```
<pre:Greeting xmlns:pre="http://example"><pre:msg>Hello, world!</pre:msg></pre:Greeting>
```

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ない、XML 1.0 標準に従う正式名でなければならない。

例えば、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 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</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、次のデータ構造および XML GENERATE ステートメントを検討してみましょう。

```
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。条件ステートメントで ON EXCEPTION または NOT ON EXCEPTION 句を指定している XML GENERATE ステートメントをネストするには、END-XML をコーディングします。

XML への COBOL ソース・レコードの変換が完了するか、またはエラーが発生するまで、XML 生成は継続します。エラーが発生した場合、結果は次のようになります。

- XML-CODE 特殊レジスターには、ゼロ以外の例外コードが含まれます。
- ON EXCEPTION 句が指定されている場合、これに制御が渡されます。指定されていない場合は、XML GENERATE ステートメントの最後に制御が渡されます。

XML 生成時にエラーが発生しなかった場合、XML-CODE 特殊レジスターにはゼロが入り、制御は、NOT ON EXCEPTION 句が指定されている場合はこの句に、指定されていない場合は XML GENERATE ステートメントの最後に渡されます。

682 ページの『例: XML の生成』

関連概念

*Uniform Resource Identifier (URI): Generic Syntax*

関連タスク

681 ページの『生成される XML 出力のエンコードの制御』

682 ページの『XML GENERATE 例外の処理』

159 ページの『UTF-8 データの処理』

#### 関連参照

XML GENERATE ステートメント (*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 GENERATE ステートメントを使用して XML 出力を生成する場合、出力を受け取るデータ項目のカテゴリによって、および XML GENERATE ステートメントの WITH ENCODING 句を使用してコード・ページを指定することによって、出力のエンコードを制御することができます。

WITH ENCODING *codepage* 句を指定して出力文書のコード化文字セット ID (CCSID) を指定した場合、*codepage* には、XML 文書のエンコードに関する下記の関連参照で説明されている COBOL XML 処理でサポートされているコード・ページのいずれかを指定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。
- 受け取る ID のカテゴリが英数字の場合、WITH ENCODING 句には 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 74. 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有効な CODEPAGE コンパイラ・オプションで指定されたコード・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

359 ページの『CODEPAGE』

648 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 出力の生成中にエラーが検出された場合は、例外条件が存在します。エラー・タイプを示す数値の例外コードが格納される、XML-CODE 特殊レジスターを検査するコードを記述することができます。

エラーを処理するには、XML GENERATE ステートメントの以下の句の一方または両方を使用してください。

- ON EXCEPTION
- COUNT IN

XML GENERATE ステートメントに ON EXCEPTION 句をコーディングした場合、指定された命令ステートメントに制御が転送されます。命令ステートメントをコーディングして、例えば、XML-CODE 値を表示できます。ON EXCEPTION 句がコーディングされていない場合、制御は XML GENERATE ステートメントの終わりに移動します。

エラーが発生した場合は、XML 出力を受け取るデータ項目が十分に大きくないことが問題である可能性があります。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。

以下の手順を実行して、生成された XML 出力を調べることができます。

1. XML GENERATE ステートメントに COUNT IN 句をコーディングしてください。

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。

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リストについては、下記の関連参照を参照してください。

関連タスク

124 ページの『データ項目のサブストリングの参照』

関連参照

832 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
.
println.
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

627 ページの『第 31 章 XML 入力の処理』

関連参照 436 ページの『XMLPARSE』（コンパイラ・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 フォーマットで表したい情報が既に DATA DIVISION のグループ項目に存在しているが、1 つ以上の要因のためその項目を使用して XML 文書を直接生成できないおそれがあります。

以下に例を示します。

- 必要データのほかに、項目には、XML 出力文書とは無関係な値を含んでいる従属データ項目が含まれています。
- 必要データ項目の名前が、外部表示には不適当なものであり、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。

『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 ( 682 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 出力のこのような特性は、以下のように XML GENERATE ステートメントの句を追加でを使用することにより修正できます。

- 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 33 章 オブジェクト指向プログラムの作成

オブジェクト指向 (OO) プログラムを書く際には、必要とするクラス、およびクラスが作業を行うのに必要なメソッドとデータを決定する必要があります。

OO プログラムは、オブジェクト (状態と動作をカプセル化するエンティティー) ならびにオブジェクトのクラス、メソッド、およびデータに基づいています。クラスは、オブジェクトの状態および機能を定義するテンプレートです。通常、プログラムは、あるクラスの複数のオブジェクト・インスタンス (または単にインスタンス)、つまりそのクラスのメンバーである複数のオブジェクトを作成し、それを扱う仕事をします。それぞれのインスタンスの状態はインスタンス・データと呼ばれるデータに保管され、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 を使用して、以下のことを行うことができます。

- メソッドとデータを COBOL でインプリメントした状態で、クラスを定義する。
- Java および COBOL クラスのインスタンスを作成する。
- Java および COBOL オブジェクトにメソッドを呼び出す。
- Java クラスまたはほかの COBOL クラスから継承するクラスを書き込む。
- 多重定義メソッドを定義して呼び出す。

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。

Enterprise COBOL クラスでは、CALL ステートメントをコーディングして、プロシージャー型 COBOL プログラムとインターフェースを取ることができます。したがって、COBOL クラス定義構文は、プロシージャー型 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 UNIX 環境で行うことが推奨されます。

制約事項:

- COBOL クラス定義およびメソッドは、EXEC SQL ステートメントを含むことができず、SQL コンパイラー・オプションを使用してコンパイルすることもできません。

- COBOL クラス定義およびメソッドは、EXEC SQLIMS ステートメントを含むことができず、SQLIMS コンパイラー・オプションを使用してコンパイルすることもできません。
- 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 コンパイラー・オプションを使用してコンパイルすることができません。

『例: 口座』

#### 関連タスク

697 ページの『クラスの定義』

702 ページの『クラス・インスタンス・メソッドの定義』

711 ページの『クライアントの定義』

724 ページの『サブクラスの定義』

728 ページの『ファクトリー・セクションの定義』

333 ページの『第 16 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

顧客が口座を開き、その口座で預金や引き出しを行うことができる、銀行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。

必要とするクラスを判別したら、次のステップでは、それらのクラスがそれぞれの作業を実行するために必要なメソッドを判別します。Account クラスは以下のサービスを提供する必要があります。

- 口座を開設する。
- 現在の収支を取る。
- 口座に預金する。
- 口座から預金を引き出す。
- 口座状況を報告する。

Account クラスの以下のメソッドは、上記の要件を満たします。

**init**     口座を開設し、それに口座番号を割り当てます。

**getBalance**

      口座の現在の収支を戻します。

**credit**

      指定の金額を口座に預金します。

## debit

指定の金額を口座から引き出します。

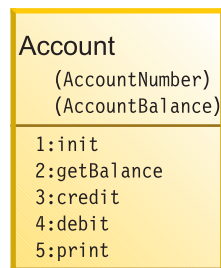
**print** 口座番号と勘定残高を表示します。

**Account** クラスとそのメソッドを設計すると、クラスはいくつかのインスタンス・データを保持する必要があることがわかります。一般に、**Account** オブジェクトには次のようなインスタンス・データが必要です。

- 口座番号
- 勘定残高
- カスタマー情報: 名前、住所、自宅の電話番号、勤務先電話番号、社会保障番号など

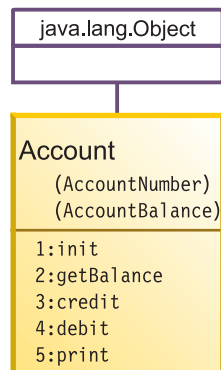
ただし、上記の例を単純化するため、口座番号と勘定残高は、**Account** クラスが必要とする唯一のインスタンス・データであると想定します。

クラスやメソッドを設計するとき、ダイアグラムは役に立ちます。次のダイアグラムで、**Account** クラスの設計における最初の試みを示します。



ダイアグラムにおいて、括弧内のワードはインスタンス・データの名前です。番号とコロンに続くワードは、インスタンス・メソッドの名前です。

以下の構造は、クラスの相互関係を示しており、継承の階層と呼ばれています。**Account** クラスはクラス `java.lang.Object` から直接継承します。



## サブクラス

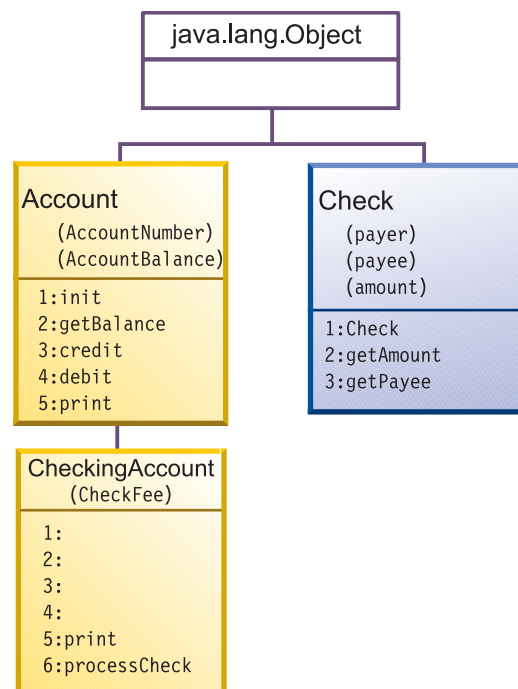
口座の例では、**Account** は汎用クラスです。ただし、銀行は、当座預金、普通預金、住宅ローンなど、多くの種類の口座を用意することができます。これらの口座

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当座ごとに適用される当座手数料が含まれる場合があります。また、**CheckingAccount** クラスには、当座を処理する (つまり、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** クラスを策定すると、当座をモデル化するクラスの必要性がわかります。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口座システムでは多数の追加クラス (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) を策定する必要があると思われますが、ここでは例を単純に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、1 つの親だけを持っていなければなりません。**java.lang.Object** は、すべての継承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。

709 ページの『例: メソッドの定義』

関連タスク

『クラスの定義』

702 ページの『クラス・インスタンス・メソッドの定義』

724 ページの『サブクラスの定義』

## クラスの定義

COBOL クラス定義は、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION、その後にオプションのファクトリー定義とオプションのオブジェクト定義、さらにその後に END CLASS マーカーが続く構成となっています。

表 75. クラス定義の構成

セクション	目的	構文
IDENTIFICATION DIVISION (必須)	クラスの名前。継承情報を提供する。	698 ページの『クラス定義用の CLASS-ID 段落』 (必須) AUTHOR 段落 (オプション) INSTALLATION 段落 (オプション) DATE-WRITTEN 段落 (オプション) DATE-COMPILED 段落 (オプション)
ENVIRONMENT DIVISION (必須)	コンピューター環境を記述する。クラス定義内で使用されるクラス名を、コンパイル単位の外側で判明している、対応する外部クラス名に関連付ける。	CONFIGURATION SECTION (必須) 699 ページの『クラス定義用の REPOSITORY 段落』 (必須) SOURCE-COMPUTER 段落 (オプション) OBJECT-COMPUTER 段落 (オプション) SPECIAL-NAMES 段落 (オプション)
ファクトリー定義 (オプション)	クラスのすべてのインスタンスが共有するデータとオブジェクト・インスタンスとは別々にサポートされるメソッドを定義する。	IDENTIFICATION DIVISION. FACTORY. DATA DIVISION. WORKING-STORAGE SECTION. * (Factory data here) PROCEDURE DIVISION. * (Factory methods here) END FACTORY.
オブジェクト定義 (オプション)	インスタンス・データとインスタンス・メソッドを定義する。	IDENTIFICATION DIVISION. OBJECT. DATA DIVISION. WORKING-STORAGE SECTION. * (Instance data here) PROCEDURE DIVISION. * (Instance methods here) END OBJECT.

SOURCE-COMPUTER、OBJECT-COMPUTER、または SPECIAL-NAMES 段落をクラス CONFIGURATION SECTION に指定すると、それらの段落は、そのクラスが導入するすべてのメソッドを含む、クラス定義全体に適用されます。

クラス CONFIGURATION SECTION は、プログラム CONFIGURATION SECTION と同じ記入項目で構成されています。ただし、クラス CONFIGURATION SECTION には INPUT-OUTPUT SECTION を含めることはできません。INPUT-OUTPUT SECTION の定義は、クラス・レベルでその定義を行うのではなく、それを必要とする個々のメソッドにおいてのみ行います。

上記で説明したように、インスタンス・データとメソッドの定義は、そのクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION において、行います。個別のオブジェクト・インスタンスではなく、クラス自体に関連付けるデータとメソッドを必要とするクラスに、クラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

各 COBOL クラス定義は、別々のソース・ファイルになければなりません。

701 ページの『例: クラスの定義』

関連タスク

701 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

702 ページの『クラス・インスタンス・メソッドの定義』

724 ページの『サブクラスの定義』

728 ページの『ファクトリー・セクションの定義』

6 ページの『コンピューター環境の記述』

333 ページの『第 16 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

クラスを指定し、それに継承情報を提供するには、IDENTIFICATION DIVISION の CLASS-ID 段落を使用してください。

Identification Division.            必須  
Class-id. Account inherits Base.   必須

以下のクラスを識別するには、CLASS-ID 段落を使用してください。

- 定義中のクラス (上記の例の Account)。
- 定義中のクラスがその特性を継承する元の即時スーパークラス。スーパークラスは、Java または COBOL でインプリメントされます。

上記の例において、inherits Base では、Account クラスは、クラス定義内で Base として認識されているクラスからメソッドとデータを継承することを示します。オブジェクト指向 COBOL プログラムにおいて、名前 Base は java.lang.Object を参照するために使用することをお勧めします。

クラス名では 1 バイト文字を使用する必要があり、クラス名は COBOL ユーザー定義語の通常の形成規則に準拠している必要があります。

ENVIRONMENT DIVISION の CONFIGURATION SECTION で REPOSITORY 段落を使用し、スーパークラス名 (例の Base) を外部に判明しているスーパークラス名 (Base の場合の java.lang.Object) に関連付けます。また、オプションとして、定義中のクラスの名前 (例の Account) を REPOSITORY 段落に指定し、その対応する外部クラス名にそれに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

『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に宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

外部クラス名は大/小文字が区別されます。したがって、Java 形成規則に準拠しなければなりません。例えば、Account クラス定義で以下のようにコーディングします。

Environment Division.	必須
Configuration Section.	必須
Repository.	必須
Class Base is "java.lang.Object"	必須
Class Account is "Account".	オプション

REPOSITORY 段落記入項目では、クラス定義内で Base および Account として参照されるクラスの外部クラス名は、それぞれ、java.lang.Object および Account であることが示されます。

REPOSITORY 段落では、クラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。以下に例を示します。

- ベース
- 定義中のクラスが継承する元のスーパークラス
- クラス定義内のメソッドで参照するクラス

REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定しなければなりません。Java パッケージの一部である参照クラスごとに外部クラス名も指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド (.) が付き、続いてその後に Java クラスの単純名が付きます。例えば、Object クラスは java.lang パッケージの一部です。したがって、上記に示したように、その外部名を java.lang.Object として指定します。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形成規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名は以下の方法でクラス名から作成されます。

- クラス名は大文字に変換されます。
- 各ハイフンはゼロに変更されます。
- 先頭文字が数字の場合は、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。

『例: 外部クラス名および Java パッケージ』

関連タスク

749 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
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

699 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**

COBOL クラスが必要とする インスタンス・データ、つまりクラスの各インスタンスに割り当てられるデータを記述するには、OBJECT 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。

IDENTIFICATION DIVISION 宣言の直前に入れる必要がある OBJECT キーワードは、クラスのインスタンス・データおよびインスタンス・メソッドの定義の開始を示します。例えば、Account クラスのインスタンス・データの定義は、以下のようになります。

```
IDENTIFICATION DIVISION.
Object.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 AccountNumber pic 9(6).
 01 AccountBalance pic S9(9) value zero.
 . . .
End Object.
```

インスタンス・データは、オブジェクト・インスタンスが作成されるときに割り振られ、Java ランタイムによるインスタンスのガーベッジ・コレクションが行われるまで存在します。

上記に示すように、単純インスタンス・データの初期化は、VALUE 節を使用して行うことができます。より複雑なインスタンス・データの初期化は、カスタマイズしたメソッドをコーディングし、クラスのインスタンスを作成して初期化して行うことができます。

COBOL インスタンス・データは、Java private 非静的メンバー・データと同等です。他のクラスまたはサブクラスは (同じクラス内にあるファクトリー・メソッドも同様)、COBOL インスタンス・データを直接参照することはできません。インスタンス・データは、OBJECT 段落で定義するすべてのインスタンス・メソッドにグローバルです。OBJECT 段落の外側からインスタンス・データにアクセス可能にしたい場合には、アクセスを可能にするために属性 (get または set) インスタンス・メソッドを定義します。

インスタンス・データ定義のための WORKING-STORAGE SECTION の構文は、プログラムにおける場合と一般的に同じですが、次のような例外があります。

- EXTERNAL 属性を使用することはできません。
- GLOBAL 属性を使用できますが、効力はありません。

### 関連タスク

- 721 ページの『クラスのインスタンスの作成および初期化』
- 723 ページの『クラスのインスタンスの解放』
- 730 ページの『ファクトリー・メソッドの定義』
- 708 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

333 ページの『第 16 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』

711 ページの『クライアントの定義』

## クラス・インスタンス・メソッドの定義

クラス定義の OBJECT 段落の PROCEDURE DIVISION に COBOL インスタンス・メソッドを定義します。インスタンス・メソッドは、クラスのそれぞれのオブジェクト・インスタンスごとにサポートされる操作を定義します。

COBOL インスタンス・メソッド定義は、4 つの部 (COBOL プログラムに類似) とその後の END METHOD マーカーから構成されます。

表 76. インスタンス・メソッド定義の構成

除算	目的	構文
IDENTIFICATION (必須)	メソッドの名前を指定する。	<p>703 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)</p> <p>AUTHOR 段落 (オプション)</p> <p>INSTALLATION 段落 (オプション)</p> <p>DATE-WRITTEN 段落 (オプション)</p> <p>DATE-COMPILED 段落 (オプション)</p>
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。	<p>704 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)</p>

表 76. インスタンス・メソッド定義の構成 (続き)

除算	目的	構文
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。	704 ページの『クラス・インスタンス・メソッド定義用の DATA DIVISION』 (オプション)
PROCEDURE (オプション)	メソッドによって提供されるサービスを完了するために実行可能ステートメントをコーディングする。	705 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』 (オプション)

定義: メソッドのシグニチャーは、メソッドの名前およびその仮パラメーターの数と型から構成されています。(COBOL メソッドの仮パラメーターの定義は、そのメソッドの PROCEDURE DIVISION ヘッダーの USING 句で行います。)

クラス定義内では、各メソッド名を固有にする必要はありませんが、各メソッドに固有のシグニチャーを与える必要があります。(メソッドに同じ名前を与えても、別々のシグニチャーを与えると、メソッドを多重定義 することになります。)

COBOL インスタンス・メソッドは Java public 非静的メソッドと同じです。

709 ページの『例: メソッドの定義』

#### 関連タスク

705 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

707 ページの『インスタンス・メソッドの多重定義』

706 ページの『インスタンス・メソッドのオーバーライド』

716 ページの『メソッドの呼び出し (INVOKE)』

727 ページの『サブクラス・インスタンス・メソッドの定義』

730 ページの『ファクトリー・メソッドの定義』

## クラス・インスタンス・メソッド定義用の **METHOD-ID** 段落

インスタンス・メソッドを指定するには、METHOD-ID 段落を使用してください。IDENTIFICATION DIVISION 宣言とともに、METHOD-ID 段落の直前に置いて、メソッド定義の開始を表します。

例えば、Account クラス内の credit メソッドの定義は、以下のように開始します。

```
Identification Division.
Method-id. "credit".
```

メソッド名は英数字リテラル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

716 ページの『メソッドの呼び出し (INVOKE)』

147 ページの『COBOL での国別データ (Unicode) の使用』

関連参照

*The Java Language Specification* (Meaning of method names)

*The Java Language Specification* (Identifiers)

METHOD-ID 段落 (*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。

例えば、情報をファイルから読み取ったメソッドを Account クラスが定義した場合には、その Account クラスは、以下のようにコーディングされる INPUT-OUTPUT SECTION を持つと考えられます。

```
Environment Division.
Input-Output Section.
File-Control.
 Select account-file Assign AcctFile.
```

メソッドの INPUT-OUTPUT SECTION の構文は、プログラムの INPUT-OUTPUT SECTION の構文と同じです。

関連タスク

6 ページの『コンピューター環境の記述』

関連参照

INPUT-OUTPUT SECTION (*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**

インスタンス・メソッドの DATA DIVISION は、4 つのセクション (FILE SECTION、LOCAL-STORAGE SECTION、WORKING-STORAGE SECTION、および LINKAGE SECTION) の任意のもので構成されます。

### **FILE SECTION**

メソッド FILE SECTION では EXTERNAL ファイルしか定義できないことを除けば、プログラム FILE SECTION と同じです。

### **LOCAL-STORAGE SECTION**

メソッドの呼び出しごとに、LOCAL-STORAGE データの別個のコピーを割り振り、そのメソッドからの戻り時に解放します。メソッド LOCAL-STORAGE SECTION は、プログラム LOCAL-STORAGE SECTION に類似しています。

データ項目上の VALUE 節を指定すると、メソッドの呼び出しのたびに、項目はその値に初期化されます。

### **WORKING-STORAGE SECTION**

WORKING-STORAGE データの単一コピーが割り振られます。データは、実行

単位が終了するまで、最後に使用さ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とは無関係に、データの同じ単一コピーが使用されます。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』

584 ページの『**EXTERNAL** 節によるデータの共用』

#### 関連参照

**DATA DIVISION** 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。

プログラムの **PROCEDURE DIVISION** でコーディングできるメソッドの **PROCEDURE DIVISION** において、ほとんどの **COBOL** ステートメントをコーディングすることができます。ただし、メソッドで以下のステートメントをコーディングすることはできません。

- **ENTRY**
- **EXIT PROGRAM**
- 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。どちらのステ

ートメントも効果は同じです。メソッドが呼び出されるときに RETURNING 句が指定されていると、EXIT METHOD または GOBACK ステートメントは、呼び出し側のクライアントにデータの値を戻します。

各メソッドの PROCEDURE DIVISION では、暗黙の EXIT METHOD が最後のステートメントとして生成されます。

メソッドで STOP RUN を指定することができます。この指定を行うと、実行単位内で実行しているすべてのスレッドを含め、実行単位全体が終了します。

メソッド定義の終了は、END METHOD マーカーを使用して行う必要があります。例えば、次のステートメントは credit メソッドの終わりを示します。

```
End method "credit".
```

渡された引数の取得のための **USING** 句: メソッドの PROCEDURE DIVISION ヘッダーの USING 句に、メソッドの仮パラメーター (ある場合) を指定してください。引数が BY VALUE で渡される指定をしなければなりません。各パラメーターは、メソッドの LINKAGE SECTION で、レベル 01 またはレベル 77 項目として定義します。各パラメーターのデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

値を返すための **RETURNING** 句: メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句に、メソッドの結果として戻されるデータ項目 (ある場合) を指定してください。データ項目は、メソッドの LINKAGE SECTION で、レベル 01 またはレベル 77 項目として定義します。戻り値のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

#### 関連タスク

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

『インスタンス・メソッドのオーバーライド』

707 ページの『インスタンス・メソッドの多重定義』

715 ページの『オブジェクト参照の比較および設定』

716 ページの『メソッドの呼び出し (INVOKE)』

333 ページの『第 16 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

428 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。

スーパークラス・インスタンス・メソッド m1 を COBOL サブクラスでオーバーライドするには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでインスタンス・メソッド m1 を定義しま



す。(スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラスのインスタンスで `m1` を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

例えば、`Account` クラスは、その `LINKAGE SECTION` および `PROCEDURE DIVISION` ヘッダーが以下のような、メソッド `debit` を定義します。

```
Linkage section.
01 inDebit pic S9(9) binary.
Procedure Division using by value inDebit.
```

`CheckingAccount` サブクラスを定義し、`Account` スーパークラスで定義された `debit` メソッドをオーバーライドする `debit` メソッドをそれに持たす場合には、`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。

メソッド戻り値の有無および `PROCEDURE DIVISION RETURNING` 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・インスタンス・メソッドとオーバーライドしたスーパークラス・インスタンス・メソッドにおいて同一でなければなりません。

インスタンス・メソッドは、COBOL スーパークラスのファクトリー・メソッドをオーバーライドしてはならないし、Java スーパークラスの静的メソッドをオーバーライドすることもできません。

709 ページの『例: メソッドの定義』

#### 関連タスク

705 ページの『クラス・インスタンス・メソッド定義用の `PROCEDURE DIVISION`』

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

716 ページの『メソッドの呼び出し (INVOKE)』

720 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』

724 ページの『サブクラスの定義』

732 ページの『ファクトリー・メソッドまたは静的メソッドの隠蔽』

#### 関連参照

*The Java Language Specification* (Inheritance, overriding, and hiding)

## インスタンス・メソッドの多重定義

クラスでサポートされる 2 つのメソッド (クラスで定義されているか、スーパークラスから継承されたかにかかわらず) は、それらが同じ名前を持っており、シグニチャーが異なる場合には、多重定義されていると言います。

例えば、別々の一組のパラメーターを使用してデータを初期化するために、クライアントが別々の版のメソッドを呼び出せるようにするときは、メソッドを多重定義します。

メソッドを多重定義するには、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数や型が、同じクラスでサポートされる同一の名前のメソッドと異なるメソッドを定義します。例えば、Account クラスは、必ず 1 つの仮パラメーターを持つインスタンス・メソッド `init` を定義します。init メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下のようになります。

```
Linkage section.
01 inAccountNumber pic S9(9) binary.
Procedure Division using by value inAccountNumber.
```

クライアントはこのメソッドを呼び出し、`inAccountNumber` のデータ型と一致する引数を必ず 1 つ渡して、指定の口座番号 (およびデフォルトの勘定残高ゼロ) で Account インスタンスを初期化します。

ただし、Account クラスでは、例えば、開始の勘定残高も指定できる別の仮パラメーターを持つ、2 番目のインスタンス・メソッド `init` を定義することができます。この `init` メソッドの LINKAGE SECTION および PROCEDURE DIVISION ヘッダーは、以下のようになります。

```
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多重定義できます。

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

716 ページの『メソッドの呼び出し (INVOKE)』

730 ページの『ファクトリー・メソッドの定義』

#### 関連参照

*The Java Language Specification (Overloading)*

## 属性 (**get** および **set**) メソッドのコーディング

X のための accessor (`get`) メソッドおよび mutator (`set`) メソッドをコーディングして X を定義するクラスの外側から、インスタンス変数 X へのアクセスを提供することができます。

COBOL のインスタンス変数は、*private* です。インスタンス変数を定義するクラスは、インスタンス変数を完全にカプセル化します。したがって、直接アクセスでき

るのは、同じ OBJECT 段落で定義するインスタンス・メソッドだけです。通常、優れた設計のオブジェクト指向アプリケーションは、クラスの外側からインスタンス変数にアクセスする必要はありません。

public インスタンス変数の概念は、Java および他のオブジェクト指向言語で定義されており、クラス属性の概念は CORBA で定義されていますが、どちらの概念も COBOL には直接サポートされていません。(CORBA 属性 は、変数が読み取り専用でない場合に、変数の値にアクセスするための自動生成 get メソッドと変数の値を変更するための自動生成 set メソッドを持つ、インスタンス変数です。)

『例: ゲット・メソッドのコーディング』

関連タスク

701 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

19 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、701 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 ZZZZZ999999.
01 PrintableAccountBalance pic $$$,$$$,$$9CR.
Procedure Division.
 Move AccountNumber to PrintableAccountNumber
 Move AccountBalance to PrintableAccountBalance
 Display " Account: " PrintableAccountNumber
 Display " Balance: " PrintableAccountBalance.
End method "print".
*
End Object.
*
End class Account.

```

## Check クラス

```

/**
 * A Java class for check information
 */
public class Check {
 private CheckingAccount payer;
 private Account payee;
 private int amount;

 public Check(CheckingAccount inPayer, Account inPayee, int inAmount) {
 payer=inPayer;
 payee=inPayee;
 amount=inAmount;
 }

 public int getAmount() {
 return amount;
 }

 public Account getPayee() {
 return payee;
 }
}

```

### 関連タスク

333 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

---

## クライアントの定義

クラス内で 1 つ以上のメソッドからサービスを要求するプログラムまたはメソッドは、そのクラスの **クライアント** と呼ばれます。

COBOL クライアントまたは Java クライアントで、以下のことを行うことができます。

- Java および COBOL クラスのオブジェクト・インスタンスを作成する。
- Java および COBOL オブジェクトにインスタンス・メソッドを呼び出す。
- COBOL ファクトリー・メソッドおよび Java 静的メソッドを呼び出す。

COBOL クライアントでは、Java Native Interface (JNI) が提供するサービスを呼び出すこともできます。

COBOL クライアント・プログラムは、次のような通常の 4 つの部分で成り立っています。

表 77. COBOL クライアントの構成

除算	目的	構文
IDENTIFICATION (必須)	クライアントの名前。	通常のようにコーディング。ただし、クライアント・プログラムは以下のとおりにする必要があります。 <ul style="list-style-type: none"> <li>再帰的 (PROGRAM-ID 段落内の RECURSIVE 宣言)</li> <li>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)</li> </ul>
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) 713 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。	714 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

THREAD コンパイラー・オプションを使用して、オブジェクト指向構文を含む、または Java と相互協調処理する、すべての COBOL プログラムをコンパイルする必要があるため、以下の言語エレメントは COBOL クライアントで使用できません。

- 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。

723 ページの『例: クライアントの定義』

#### 関連タスク

333 ページの『第 16 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』

603 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

743 ページの『第 34 章 Java メソッドとの通信』

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

721 ページの『クラスのインスタンスの作成および初期化』

715 ページの『オブジェクト参照の比較および設定』

716 ページの『メソッドの呼び出し (INVOKE)』

732 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』

## クライアント定義用の **REPOSITORY** 段落

指定された語を COBOL クライアント内で使用するときその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

外部クラス名は大/小文字が区別されます。したがって、Java 形成規則に準拠しなければなりません。例えば、Account および Check クラスを使用するクライアント・プログラムにおいて、以下のようにコーディングすることがあります。

```
Environment division. 必須
Configuration section. 必須
Source-Computer. IBM-390.
Object-Computer. IBM-390.
Repository. 必須
 Class Account is "Account"
 Class Check is "Check".
```

REPOSITORY 段落記入項目では、クライアント内で Account および Check として参照されるクラスの外部クラス名は、それぞれ、Account および Check であることが示されます。

REPOSITORY 段落では、クライアントにおいて明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。REPOSITORY 段落記入項目において、名前に非 COBOL 文字が含まれている場合には、外部クラス名を指定しなければなりません。

Java パッケージの一部である参照クラスごとに外部クラス名を指定しなければなりません。そのようなクラスごとに、外部クラス名をパッケージの完全修飾名として指定し、後にピリオド (.) が付き、続いてその後に Java クラスの単純名が付きます。

REPOSITORY 段落で指定する外部クラス名は、完全修飾 Java クラス名の形成規則に準拠した英数字リテラルでなければなりません。

REPOSITORY 段落記入項目に外部クラス名を組み込まない場合、外部クラス名の作成は、クラス定義で外部クラス名が REPOSITORY 段落記入項目に組み込まれていない場合と同じ方法で、クラス名から行われます。外部名は英大/小文字混合で名前付けされます。したがって、上記の例で、クラス Account およびクラス Check は、それぞれ、外部的には Account および Check (英大/小文字混合) として認識されます。

CONFIGURATION SECTION の SOURCE-COMPUTER、OBJECT-COMPUTER、および SPECIAL-NAMES 段落はオプションです。

関連タスク

699 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)

## クライアント定義用の DATA DIVISION

クライアントが必要とするデータを記述するには、DATA DIVISION の任意のセクションを使用できます。

```
Data Division.
Local-storage section.
01 anAccount usage object reference Account.
01 aCheckingAccount usage object reference CheckingAccount.
01 aCheck usage object reference Check.
01 payee usage object reference Account.
. . .
```

クライアントはクラスを参照するので、オブジェクト参照 (つまり、クラスのインスタンスへの参照) と呼ばれる 1 つ以上の特別なデータ項目を必要とします。インスタンス・メソッドに対するすべての要求で、メソッドがサポートされる (つまり、継承によって定義されているか使用可能である) クラスのインスタンスに対するオブジェクト参照が必要です。COBOL クラスのインスタンスを参照する場合と同じ構文を使用して、オブジェクト参照をコーディングして Java クラスのインスタンスを参照します。上記の例では、usage object reference という句は、オブジェクト参照データ項目を表します。

上記コードの 4 つのオブジェクト参照はすべて、クラス名が OBJECT REFERENCE 句の後に現れるので、型式化オブジェクト参照と呼ばれます。型式化オブジェクト参照の参照先は、OBJECT REFERENCE 句で名前付けしたクラスのインスタンス、またはそのサブクラスのいずれか 1 つに限られます。したがって、anAccount は、Account クラスのインスタンス、またはそのサブクラスのいずれかを参照できますが、ほかのクラスのインスタンスを参照することはできません。同様に、aCheck の参照先は、Check クラスのインスタンス、またはそのサブクラスのインスタンスに限られます。

別の型のオブジェクト参照 (上に示されていない) には、OBJECT REFERENCE 句の後にクラス名がありません。そのような参照を汎用 オブジェクト参照と呼びます。すべてのクラスのインスタンスを参照できるという意味です。汎用オブジェクト参照はコーディングしないようにしてください。汎用オブジェクト参照は、非常に限られた環境で (INVOKE *class-name* NEW . . . ステートメントの RETURNING 句で使用される場合に) のみ Java と相互運用可能であるためです。

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

- 715 ページの『LOCAL-STORAGE または WORKING-STORAGE の選択』
- 749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
- 716 ページの『メソッドの呼び出し (INVOKE)』
- 713 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。

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

603 ページの『第 27 章 マルチスレッド化のための COBOL プログラムの準備』

## オブジェクト参照の比較および設定

条件ステートメントまたは JNI サービス IsSameObject の呼び出しをコーディングすることによってオブジェクト参照を比較できます。また、SET ステートメントを使用して、オブジェクト参照を設定できます。

例えば、次の IF ステートメントのいずれかをコーディングして、オブジェクト参照 anAccount がオブジェクト・インスタンスをまったく参照していないことをチェックします。

```
If anAccount = Null . . .
If anAccount = Nulls . . .
```

IsSameObject の呼び出しをコーディングして、2 つのオブジェクト参照 (object1 と object2) が同じオブジェクト・インスタンスを指すかどうか、あるいはそれぞれがオブジェクト・インスタンスを指さないかどうかを検査することができます。引数および戻り値が Java と相互運用可能であることを確認し、呼び出し可能サービスへのアドレス可能性を確立するには、IsSameObject への呼び出しの前に、以下のデータ定義およびステートメントをコーディングします。

```
Local-storage Section.
. . .
01 is-same Pic X.
 88 is-same-false Value X'00'.
 88 is-same-true Value X'01' Through X'FF'.
Linkage Section.
 Copy JNI.
Procedure Division.
 Set Address Of JNIEnv To JNIEnvPtr
 Set Address Of JNINativeInterface To JNIEnv
 Call IsSameObject Using By Value JNIEnvPtr object1 object2
 Returning is-same
 If is-same-true . . .
```

メソッド内では、オブジェクト参照と SELF を比較する IsSameObject の呼び出しをコーディングすることにより、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト参照が指すかどうかを検査できます。

上記の代わりに、Java equals メソッド (java.lang.Object からの継承) を呼び出して、2 つのオブジェクト参照が同一のオブジェクト・インスタンスを参照するかを決定することができます。

SET ステートメントを使用することにより、オブジェクト参照がオブジェクト・インスタンスを指さないようにすることができます。以下に例を示します。

```
Set anAccount To Null.
```

また、SET ステートメントを使用して、1 つのオブジェクト参照が別のオブジェクト参照と同じインスタンスを参照するように設定することもできます。以下に例を示します。

```
Set anotherAccount To anAccount.
```

この SET ステートメントによって、anotherAccount は anAccount と同じオブジェクト・インスタンスを参照します。受け取り側 (anotherAccount) が汎用オブジェクト参照である場合、送り出し側 (anAccount) は、汎用オブジェクト参照または型式化オブジェクト参照のどちらかになります。受け取り側が型式化オブジェクト参照である場合は、送り出し側も、受け取り側と同じクラス、または、そのサブクラスのいずれか 1 つにバインドされた、型式化オブジェクト参照でなければなりません。

メソッド内では、オブジェクト参照を SELF に設定して、メソッドが呼び出されたオブジェクト・インスタンスをオブジェクト参照が参照するように設定することができます。以下に例を示します。

```
Set anAccount To Self.
```

#### 関連タスク

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

743 ページの『JNI サービスへのアクセス』

#### 関連参照

*The Java Native Interface (IsSameObject)*

## メソッドの呼び出し (INVOKE)

Java クライアントでは、COBOL でインプリメントされたクラスのオブジェクト・インスタンスを作成し、標準 Java 構文を使用して、それらのオブジェクトでメソッドを呼び出すことができます。COBOL クライアントでは、INVOKE ステートメントをコーディングすることにより、Java または COBOL クラスで定義されたメソッドを呼び出すことができます。

```
Invoke Account "createAccount"
 using by value 123456
 returning anAccount
Invoke anAccount "credit" using by value 500.
```

上の最初の例の INVOKE ステートメントは、クラス名 Account を使用して、createAccount という名前のメソッドを呼び出します。このメソッドは、Account クラスで定義または継承されている必要があります。また、次のいずれかの型である必要があります。

- Java 静的メソッド
- COBOL ファクトリー・メソッド

using by value 123456 という句は、123456 が、メソッドの入力引数であり、値により受け渡されることを表します。入力引数 123456 と戻されるデータ項目

anAccount は、(多重定義されているかもしれない) createAccount メソッドの仮パラメーターおよび戻りの型の定義にそれぞれ準拠している必要があります。

2 番目の INVOKE ステートメントは、戻されるオブジェクト参照 anAccount を使用して、インスタンス・メソッド credit (Account クラスに定義されている) を呼び出します。 入力引数 500 は、(おそらく多重定義された) credit メソッドの仮パラメーターの定義に準拠しなければなりません。

呼び出すメソッドの名前を、実行時の値がターゲット・メソッドのシグニチャーにあるメソッド名に一致する、リテラルまたは ID として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ない、解釈されるときには大/小文字が区別されます。

INVOKE ステートメントを (上記の 2 番目の例のステートメントのように) オブジェクト参照を使用してコーディングする場合、そのステートメントは次の 2 つの形式のうちのいずれかで始まります。

```
Invoke objRef "literal-name" . . .
Invoke objRef identifier-name . . .
```

メソッド名が ID である場合には、オブジェクト参照 (objRef) を、指定する型のない USAGE OBJECT REFERENCE として、すなわち、汎用オブジェクト参照として、定義しなければなりません。

呼び出されたメソッドが、オブジェクト参照の参照先のクラスでサポートされない場合、重大度 3 の言語環境プログラム条件が発生時に発生します。ただし、INVOKE ステートメントで ON EXCEPTION 句をコーディングした場合は別です。

オプションの範囲終了符号 END-INVOKE を INVOKE ステートメントで 사용할 ことができます。

INVOKE ステートメントは RETURN-CODE 特殊レジスターを設定しません。

#### 関連タスク

『引数の引き渡し用の USING 句』

720 ページの『戻り値の取得用の RETURNING 句』

705 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

720 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』

732 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。

表 78. 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 <b>DATA DIVISION</b> 定義を次のようにコーディングします	制約事項
COBOL	いいえ	対応する仮パラメーターの定義と同じ	
Java	いいえ	対応する Java パラメーターと相互運用可能	
COBOL または Java	はい	ターゲット・メソッドの対応するパラメーターと同じクラスに型式化されるオブジェクト参照	COBOL クライアントでは (Java クライアントとは異なり)、引数のクラスを、対応するパラメーターのクラスのサブクラスにすることができません。

SET ステートメントまたは REDEFINES 節を使用して、オブジェクト参照引数を対応する仮パラメーターの型と一致させる方法については、以下に参照されている例を参照してください。

『例: COBOL クライアントからの規格合致オブジェクト参照の引数の引き渡し』

ターゲット・メソッドが多重定義されている場合、引数のデータ型は、同じ名前を持つメソッドの中から選択するために使用されます。

引数が BY VALUE で渡される指定をしなければなりません。言い換えれば、引数は、呼び出されるメソッドの対応する仮パラメーターが変更されても影響を受けません。

各引数のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。

#### 関連タスク

- 705 ページの『クラス・インスタンス・メソッド定義用の PROCEDURE DIVISION』
- 707 ページの『インスタンス・メソッドの多重定義』
- 749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
- 573 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) を割り当てた後、anObj を引数として M に渡すことができます。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。

756 ページの『例: COBOL で書かれた J2EE クライアント』

関連タスク

- 749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』
- 705 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。

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 79. **COBOL** クライアントでの戻されるデータ項目の合致

ターゲット・メソッド のプログラミング言語	戻される項目はオブジ ェクト参照ですか	戻される項目の <b>DATA DIVISION</b> 定義を次の ようにコーディングします
COBOL	いいえ	ターゲット・メソッドの RETURNING 項目の 定義と同じ
Java	いいえ	戻された Java データ項目と相互運用可能
COBOL または Java	はい	ターゲット・メソッドが戻すオブジェクト参 照と同じクラスに型式化されるオブジェクト 参照

すべての場合において、戻り値のデータ型は、Java と相互運用可能になる型のうちのいずれかでなければなりません。

関連タスク

- 749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

- INVOKE ステートメント (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。

例えば、CheckingAccount クラスが、その即時スーパークラス Account に定義されている debit インスタンス・メソッドをオーバーライドすると想定します。次のステートメントをコーディングして、CheckingAccount クラスのメソッド内で Account の debit メソッドを呼び出すことができます。

Invoke Super "debit" Using By Value amount.

debit メソッドのシグニチャーに一致するように、amount を PIC S9(9) BINARY として定義します。

CheckingAccount クラスは、Account クラスに定義されている print メソッドをオーバーライドします。print メソッドには仮パラメーターがありません。したがって、CheckingAccount クラス内のメソッドは、次のステートメントでスーパークラス print メソッドを呼び出すことができます。

Invoke Super "print".

キーワード SUPER は、現行クラス内のメソッドではなく、スーパークラス・メソッドを呼び出すことを指示します。(SUPER は、現在実行中のメソッドの起動に使用されているオブジェクトへの暗黙の参照です。)

694 ページの『例: 口座』

関連タスク

706 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。

クラス *class-name* の新しいインスタンスを生成するには、また、作成したオブジェクトへの参照 *object-reference* を取得するには、次の形式のステートメントをコーディングします (*object-reference* は、クライアントの DATA DIVISION で定義されます)。

INVOKE *class-name* NEW . . . RETURNING *object-reference*

メソッド内で INVOKE . . . NEW ステートメントをコーディングし、戻されたオブジェクト参照の使用がメソッド起動の期間に限定されていない場合は、JNI サービス NewGlobalRef を呼び出すことによって、戻された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。

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

722 ページの『Java クラスのインスタンス化』

722 ページの『COBOL クラスのインスタンス化』

743 ページの『JNI サービスへのアクセス』

746 ページの『ローカル参照とグローバル参照の管理』

714 ページの『クライアント定義用の DATA DIVISION』

716 ページの『メソッドの呼び出し (INVOKE)』

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

関連参照

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

## Java クラスのインスタンス化

Java クラスをインスタンス化するには、対象のクラスでサポートされる任意のパラメーター化コンストラクターを呼び出します。そのためには、`INVOKE . . . NEW` ステートメントで `USING` 句を `RETURNING` 句の直前にコーディングし、そのコンストラクターのシグニチャーに適合する引数の数とタイプを `BY VALUE` に渡します。

各引数のデータ型は、Java と相互運用可能な型のいずれかでなければなりません。デフォルトの (パラメーターなし) コンストラクターを呼び出すには、`USING` 句を省略します。

例えば、`Check` クラスのインスタンスを生成し、そのインスタンス・データを初期化し、生成した `Check` インスタンスへの参照 `aCheck` を取得するには、次のステートメントを COBOL クライアントにコーディングすることができます。

```
Invoke Check New
 using by value aCheckingAccount, payee, 125
 returning aCheck
```

### 関連タスク

716 ページの『メソッドの呼び出し (INVOKE)』

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

### 関連参照

VALUE 節 (*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。

したがって、`INVOKE . . . NEW` ステートメントは、単一のインスタンス・データのみを有する COBOL クラスのインスタンスを生成するときに役立ちます。例えば、次のステートメントは、`Account` クラスのインスタンスを作成し、`Account` クラス定義の `OBJECT` 段落の `WORKING-STORAGE SECTION` の `VALUE` 節に指定されたとおりに、インスタンス・データを初期化し、新しいインスタンスへの参照 `outAccount` を提供します。

```
Invoke Account New returning outAccount
```

`VALUE` 節だけを使用して初期化することができない COBOL クラス・データの初期化を可能にするには、COBOL クラスを設計する際に、`FACTORY` 段落にパラメーター化生成メソッドを定義し、`OBJECT` 段落にパラメーター化初期化メソッドを定義する必要があります。

1. パラメーター化ファクトリー生成メソッドで、以下の手順を実行します。
  - a. `INVOKE class-name NEW RETURNING objectRef` をコーディングして、`class-name` のインスタンスを作成し、`VALUE` 節を有するインスタンス・データ項目に初期値を与えます。
  - b. パラメーター化した初期化メソッドをインスタンス (`objectRef`) 上で呼び出し、指定された引数 `BY VALUE` をファクトリー・メソッドに渡します。



2. 初期化メソッドで、ロジックをコーディングし、仮パラメーターを介して指定された値を使用して、インスタンス・データ初期化を完了します。

COBOL クラスのインスタンスを作成して適切に初期化するために、クライアントはパラメーター化ファクトリー・メソッドを呼び出し、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。

733 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

- 743 ページの『JNI サービスへのアクセス』
- 746 ページの『ローカル参照とグローバル参照の管理』
- 716 ページの『メソッドの呼び出し (INVOKE)』
- 728 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。これを行うために使用できる構文はありません。Java ランタイム・システムは自動的にガーベッジ・コレクションを実行します。すなわち、使用されなくなったオブジェクトのメモリーを再利用します。

ただし、参照済みオブジェクトのガーベッジ・コレクションを許可するために、ネイティブ COBOL クライアント内のオブジェクトへのローカル参照またはグローバル参照を明示的に解放する必要がある場合があります。

#### 関連タスク

- 746 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、709 ページの『例: メソッドの定義』に示されています。)

```

cbl 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".

```

733 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

730 ページの『ファクトリー・メソッドの定義』

732 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』

333 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

---

## サブクラスの定義

クラス (サブクラス、派生クラス、または子クラスと呼ばれる) を別のクラス (スーパークラス、基本クラス、または親クラスと呼ばれる) の特殊化クラスにすることができます。

サブクラスは、そのスーパークラスのメソッドおよびインスタンス・データを継承し、*is-a* 関係によって、そのスーパークラスに関連付けられています。例えば、サブクラス P がスーパークラス Q から継承し、サブクラス Q がスーパークラス S から継承した場合、P のインスタンスは Q のインスタンスであり、また (推移性によって) S のインスタンスでもあります。したがって、P のインスタンスは、Q と S のメソッドおよびデータを継承します。

サブクラスを使用することの利点:

- コードの再利用: 継承を通じて、サブクラスは、スーパークラスに既に存在するメソッドを再利用することができます。
- 特化: サブクラスでは、スーパークラスが処理しないケースを処理するために新規のメソッドを追加することができます。また、スーパークラスが必要としない新規のデータ項目を追加することもできます。

- アクションの変更: サブクラスは、スーパークラスにあるシグニチャーと同じシグニチャーのメソッドを定義して、スーパークラスから継承するメソッドをオーバーライドすることができます。メソッドをオーバーライドするときは、メソッドの実行内容について、いくつかの小さい変更を行うだけの場合、または全面的な変更を行う場合があります。

制約事項: COBOL プログラムでは多重継承を使用することはできません。定義する各 COBOL クラスには、Java または COBOL でインプリメントされた即時スーパークラスは必ず 1 つだけでなければなりません。また、それぞれのクラスは、直接的または間接的に `java.lang.Object` から派生したものでなければなりません。継承のセマンティクスは、Java によって定義されます。

サブクラスの構造および構文は、クラス定義の構造および構文と同一です。サブクラス定義の OBJECT 段落内で、それぞれ、DATA DIVISION および PROCEDURE DIVISION に、インスタンス・データとメソッドを定義します。個別のオブジェクト・インスタンスではなく、サブクラス自体に関連付けるデータとメソッドを必要とするサブクラスに、サブクラス定義の FACTORY 段落内で、別々の DATA DIVISION および PROCEDURE DIVISION を定義します。

COBOL インスタンス・データは `private` です。サブクラスが COBOL スーパークラスのインスタンス・データにアクセスすることができるのは、スーパークラスがそのアクセスを可能にするために属性 (`get` または `set`) インスタンス・メソッドを定義する場合に限られます。

694 ページの『例: 口座』

727 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

697 ページの『クラスの定義』

706 ページの『インスタンス・メソッドのオーバーライド』

708 ページの『属性 (`get` および `set`) メソッドのコーディング』

727 ページの『サブクラス・インスタンス・メソッドの定義』

728 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

サブクラスを指定し、それが特性を継承する直接の Java または COBOL スーパークラスを示すには、CLASS-ID 段落を使用してください。

Identification Division. 必須  
Class-id. CheckingAccount inherits Account. 必須

上の例で、CheckingAccount は定義するサブクラスです。CheckingAccount は、サブクラス定義において Account として認識されているクラスのすべてのメソッドを継承します。CheckingAccount メソッドが Account インスタンス・データにアクセスできるのは、Account クラスが、そのアクセスを可能にするために、属性 (`get` または `set`) メソッドを指定する場合に限られます。

ENVIRONMENT DIVISION の CONFIGURATION SECTION の REPOSITORY 段落に、即時スーパークラスの名前を指定しなければなりません。 オプションで、スーパークラス名を、外部で認識されているクラスの名前に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

698 ページの『クラス定義用の CLASS-ID 段落』

708 ページの『属性 (get および set) メソッドのコーディング』

『サブクラス定義用の REPOSITORY 段落』

## サブクラス定義用の REPOSITORY 段落

指定された語をサブクラス定義内で使用するときその語がクラス名であることをコンパイラーに宣言する場合、さらに必要に応じてクラス名を対応する外部クラス名 (コンパイル単位の外側で認識されているクラス名) に関係付ける場合に、REPOSITORY 段落を使用してください。

例えば、CheckingAccount サブクラス定義では、これらの REPOSITORY 段落記入項目は、サブクラス定義内で CheckingAccount、Check、および Account として参照されるクラスの外部クラス名が、それぞれ、CheckingAccount、Check、および Account であることを示します。

Environment Division.		必須
Configuration Section.		必須
Repository.		必須
Class CheckingAccount	is "CheckingAccount"	オプション
Class Check	is "Check"	必須
Class Account	is "Account".	必須

REPOSITORY 段落では、サブクラス定義において明示的に参照するそれぞれのクラス名ごとに記入項目をコーディングする必要があります。 以下に例を示します。

- 定義中のサブクラスが継承する元のユーザー定義スーパークラス
- サブクラス定義内のメソッドで参照するクラス

サブクラス内の REPOSITORY 段落記入項目をコーディングする場合の規則は、クラス内の REPOSITORY 段落記入項目をコーディングする場合の規則と同一です。

関連タスク

699 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

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

701 ページの『クラス・インスタンス・データ定義用の WORKING-STORAGE SECTION』

## サブクラス・インスタンス・メソッドの定義

サブクラスは、そのスーパークラスのメソッドを継承します。サブクラス定義において、継承したメソッドと同じシグニチャーのインスタンス・メソッドを定義して、サブクラスが継承するインスタンス・メソッドをオーバーライドすることができます。また、サブクラスが必要とする新しいメソッドを定義することもできます。

サブクラス・インスタンス・メソッドの構造と構文は、クラス・インスタンス・メソッドの構造と構文と同一です。サブクラス定義の OBJECT 段落の PROCEDURE DIVISION にサブクラス・インスタンス・メソッドを定義します。

『例: サブクラスの定義 (メソッドに関して)』

関連タスク

702 ページの『クラス・インスタンス・メソッドの定義』

706 ページの『インスタンス・メソッドのオーバーライド』

707 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、709 ページの『例: メソッドの定義』に示されています。)

## CheckingAccount クラス (Account のサブクラス)

```
cb1 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

333 ページの『第 16 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』

716 ページの『メソッドの呼び出し (INVOKE)』

706 ページの『インスタンス・メソッドのオーバーライド』

720 ページの『オーバーライドされたスーパークラス・メソッドの呼び出し』

---

## ファクトリー・セクションの定義

個別のオブジェクト・インスタンスではなく、クラス自身に関連付けるデータおよびメソッドを定義するためには、クラス定義の FACTORY 段落を使用します。

COBOL ファクトリー・データ は、Java private 静的データと同じです。データの単一コピーは、そのクラス用にインスタンス生成され、クラスのすべてのオブジェクト・インスタンスに共用されます。クラスのすべてのインスタンスからデータを収集したい場合は一般的に、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 80. ファクトリー定義の構成

除算	目的	構文
IDENTIFICATION (必須)	ファクトリー定義の開始を示す。	IDENTIFICATION DIVISION. FACTORY.
DATA (オプション)	このクラス用に一度割り振られたデータを記述する (クラスのそれぞれのインスタンスごとに割り振られたデータとは正反対)。	『ファクトリー・データ定義用の WORKING-STORAGE SECTION』 (オプション)
PROCEDURE (オプション)	ファクトリー・メソッドを定義する。	ファクトリー・メソッドの定義: 730 ページの『ファクトリー・メソッドの定義』

733 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

- 697 ページの『クラスの定義』
- 722 ページの『COBOL クラスのインスタンス化』
- 738 ページの『プロシージャ指向 COBOL プログラムのラッピング』
- 739 ページの『OO アプリケーションの構造化』

ファクトリー・データ定義用の **WORKING-STORAGE SECTION**

COBOL クラスが必要とする ファクトリー・データ、つまりクラスのすべてのオブジェクト・インスタンスに共有される、静的に割り当てられたデータを記述するには、FACTORY 段落の DATA DIVISION にある WORKING-STORAGE SECTION を使用します。

IDENTIFICATION DIVISION 宣言の直前に入れる必要がある FACTORY キーワードは、クラスのファクトリー・データおよびファクトリー・メソッドの定義の開始を示します。例えば、Account クラスのファクトリー・データの定義は、以下のようになります。

```
IDENTIFICATION DIVISION.
Factory.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 NumberOfAccounts pic 9(6) value zero.
.
.
End Factory.
```

上記に示すように、単純ファクトリー・データの初期化は、VALUE 節を使用して行うことができます。

COBOL ファクトリー・データは、Java private 静的データと同じです。他のクラスまたはサブクラスは (同じクラス内にあるインスタンス・メソッドも同様)、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・メソッドに対してグローバルです。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

708 ページの『属性 (get および set) メソッドのコーディング』

722 ページの『COBOL クラスのインスタンス化』

## ファクトリー・メソッドの定義

クラス定義の FACTORY 段落の PROCEDURE DIVISION に COBOL ファクトリー・メソッド を定義します。ファクトリー・メソッドは、クラスのどのオブジェクト・インスタンスとも無関係に、クラスでサポートされる操作を定義します。COBOL ファクトリー・メソッドは Java public 静的メソッドと同じです。

一般的には、そのインスタンスが複雑な初期化を必要とするクラスについて、すなわち、VALUE 節だけの使用では割り当てることができない値に対して、ファクトリー・メソッドを定義します。ファクトリー・メソッド内で、インスタンス・メソッドを呼び出してインスタンス・データを初期化できます。ファクトリー・メソッドは、インスタンス・データに直接アクセスすることはできません。

ファクトリー属性 (get および set) メソッドをコーディングして、FACTORY 段落の外側からファクトリー・データにアクセス可能にすることができます。例えば、同じクラスのインスタンス・メソッドから、またはクライアント・プログラムからファクトリー・データにアクセス可能にすることができます。例えば、Account クラスはファクトリー・メソッド getNumberOfAccounts を定義して、口座数の現在の集計を戻すことができます。

ファクトリー・メソッドを使用して、Java プログラムからアクセス可能になるようにプロシージャ指向の COBOL プログラムをラップすることもできます。main という名前のファクトリー・メソッドをコーディングすることで、java コマンドを



使用してオブジェクト指向アプリケーションを実行したり、Java の標準的な方法に従ってアプリケーションを構成したりすることができます。詳細については、関連タスクを参照してください。

ファクトリー・メソッドの定義では、インスタンス・メソッドの定義に使用するもの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 81. ファクトリー・メソッド定義の構成

除算	目的	構文
IDENTIFICATION (必須)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ (必須)
ENVIRONMENT (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
DATA (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ
PROCEDURE (オプション)	クラス・インスタンス・メソッドの場合と同じ	クラス・インスタンス・メソッドの場合と同じ

クラス定義内では、各ファクトリー・メソッド名を固有にする必要はありませんが、各ファクトリー・メソッドに固有のシグニチャーを与える必要があります。ファクトリー・メソッドは、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多重定義できます。例えば、CheckingAccount サブクラスは、ファクトリー・メソッド createCheckingAccount の 2 つの版 (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。

733 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

- 739 ページの『OO アプリケーションの構造化』
- 738 ページの『プロシーチャー指向 COBOL プログラムのラッピング』
- 722 ページの『COBOL クラスのインスタンス化』
- 702 ページの『クラス・インスタンス・メソッドの定義』
- 708 ページの『属性 (get および set) メソッドのコーディング』
- 707 ページの『インスタンス・メソッドの多重定義』
- 732 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』
- 732 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』
- 535 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。

スーパークラス・ファクトリー・メソッド f1 を COBOL サブクラスで隠すには、スーパークラス・メソッドと名前が同じで、その PROCEDURE DIVISION USING 句 (ある場合) の仮パラメーターの数およびタイプがスーパークラス・メソッドと同じであるサブクラスでファクトリー・メソッド f1 を定義します。 (スーパークラス・メソッドが Java でインプリメントされる場合には、対応する Java パラメーターのデータ型と相互運用可能な仮パラメーターをコーディングする必要があります。) クライアントがサブクラス名を使用して f1 を呼び出すとき、スーパークラス・メソッドではなく、サブクラス・メソッドが呼び出されます。

メソッド戻り値の有無および PROCEDURE DIVISION RETURNING 句 (ある場合) で使われる戻り値のデータ型は、サブクラス・ファクトリー・メソッドと隠されたスーパークラス・メソッドにおいて同一でなければなりません。

ファクトリー・メソッドは、Java または COBOL スーパークラスに、インスタンス・メソッドを隠してはいけません。

733 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

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

706 ページの『インスタンス・メソッドのオーバーライド』

716 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・メソッドまたは Java 静的メソッドを COBOL メソッドまたはクライアント・プログラムで呼び出すには、クラス名を INVOKE ステートメントの第 1 オペランドとしてコーディングしてください。

例えば、クライアント・プログラムは次のステートメントをコーディングして、createCheckingAccount という名前の多重定義 CheckingAccount ファクトリー・メソッドの 1 つを呼び出して、口座番号 777777 および開始残高 \$300 の当座預金を作成することができます。

```
Invoke CheckingAccount "createCheckingAccount"
 using by value 777777 300
 returning aCheckingAccount
```

ファクトリー・メソッドを定義する同じクラス内からファクトリー・メソッドを呼び出す場合にも、クラス名を INVOKE ステートメントの第 1 オペランドとして使用します。

実行時の値がメソッド名であるリテラルまたは 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ない、解釈されるときには大/小文字が区別されます。

呼び出されたメソッドが、`INVOKE` ステートメントで指定されたクラスでサポートされない場合、重大度 3 の言語環境プログラム条件が実行時に発生します。ただし、`INVOKE` ステートメントで `ON EXCEPTION` 句をコーディングした場合は別です。

`USING` 句で `COBOL` ファクトリー・メソッドまたは `Java` 静的メソッドに引数を渡すときの適合要件と、`RETURNING` 句で戻り値を受けるときの適合要件は、インスタンス・メソッドを呼び出す場合と同じです。

『例: ファクトリーの定義 (メソッドに関して)』

関連タスク

716 ページの『メソッドの呼び出し (`INVOKE`)』

147 ページの『`COBOL` での国別データ (`Unicode`) の使用』

749 ページの『`COBOL` および `Java` での相互運用可能なデータ型のコーディング』

関連参照

`INVOKE` ステートメント (*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、709 ページの『例: メソッドの定義』、723 ページの『例: クライアントの定義』、および 727 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 ZZZZZ999999.
01 PrintableAccountBalance pic $$$,$$$,$$9CR.
Procedure Division.
 Move AccountNumber to PrintableAccountNumber
 Move AccountBalance to PrintableAccountBalance
 Display " Account: " PrintableAccountNumber
 Display " Balance: " PrintableAccountBalance.
End method "print".
*
End Object.
*
End class Account.

```

## CheckingAccount クラス (Account のサブクラス)

```

cbl 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 730 ページの『ファクトリー・メソッドの定義』

732 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』

333 ページの『第 16 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。

COBOL コードをラップするためには、以下の手順を実行します。

1. FACTORY 段落を含む単純 COBOL クラスを作成する。



2. FACTORY 段落で、CALL ステートメントを使用してプロシージャ型プログラムを呼び出すファクトリー・メソッドをコーディングする。

Java プログラムは、静的メソッドの呼び出しの式を使用して、すなわち、COBOL プロシージャ型プログラムを呼び出して、ファクトリー・メソッドを呼び出すことができます。

#### 関連タスク

697 ページの『クラスの定義』

728 ページの『ファクトリー・セクションの定義』

730 ページの『ファクトリー・メソッドの定義』

---

## 00 アプリケーションの構造化

次の 3 つの方法のいずれかで、オブジェクト指向 COBOL 構文を使用するアプリケーションを構造化することができます。

オブジェクト指向アプリケーションは、次のいずれかで始めることができます。

- COBOL プログラム。任意の名前を付けることができます。

z/OS UNIX 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、単一の USING パラメーター (java.lang.String 型エレメントを持つ配列であるクラスに対するオブジェクト参照) を指定して宣言します。つまり、main は、事実上、String[] 型のパラメーターを 1 つ持つ、public、static、および void です。

main を含むクラスの名前を指定し、0 以上のストリングをコマンド行引数として渡すことで、java コマンドでアプリケーションを実行できます。

以下の場合には、この方法でオブジェクト指向アプリケーションを構成します。

- java コマンドを使用してアプリケーションを実行する。
- アプリケーションが Java クラスの main メソッドで開始しなければならない環境 (Java 従属領域など) でアプリケーションを実行する。
- 標準的な Java プログラミング方式に従う。

740 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

333 ページの『第 16 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』

730 ページの『ファクトリー・メソッドの定義』

749 ページの『Java 用の配列およびストリングの宣言』

529 ページの『第 22 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 型の要素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

333 ページの『第 16 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』

730 ページの『ファクトリー・メソッドの定義』

743 ページの『第 34 章 Java メソッドとの通信』



---

## 第 34 章 Java メソッドとの通信

Java との言語間インターオペラビリティを達成するには、Java Native Interface (JNI) でのサービスの使用、データ型のコーディング、および COBOL プログラムのコンパイルに関する特定の規則および指針に従う必要があります。

Java で書き込まれたメソッドを COBOL プログラムから呼び出したり、COBOL で書き込まれたメソッドを Java プログラムから呼び出し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。

756 ページの『例: COBOL で書かれた J2EE クライアント』

760 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

333 ページの『第 16 章 オブジェクト指向アプリケーションのコンパイル、リンク、および実行』

『JNI サービスへのアクセス』

748 ページの『Java とのデータ共用』

693 ページの『第 33 章 オブジェクト指向プログラムの作成』

603 ページの『第 27 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*

---

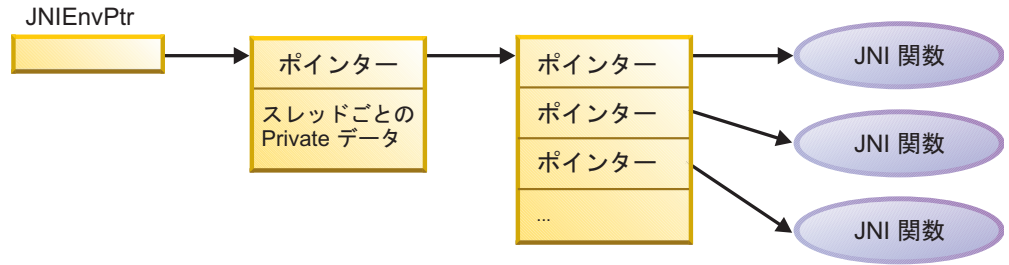
## JNI サービスへのアクセス

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 JNI.cpy コピーブックには、以下の定義が含まれています。

- Java JNI タイプに対応する COBOL データ定義
- JNINativeInterface、呼び出し可能サービス関数にアクセスするための関数ポインターが含まれている JNI 環境構造

次の図に示すように、JNI 環境ポインターからの 2 つのレベルの間接化技法によって JNI 環境構造を取得します。



特殊レジスター JNIEnvPtr を使用して JNI 環境ポインターを参照し、JNI 環境構造のアドレスを取得します。JNIEnvPtr は、USAGE POINTER として暗黙的に定義されます。それを受取データ項目として使用することはできません。JNI 環境構造の内容を参照する前に、以下のステートメントをコーディングして、そのアドレス可能性を確立する必要があります。

```

Linkage section.
COPY JNI
...
Procedure division.
 Set address of JNIEnv to JNIEnvPtr
 Set address of JNINativeInterface to JNIEnv
 ...

```

上記コードは、以下の項目のアドレスを設定します。

- JNIEnv。JNI.cpy が提供するポインター・データ項目です。JNIEnvPtr は、環境ポインターが含まれている COBOL 特殊レジスターです。
- JNINativeInterface。JNI.cpy に含まれている COBOL グループ構造です。この構造には、JNI 呼び出し可能サービスの関数ポインター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、関数ポインターを参照する 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、環境ポインターを必要とするサービスに、最初の引数として JNIEnvPtr 特殊レジスター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 環境ポインター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

746 ページの『ローカル参照とグローバル参照の管理』

『Java 例外の処理』

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

711 ページの『クライアントの定義』

関連参照

865 ページの『付録 F. 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。

例外のスロー: COBOL メソッドから Java 例外をスローするには、次のいずれかのサービスを使用します。

- `Throw`
- `ThrowNew`

`throw` したオブジェクトは、`java.lang.Throwable` のサブクラスのインスタンスにする必要があります。

Java 仮想マシン (JVM) は、呼び出しを含んでいるメソッドが完了して JVM に戻るまで、`throw` された例外を認識して処理することはいけません。

例外のキャッチ: Java 例外をスローした可能性があるメソッドを呼び出してから、次の手順を行うことができます。

1. 例外が発生したかどうかをテストします。
2. 例外が発生した場合は、その例外を処理します。
3. 例外をクリアします (クリアが適切な場合)。

次の JNI サービスを使用します。

- `ExceptionOccurred`
- `ExceptionCheck`
- `ExceptionDescribe`
- `ExceptionClear`

エラー分析を行うには、返された例外オブジェクトでサポートされるメソッドを使用してください。このオブジェクトは、`java.lang.Throwable` クラスのインスタンスです。

『例: Java 例外の処理』

### 例: Java 例外の処理

次の例は、Java からの例外を `catch` するための JNI サービスの使用と、エラー分析を行うための `java.lang.Throwable` の `printStackTrace` メソッドの使用を示しています。

```

Repository.
 Class JavaException is "java.lang.Exception".
. . .
Local-storage section.
01 ex usage object reference JavaException.
Linkage section.
COPY "JNI.cpy".
. . .
Procedure division.
 Set address of JNIEnv to JNIEnvPtr
 Set address of JNINativeInterface to JNIEnv
 . . .
 Invoke anObj "someMethod"
 Perform ErrorCheck
. . .
ErrorCheck.
 Call ExceptionOccurred
 using by value JNIEnvPtr
 returning ex
 If ex not = null then
 Call ExceptionClear using by value JNIEnvPtr
 Display "Caught an unexpected exception"
 Invoke ex "printStackTrace"
 Stop run
 End-if

```

## ローカル参照とグローバル参照の管理

Java 仮想マシンは、ネイティブ・メソッド (COBOL メソッドなど) で使用されるオブジェクト参照を追跡します。この追跡によって、ガーベッジ・コレクションの際、まだ使用中のオブジェクトが解放されないようにします。

オブジェクト参照には、以下の 2 つのクラスがあります。

### ローカル参照

ローカル参照は、呼び出したメソッドが稼働している間のみ有効です。ネイティブ・メソッドが戻ると、ローカル参照の自動解放が実行されます。

### グローバル参照

グローバル参照は、明示的に削除するまで有効です。グローバル参照は、JNI サービス `NewGlobalRef` を使用して、ローカル参照から作成することができます。

以下のオブジェクト参照は常にローカルです。

- メソッド・パラメーターとして受け取られるオブジェクト参照
- メソッドの `RETURNING` 値としてメソッドの起動から戻されるオブジェクト参照
- JNI 関数への呼び出しによって戻されるオブジェクト参照
- `INVOKE . . . NEW` ステートメントを使用して作成するオブジェクト参照

ローカル参照またはグローバル参照のいずれかをオブジェクト参照引数として JNI サービスに渡すことができます。

`RETURNING` 値としてローカル参照またはグローバル参照のいずれかを戻すメソッドをコーディングできます。ただし、いずれの場合も、呼び出すプログラムが受け取る参照はローカル参照です。



メソッドの起動で USING 引数としてローカル参照またはグローバル参照のいずれかを渡すことができます。ただし、いずれの場合も、呼び出されたメソッドが受け取る参照はローカル参照です。

ローカル参照は、それが作成されたスレッド内でのみ有効です。ローカル参照をスレッドから別のスレッドに渡すことはできません。

#### 関連タスク

743 ページの『JNI サービスへのアクセス』

『ローカル参照の削除、保管、および解放』

### ローカル参照の削除、保管、および解放

ローカル参照は、メソッド内で、随時に手動で削除できます。ローカル参照は、メソッドの LOCAL-STORAGE SECTION に定義したオブジェクト参照内にもみ保管します。

次のいずれかのデータ項目で参照を保管する場合にローカル参照をグローバル参照に変換するには、SET ステートメントを使用します。

- オブジェクト・インスタンス変数
- ファクトリー変数
- メソッドの WORKING-STORAGE SECTION 内のデータ項目

そうしないと、エラーが発生します。メソッドが戻ったときにこれらのストレージ域は保持されるので、ローカル参照は無効になります。

ほとんどのケースにおいて、メソッドが戻るときに発生するローカル参照の自動解放に依存することができます。ただし、場合によっては、JNI サービス DeleteLocalRef を使用して、メソッド内のローカル参照を明示的に解放する必要があります。以下に、明示的解放が適切な 2 つの状態を示します。

- メソッドにおいて、ラージ・オブジェクトにアクセスすることで、オブジェクトへのローカル参照を作成します。膨大な計算を行った後で、メソッドが戻ります。このラージ・オブジェクトを別の計算に必要としない場合には、このオブジェクトを解放してください。このローカル参照は、ガーベッジ・コレクションの間にオブジェクトを解放する妨げになるからです。
- メソッドに多数のローカル参照を作成しますが、それらのすべてのローカル参照を同時には使用しません。Java 仮想マシンは、各ローカル参照を追跡するためのスペースが必要です。不要になったローカル参照を解放すると、システムがメモリー不足にならないようにすることができます。

例えば、COBOL メソッドで、大き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、反復ごとに 1 つのエレメントを操作するとします。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 82. ローカルおよびグローバル参照の JNI サービス

サービス	入力引数	戻り値	目的
NewGlobalRef	<ul style="list-style-type: none"> <li>JNI 環境ポインタ</li> <li>ローカルまたはグローバル・オブジェクト参照</li> </ul>	グローバル参照、またはシステムがメモリー不足のときは NULL	入力オブジェクト参照が参照するオブジェクトに新規グローバル参照を作成する
DeleteGlobalRef	<ul style="list-style-type: none"> <li>JNI 環境ポインタ</li> <li>グローバル・オブジェクト参照</li> </ul>	なし	入力オブジェクト参照が参照するオブジェクトへのグローバル参照を削除する
DeleteLocalRef	<ul style="list-style-type: none"> <li>JNI 環境ポインタ</li> <li>ローカル・オブジェクト参照</li> </ul>	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

743 ページの『JNI サービスへのアクセス』

## Java アクセス制御

Java アクセス修飾子 `protected` および `private` は、Java Native Interface を使用すると、強制されません。したがって、COBOL プログラムは、Java クライアントからは呼び出し不可能な `protected` または `private` Java メソッドを呼び出すことができます。この使用法はお勧めできません。

## Java とのデータ共用

Java データ型と同じものを持つ COBOL データ型を共有することができます。(COBOL データ型には、Java データ型と同じものがありますが、同じでないものもあります。)

次の方法で Java とデータ項目を共有します。

- INVOKE ステートメントの USING 句に引数として渡します。
- Java メソッドから、USING 句のパラメーターとして受け取ります。
- INVOKE ステートメントの RETURNING 値として受け取ります。
- COBOL メソッドの PROCEDURE DIVISION ヘッダーの RETURNING 句の値として戻します。

配列およびストリングを渡したり受け取ったりするには、以下のように、それらをオブジェクト参照として宣言します。

- 特殊配列クラスのうちの 1 つのインスタンスが含まれているオブジェクト参照として、配列を宣言します。
- `jstring` クラスのインスタンスが含まれているオブジェクト参照として、ストリングを宣言します。

関連タスク

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

749 ページの『Java 用の配列およびストリングの宣言』

751 ページの『Java 配列の取り扱い』

753 ページの『Java スtringの取り扱い』  
716 ページの『メソッドの呼び出し (INVOKE)』  
573 ページの『第 25 章 データの共用』

## COBOL および Java での相互運用可能なデータ型のコーディング

Java との通信時には、COBOL プログラムは、特定のデータ型しか使用できません。

表 83. 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 <sup>1</sup>	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。 <i>level-number data-name PIC X.</i> 88 <i>data-name-false</i> value X'00'. 88 <i>data-name-true</i> value X'01' through X'FF'.
byte <sup>1</sup>	1 バイト英数字: PIC X または PIC A
short	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9( <i>n</i> ) の PICTURE 節付き。ここで、1<= <i>n</i> <=4
int	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9( <i>n</i> ) の PICTURE 節付き。ここで、5<= <i>n</i> <=9
long	USAGE BINARY、COMP、COMP-4、または COMP-5、形式 S9( <i>n</i> ) の PICTURE 節。ここで、10<= <i>n</i> <=18
float <sup>2</sup>	USAGE COMP-1
double <sup>2</sup>	USAGE COMP-2
char	1 文字基本国別: PIC N USAGE NATIONAL. (国別グループは不可です。)
クラス型 (オブジェクト参照)	USAGE OBJECT REFERENCE <i>class-name</i>
1. boolean 型と byte 型はいずれも PIC X に対応しているため、この 2 つは区別する必要があります。PIC X は、前述の 2 つの条件名を指定して引数またはパラメーターを定義した場合にのみ、boolean 型として解釈されます。それ以外の場合、PIC X データ項目は、Java の byte 型として解釈されます。 2. Java 浮動小数点データは、「IEEE Standard for Binary Floating Point Arithmetic」に従って形式設定されます。Enterprise COBOLただし、16 進浮動小数点表記を使用します。 INVOKE ステートメントを使用して浮動小数点引数を渡すとき、または、浮動小数点データを Java メソッドから受け取るとき、引数およびデータは必要に応じて、自動的に変換されます。	

### 関連タスク

147 ページの『COBOL での国別データ (Unicode) の使用』

## Java 用の配列およびString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String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。

表 84. COBOL および Java で相互運用可能な配列およびストリング

Java データ型	対応する COBOL データ型
boolean[ ]	オブジェクト参照 jbooleanArray
byte[ ]	オブジェクト参照 jbyteArray
short[ ]	オブジェクト参照 jshortArray
int[ ]	オブジェクト参照 jintArray
long[ ]	オブジェクト参照 jlongArray
char[ ]	オブジェクト参照 jcharArray
Object[ ]	オブジェクト参照 jobjectArray
String	オブジェクト・リファレンス jstring

Java とのインターオペラビリティのためにこれらのクラスのいずれかを使用するには、REPOSITORY 段落で項目をコーディングする必要があります。以下に例を示します。

```
Configuration section.
Repository.
 Class jbooleanArray is "jbooleanArray".
```

オブジェクト配列型に対する REPOSITORY 段落記入項目では、以下のいずれかの形式の外部クラス名を指定しなければなりません。

```
"jobjectArray"
"jobjectArray:external-classname-2"
```

最初のケースでは、REPOSITORY 記入項目は、配列エレメントが java.lang.Object 型のオブジェクトである配列クラスを指定しています。2 番目のケースでは、REPOSITORY 記入項目は、配列のエレメントが external-classname-2 型のオブジェクトである配列クラスを指定しています。jobjectArray 型の指定と、配列のエレメントの外部クラス名の間の分離文字として、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.
. . .
```

740 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 85. COBOL および Java で相互運用可能でない配列型

Java データ型	対応する COBOL データ型
float[ ]	オブジェクト参照 jfloatArray
double[ ]	オブジェクト・リファレンス jdoubleArray

関連タスク

699 ページの『クラス定義用の REPOSITORY 段落』

## Java 配列の取り扱い

COBOL プログラムで配列を表すには、その配列の Java タイプに対応するデータ・タイプの単一基本項目が含まれているグループ項目をコーディングします。その配列に適した OCCURS または OCCURS DEPENDING ON 節を指定します。

例えば、次のコードは、jlongArray オブジェクトから 500 以下の整数値を受け取る構造を指定します。

```
01 jlongArray.
 02 X pic S9(10) comp-5 occurs 1 to 500 times depending on N.
```

特殊な Java 配列クラスのオブジェクトを操作するには、JNI が提供するサービスを呼び出します。 サービスを使用して、配列の個々のエレメントにアクセスして設定し、呼び出したサービスを使用して、以下を行います。

表 86. JNI 配列サービス

サービス	入力引数	戻り値	目的
GetArrayLength	<ul style="list-style-type: none"><li>JNI 環境ポインター</li><li>配列オブジェクトの参照</li></ul>	2 進数フルワード整数としての配列の長さ	Java 配列オブジェクト内のエレメント数を取得する
NewBooleanArray、 NewByteArray、 NewCharArray、 NewShortArray、 NewIntArray、 NewLongArray	<ul style="list-style-type: none"><li>JNI 環境ポインター</li><li>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</li></ul>	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
GetBooleanArrayElements、 GetByteArrayElements、 GetCharArrayElements、 GetShortArrayElements、 GetIntArrayElements、 GetLongArrayElements	<ul style="list-style-type: none"><li>JNI 環境ポインター</li><li>配列オブジェクトの参照</li><li>ブール項目へのポインター ポインターが NULL でない場合は、配列エレメントのコピーが作成されたときは、ブール項目は true に設定される。コピーが作成された場合、変更を配列オブジェクトに書き戻す必要がある場合は、対応する ReleasexxxArrayElements サービスを呼び出さなければならない。</li></ul>	ストレージ・バッファへのポインター	配列エレメントを Java 配列からストレージ・バッファに抽出する。サービスにより、ポインターがストレージ・バッファに戻される。ポインターは、LINKAGE SECTION に定義される COBOL グループ・データ項目のアドレスとして使用することができる。

表 86. JNI 配列サービス (続き)

サービス	入力引数	戻り値	目的
ReleaseBooleanArrayElements、 ReleaseByteArrayElements、 ReleaseCharArrayElements、 ReleaseShortArrayElements、 ReleaseIntArrayElements、 ReleaseLongArrayElements	<ul style="list-style-type: none"> <li>• JNI 環境ポインター</li> <li>• 配列オブジェクトの参照</li> <li>• ストレージ・バッファへのポインター</li> <li>• 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。)</li> </ul>	なし。配列のストレージは解放される。	Java 配列から抽出したエレメントが含まれているストレージ・バッファを解放し、条件によっては、更新された配列値を配列オブジェクトにマップして戻す。
NewObjectArray	<ul style="list-style-type: none"> <li>• JNI 環境ポインター</li> <li>• 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</li> <li>• 配列エレメント・クラスに対するオブジェクト参照</li> <li>• 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。</li> </ul>	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 <sup>1</sup>	新しい Java オブジェクト配列を作成する。
GetObjectArrayElement	<ul style="list-style-type: none"> <li>• JNI 環境ポインター</li> <li>• 配列オブジェクトの参照</li> <li>• 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</li> </ul>	オブジェクト参照 <sup>2</sup>	オブジェクト配列内の特定の索引のエレメントを返す。
SetObjectArrayElement	<ul style="list-style-type: none"> <li>• JNI 環境ポインター</li> <li>• 配列オブジェクトの参照</li> <li>• 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</li> <li>• 新しい値に対するオブジェクト参照</li> </ul>	なし <sup>3</sup>	オブジェクト配列内のエレメントを設定する。
<p>1. システムがメモリー不足の場合、NewObjectArray は例外を throw します。</p> <p>2. 索引が有効でない場合、GetObjectArrayElement は例外を throw します。</p> <p>3. 索引が有効でない場合、または新しい値が配列のエレメント・クラスのサブクラスでない場合、SetObjectArrayElement は例外を throw します。</p>			

740 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』  
753 ページの『例: Java 整数配列の処理』

#### 関連タスク

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

749 ページの『Java 用の配列およびストリングの宣言』

743 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。

```
cb1 thread,dll
Identification division.
Class-id. 00ARRAY inherits Base.
Environment division.
Configuration section.
Repository.
 Class Base is "java.lang.Object"
 Class jintArray is "jintArray".
Identification division.
Object.
Procedure division.
 Identification division.
 Method-id. "ProcessArray".
 Data Division.
 Local-storage section.
 01 intArrayPtr pointer.
 01 intArrayLen pic S9(9) comp-5.
 Linkage section.
 COPY JNI.
 01 inIntArrayObj usage object reference jintArray.
 01 intArrayGroup.
 02 X pic S9(9) comp-5
 occurs 1 to 1000 times depending on intArrayLen.
 Procedure division using by value inIntArrayObj.
 Set address of JNIEnv to JNIEnvPtr
 Set address of JNINativeInterface to JNIEnv

 Call GetArrayLength
 using by value JNIEnvPtr inIntArrayObj
 returning intArrayLen
 Call GetIntArrayElements
 using by value JNIEnvPtr inIntArrayObj 0
 returning IntArrayPtr
 Set address of intArrayGroup to intArrayPtr

* . . . process the array elements X(I) . . .

 Call ReleaseIntArrayElements
 using by value JNIEnvPtr inIntArrayObj intArrayPtr 0.
 End method "ProcessArray".
End Object.
End class 00ARRAY.
```

## Java ストリングの取り扱い

COBOL は、Java ストリング・データを Unicode で表します。Java ストリングを 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ストリング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。

**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 87. **jstring** 参照と国別データ間の変換サービス

サービス	入力引数	戻り値
NewString <sup>1</sup>	<ul style="list-style-type: none"> <li>JNI 環境ポインター</li> <li>COBOL 国別データ項目などの、Unicode スtringへのポインター</li> <li>Stringの文字数。2 進数フルワード</li> </ul>	jstring オブジェクト参照。
GetStringLength	<ul style="list-style-type: none"> <li>JNI 環境ポインター</li> <li>jstring オブジェクト参照</li> </ul>	jstring オブジェクト参照の Unicode 文字数。2 進数フルワード。
GetStringChars <sup>1</sup>	<ul style="list-style-type: none"> <li>JNI 環境ポインター</li> <li>jstring オブジェクト参照</li> <li>ブール・データ項目を指すポインター、または NULL</li> </ul>	<ul style="list-style-type: none"> <li>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。</li> <li>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。</li> </ul>
ReleaseStringChars	<ul style="list-style-type: none"> <li>JNI 環境ポインター</li> <li>jstring オブジェクト参照</li> <li>GetStringChars から戻された Unicode 文字の配列へのポインター</li> </ul>	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 用のサービス:** jstring オブジェクト参照と COBOL 英数字データ (PIC X(n)) との間の変換を行うには、以下の z/OS サービス (JNI の拡張) を使用してください。

表 88. **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	戻り値
NewStringPlatform	<ul style="list-style-type: none"> <li>JNI 環境ポインター</li> <li>jstring オブジェクトに変換するヌル終了 EBCDIC 文字Stringへのポインター</li> <li>結果のための jstring オブジェクト参照へのポインター</li> <li>Stringの Java エンコード名へのポインター。ヌル終了 EBCDIC 文字String<sup>1</sup> として表す</li> </ul>	2 進数フルワード整数としての戻りコード: <b>0</b> 正常。 <b>-1</b> 誤った形式の入力データまたは正しくない入力文字。 <b>-2</b> 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。



表 88. **jstring** 参照と英数字データ間の変換サービス (続き)

サービス	入力引数	戻り値
GetStringPlatformLength	<ul style="list-style-type: none"> <li>JNI 環境ポインター</li> <li>長さのための <b>jstring</b> オブジェクト参照</li> <li>結果のための 2 進数フルワード値へのポインター</li> <li>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<sup>1</sup> として表す</li> </ul>	<p>2 進数フルワード整数としての戻りコード:</p> <p><b>0</b> 正常。</p> <p><b>-1</b> 誤った形式の入力データまたは正しくない入力文字。</p> <p><b>-2</b> 非サポート・エンコード。<b>jstring</b> オブジェクト参照ポインターは NULL に設定されます。</p> <p>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。</p>
GetStringPlatform	<ul style="list-style-type: none"> <li>JNI 環境ポインター</li> <li>ヌル終了ストリングに変換する <b>jstring</b> オブジェクト参照</li> <li>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</li> <li>2 進数フルワード整数としての出力バッファの長さ</li> <li>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<sup>1</sup> として表す</li> </ul>	<p>2 進数フルワード整数としての戻りコード:</p> <p><b>0</b> 正常。</p> <p><b>-1</b> 誤った形式の入力データまたは正しくない入力文字。</p> <p><b>-2</b> 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。</p> <p><b>-3</b> 変換バッファがいっぱいです。</p>
1. ポインターが NULL の場合、Java file.encoding プロパティーからのエンコードが使用されます。		

これらの EBCDIC サービスは、IBM Java Software Development Kit 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 Software Development Kit 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 ストリング 'MyConverter' から Java ストリング・オブジェクトを作成します。(このコード・フラグメントは、756 ページの『例: COBOL で書かれた J2EE クライアント』に詳細な説明がある、J2EE クライアント・プログラムからのものです。)

```
Move z"MyConverter" to stringBuf
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もありません。

**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 スtring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

743 ページの『JNI サービスへのアクセス』

749 ページの『COBOL および Java での相互運用可能なデータ型のコーディング』

749 ページの『Java 用の配列およびStringの宣言』

147 ページの『COBOL での国別データ (Unicode) の使用』

333 ページの『第 16 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 Developer's Guide*」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「*The Java EE 5 Tutorial*」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 InitialCtx 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 homeObject 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 InitialCtx 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

333 ページの『第 16 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、標準の Java の System.out ファイルと System.err ファイルを z/OS データ・セットまたはスプール・ファイルへ送信する必要があることがしばしばあります。これは、Java 仮想マシン (JVM) の開始後に、com.ibm.jzos.ZUtil クラスで redirectStandardStreams メソッドを呼び出すことによって行うことができます。ZUtil クラスの詳細については、z/OS security and legacy services API Reference の ZUtil を参照してください。

この COBOL プログラムの例では、com.ibm.jzos.sample.HelloWorld クラスの main() メソッドが呼び出されますが、これを他の Java クラス・メソッドを呼び出すように変更できます。

```
//COB2JAV JOB (), 'Dovetail',
// MSGCLASS=H, REGION=128M,
// NOTIFY=&SYSUID
//*
/* Tested on z/OS V2R2 with Ent Cobol V5R1 and Java V7.0
// SET COBPRFX='SYSPROG.MNT.COBO51'
// SET LIBPRFX='CEE'
// SET SYSLIB1='G1JAVA1.PRIVATE.JZOS.DEVEL.JCL' Has JNI cpybook
/* 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"
 Class HelloWorld is "com.ibm.jzos.sample.HelloWorld"
```

(1)

(2)

(3)

(4)

```

Class HelloException is
 "com.ibm.jzos.test.helper.HelloException"
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 stringBuf 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"
 Perform ErrorCheck
 Display "Returned from ZUtil.redirectStandardStreams"
(3)
*
* We invoke com.ibm.jzos.sample.HelloWorld,
* but this could be any arbitrary Java code
*
 Perform BuildEmptyArgsArray.
 Invoke HelloWorld "main"
 using by value args
 Perform ErrorCheck
 Display "Returned from HelloWorld.main"
(4)
* We invoke com.ibm.jzos.test.HelloException
* which we expect to throw an Exception and exit RC=32
 Invoke HelloException "main"
 using by value args
 Perform ErrorCheck
 Display "Returned from HelloException.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 stringBuf
 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 argLen
 Call NewObjectArray
 using by value JNIEnvPtr
 argLen 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=&&GOSSET(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'
INCLUDE '/usr/lpp/cobol/V5R1/lib/igzjava.x'
/*
/* 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(
"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",
* 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',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=(SIRUSR,SIWUSR,SIRGRP)
//JAVAERR DD PATH='/tmp/cob2jav.javaerr',
// 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) 最初に、JNI.cpy ファイルを、COBOL インストール・ディレクトリー (通常は /usr/lpp/cobol/include) から、このソース PDS のメンバー JNI としてコピーする必要があります。
- (3) ZUtil redirectStandardStreams メソッドは、Java の System.out と System.err を DD:STDOUT と DD:STDERR にそれぞれリダイレクトします。
- (4) com.ibm.jzos.sample.HelloWorld クラスは、「Hello World! (stdout)」を System.out に、「Hello World! (stderr)」を System.err に出力します。com.ibm.jzos.sample.HelloWorld クラスは、JZOS サンプルを使用してダウンロードできます。
- (5) Java と 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 Language Environment 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 の両方をマージして DD:SYSOUT に出力されるように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、これらが DD DUMMY を指すように設定できます。

## 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す。詳しくは、*JZOS Installation and User's Guide* を参照してください。



---

## 第 7 部 特殊处理



---

## 第 35 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してください。

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

『チェックポイントの設定』

771 ページの『プログラムの再始動』

774 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

769 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』

770 ページの『チェックポイント時に生成されるメッセージ』

773 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。

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 (単一のチェックポイントを保管します)**

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。

最後に取ら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**

267 ページの『DFSORT によるチェックポイント・リスタートの使用』

『チェックポイントの設計』

769 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**

769 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。

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。

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 DD 名部分と同じ。

### **dataset-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。

『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 DSNAME=CHECK1,VOLUME=SER=ND0003,
// UNIT=TAPE,DISP=(NEW,KEEP),LABEL=(,NL)
 .
 .
 .
ENVIRONMENT DIVISION.
 .
 .
 .
RERUN 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:

```
//CHECKPT DD DSNAME=CHECK3,VOLUME=SER=111111,
// UNIT=TAPE,DISP=(MOD,PASS),LABEL=(,NL)
 .
 .
 .
ENVIRONMENT DIVISION.
 .
 .
 .
RERUN ON CHECKPT 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、その後に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。

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、標準外ラベル付きまたはラベルなしの 7 トラック・テープに置かれている場合は、SYSCHK DD ステートメントに DCB=(TRTCH=C,...) が含まれなくてはなりません。

### 関連タスク

267 ページの『DFSORT によるチェックポイント・リスタートの使用』

772 ページの『自動再始動の要求』

772 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。

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に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。

774 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。

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。SYSCHK DD ステートメントがジョブの中にあっても、JOB ステートメント

に RESTART パラメーターが含まれていないと、その 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。

774 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

『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 DSNAME=data-set-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ー・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。

774 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 プロシーチャーの 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。

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。

775 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』

775 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 DSN=PRIV.LIB3,DISP=OLD
//SYSCHK DD DSN=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 DSN=INVENT[,UNIT=TAPE],DISP=OLD,
// [VOLUME=SER=91468,]
// LABEL=RETPD=14
//REPORT DD SYSOUT=A
//WORK DD DSN=T91468,DISP=(,KEEP),
// UNIT=SYSDA,SPACE=(3000,(5000,500)),
// VOLUME=(PRIVATE,RETAIN,,6)
//DDCKPNT DD UNIT=TAPE,DISP=(MOD,PASS,CATLG),
// DSN=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 DSN=INVENT[,UNIT=TAPE],DISP=OLD,
// [VOLUME=SER=91468,]LABEL=RETPD=14
//REPORT DD SYSOUT=A
//WORK DD DSN=S91468,
// DISP=(,KEEP),UNIT=SYSDA,
// SPACE=(3000,(5000,500)),
// VOLUME=(PRIVATE,RETAIN,,6)
//DDCHKPNT DD UNIT=TAPE,DISP=(MOD,PASS,CATLG),
// DSN=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。

774 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)
//SYSCBK DD DSN=C91468,DISP=OLD
//S1 EXEC PGM=MYPROG
//INDATA DD DSN=INVENT,UNIT=TAPE,DISP=OLD,
// VOLUME=SER=91468,LABEL=RETPD=14
//REPORT DD SYSOUT=A
//WORK DD DSN=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、C0000002 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 8 部 パフォーマンスおよび生産性の向上





---

## 第 36 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。

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。以下に例を示します。
  - 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

786 ページの『最適化』

### 関連タスク

780 ページの『最適なプログラミング・スタイルの使用』

781 ページの『効率的なデータ型の選択』

783 ページの『テーブルの効率的処理』

786 ページの『コードの最適化』

787 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』

793 ページの『CICS、IMS、または VSAM での効率的な実行』

言語環境プログラム 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

788 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』

言語環境プログラム プログラミング・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。

### 関連タスク

『構造化プログラミングの使用』

『一括表示表現』

781 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。

トップダウン・プログラミング構成を使用してください。ライン外の PERFORM ステートメントは、トップダウン・プログラミングを行う本来の手段です。ライン外 PERFORM ステートメントがインラインの PERFORM ステートメントと同じくらい効率的になることがよくあります。これは、最適化プログラムがリンケージ・コードを簡略化または除去するためです。

次の構成は使用しないでください。

- ALTER ステートメント
- 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、プロシージャーの終わりに制御が渡されないために、PERFORM ステートメントに戻れないなど。

## 一括表示表現

プログラム内の式を因数処理することによって、多数の不要な計算を除去できる可能性があります。

例えば、次のコードの最初のブロックは、2 番目のブロックより効率的になっています。

```
MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
 COMPUTE TOTAL = TOTAL + ITEM(I)
END-PERFORM
COMPUTE TOTAL = TOTAL * DISCOUNT

MOVE ZERO TO TOTAL
PERFORM VARYING I FROM 1 BY 1 UNTIL I = 10
 COMPUTE TOTAL = TOTAL + ITEM(I) * DISCOUNT
END-PERFORM
```

最適化プログラムは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。

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

SYNCHRONIZED 節を使用すれば、より効率的なコードを得ることができます。

一貫性のあるデータ型を使用すると、データ項目に演算を実行する際に変換を行う必要性を減らすことができます。また、固定小数点データ型と浮動小数点データ型をいつ使用するかを慎重に判断することで、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

54 ページの『数値データの形式』

### 関連タスク

『効率的な計算データ項目の選択』

782 ページの『一貫性のあるデータ型の使用』

782 ページの『算術式の効率化』

782 ページの『指数計算の効率化』

## 効率的な計算データ項目の選択

データ項目を主に算術に、または添え字として使用する場合、その項目のデータ記述項目に USAGE BINARY をコーディングしてください。2 進データを処理する操作は、10 進データを処理する操作よりも高速で行われます。

しかし、固定小数点算術ステートメントが大きな精度 (有効数字) の中間結果を持つ場合、コンパイラーは、オペランドをパック 10 進数形式に変換したうえで、10 進数演算を使用します。固定小数点算術ステートメントについては、コンパイラーは通常、精度が 8 桁以下にとどまる場合には、2 進オペランドを使用した簡単な計算に 2 進数算術演算を使用します。18 桁を超えると、コンパイラーは常に 10 進数算術演算を使用します。9 から 18 桁の精度では、コンパイラーはいずれの形式でも使用できます。

BINARY データ項目について最も効率的なコードを作成するには、以下の特性を持たせるようにしてください。

- 符号 (PICTURE 節の S)。
- 8 桁以下。

8 桁より大きいデータ項目、あるいは DISPLAY または NATIONAL データ項目と一緒に使用されるデータ項目の場合は、PACKED-DECIMAL を使用してください。

PACKED-DECIMAL データ項目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)。
- ARCH (7) のマシンの場合、PICTURE 指定を 15 桁以内にする。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。ARCH (8) 以上のレベルのマシンでの 16 から 31 桁の PACKED-DECIMAL データ項目の場合、コンパイラは、ライブラリー呼び出しよりは効率的であっても、データ項目の桁数が 15 桁以内の場合ほど高速ではない命令を使用します。

## 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りますがあるかもしれません。

一貫性のあるデータ型を使用し、両方のオペランドに同じ使用法を与え、さらに適切な PICTURE 指定を与えることで、変換を大部分は回避できます。つまり、比較、加算、または減算を行う 2 つの数値は、同じ使用法を持つだけでなく、さらに小数部の桁数 (PICTURE 節の V の後の 9 の数) も同じでなければなりません。

## 算術式の効率化

オペランドをほとんど変換する必要がない場合、浮動小数点で評価される算術式の計算は最も効率的です。COMP-1 または COMP-2 であるオペランドを使用すると、最も効率のよいコードが作成されます。

浮動小数点データに高速変換を行うには、整数項目を BINARY または PACKED-DECIMAL (9 桁以下) として定義します。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なります。

## 指数計算の効率化

評価をもっと速く行い、結果をもっと正確にするには、大きな指数に対しては指数の浮動小数点を使用してください。

例えば、以下に示す最初のステートメントは、2 番目のステートメントより速かつより正確に計算されます。

```
COMPUTE fixed-point1 = fixed-point2 ** 100000.E+00
```

```
COMPUTE fixed-point1 = fixed-point2 ** 100000
```

これは、浮動小数点の指数があるため、べき乗計算の計算に浮動小数点演算が使用されるからです。

## 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。

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。どのような場合に **VOLATILE** を使用するかについて詳しくは、「Enterprise COBOL for z/OS 言語解説書」の『**VOLATILE** 節』を参照してください。

---

## 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。

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・オプションを使用するよりも、独自の検査をコーディングする方が速い場合があります。

以下のガイドラインに従うことによって、テーブルの効率を改善することもできます。

- すべての添え字に 2 進数データ項目を使用する。

添え字を使用してテーブルをアドレッシングする場合は、8 桁以下の BINARY 符号付きデータ項目を使用してください。さらに場合によっては、データ項目の桁数を 4 桁以下にすると、処理時間を短縮できます。

- 可変長テーブル項目に 2 進数データ項目を使用する。

可変長項目を持つテーブルの場合は、OCCURS DEPENDING ON (ODO) のコードを改善することができます。可変長項目が参照されるたびに不要な変換が行われないようにするには、OCCURS . . . DEPENDING ON オブジェクトに対して BINARY を指定します。

- 可能であれば固定長データ項目を使用する。

可変長データ項目を使用する場合、それらの使用頻度が高くなる前に、固定長データ項目にコピーすると、オーバーヘッドを緩和することができます。

- 使用する探索メソッドのタイプに従ってテーブルを編成する。

テーブルが順次に探索される場合には、検索基準を満たす可能性が最も高いデータ値をテーブルの始まりに置くようにします。テーブルが二分探索アルゴリズムを使用して探索される場合は、検索キー・フィールドに基づいてアルファベット順にソートされたテーブルに、データ値を入れてください。

#### 関連概念

785 ページの『テーブル参照の最適化』

#### 関連タスク

79 ページの『テーブル内の項目の参照』

781 ページの『効率的なデータ型の選択』

#### 関連参照

420 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トライドは d1、d2、および d3 です。ある特定次元のストライドは、その次元での出現番号が 1 だけ違い、かつ他の出現番号が等しいようなテーブル・エレメント相互間の距離 (バイト単位) です。例えば、上記の例の 2 次元のストライド d2 は、ELEMENT(S1 1 S3) と ELEMENT(S1 2 S3) との間の距離 (バイト単位) です。

指標計算は添え字計算に類似していますが、指標値ではそれらの中にストライドを含めているので、乗算をする必要がないという点が異なります。指標計算には、指標をレジスターにロードすることも含まれます。これらのデータ転送は、個々の添え字計算の項を最適化する場合とほぼ同様に、最適化することができます。

### 可変長項目の最適化

従属 OCCURS DEPENDING ON データ項目の入っているグループ項目は可変長です。プログラムは、可変長データ項目が参照されるたびに特殊コードを実行しなければなりません。

このコードはライン外のもので、最適化の妨げになる可能性があります。さらに、可変長データ項目を処理するためのコードは、固定サイズ・データ項目を処理するコードよりかなり効率が下がり、処理時間も大幅に増加することがあります。例えば、可変長データ項目を比較したり移動したりするためのコードには、ライブラリー・ルーチンの呼び出しが必要なため、固定長データ項目の場合の同じコードよりかなり低速になります。

### 直接指標付けと間接指標付けの比較

相対指標参照は、直接指標参照と同じくらい迅速に実行されます。

ELEMENT (I5, J3, K2) の直接索引付けには、次のプリプロセスが必要になります。

```
SET I5 TO I
SET I5 UP BY 5
SET J3 TO J
SET J3 DOWN BY 3
SET K2 TO K
SET K2 UP BY 2
```

この処理のため、直接索引付けは、ELEMENT (I + 5, J - 3, K + 2) の相対索引付けよりも効率が悪くなります。

#### 関連概念

786 ページの『最適化』

#### 関連タスク

783 ページの『テーブルの効率的処理』

---

## コードの最適化

プログラムの最終テストの準備ができたなら、OPTIMIZE(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。

開発中に再コンパイルしないで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。ただし、頻繁に再コンパイルする場合は、アセンブラ言語の拡張部分 (LIST コンパイラー・オプション) を使用してプログラムの微調整を行う場合を除き、OPTIMIZE(1|2) のオーバーヘッドが利点を上回ることがあります。

プログラムのユニット・テストについては、最適化されていないコードをデバッグする方が簡単です。

最適化プログラムがプログラム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  
『最適化』

関連参照  
387 ページの『LIST』  
402 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) を使用すると、COBOL 最適化プログラムが以下の最適化を行います。

- 不必要な制御権移動および非効率的な分岐を除去します。ソース・プログラムを見るだけではわからない、コンパイラーが生成する分岐も含みます。
- 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 サブ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・プロシージャーの統合

含まれているプログラム・プロシージャーの統合では、含まれているプログラム・コードが、含まれているプログラムへの CALL と置き換わります。結果として生じるプログラムは、CALL リンケージのオーバーヘッドもなく、より線形の制御フローとなり、より速く実行されます。

プログラム・サイズ: 含まれているプログラムを複数の CALL ステートメントが呼び出している場合、それぞれのプログラムがこのような各ステートメントと置き換わるのであれば、収容プログラムが相当大きくなる可能性があります。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

『PERFORM プロシージャー統合』

#### 関連参照

402 ページの『OPTIMIZE』

### PERFORM プロシージャー統合

PERFORM プロシージャー統合とは、PERFORM ステートメントが、実行されるプロシージャー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ー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ーを統合しません。

#### 関連参照

384 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。

カスタマイズ済みシステムには、最適なパフォーマンスを得るために特定のオプションが必要なものもあります。以下の手順を実行します。

1. システム・デフォルトの内容を調べるには、プログラムの短縮リストを入手し、リストされたオプション設定値を検討する。
2. 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. インストール先で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オプションを選択する。

**重要:** COBOL プログラムの調整方法については、システム・プログラマーと相談してください。 そうすれば、選択したオプションがインストール先のプログラムに適しているかどうかを確認できます。

考慮する必要があるもう 1 つのコンパイラー機能として、USE FOR DEBUGGING ON ALL PROCEDURES ステートメントがあります。 このステートメントは、コンパイラーの最適化プログラムに大きく影響する可能性があります。 ON ALL PROCEDURES オプションは、プロシージャー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

786 ページの『最適化』

#### 関連タスク

786 ページの『コードの最適化』

464 ページの『リストの入手』

#### 関連参照

『パフォーマンスに関連するコンパイラー・オプション』

## パフォーマンスに関連するコンパイラー・オプション

以下の表には、それぞれのオプションの目的、パフォーマンス上の利点と欠点、および使用上の注意 (該当する場合) が示されています。

表 89. 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) ( 351 ページの『AFP』を参照)	z/Architecture プロセッサに備わる追加浮動小数点 (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) を指定すると、コンパイラーは浮動小数点演算を行うプログラムに対して、より効率的なコード・シーケンスを生成します。	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ます。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 プログラムは安全にそのようなルーチンを呼び出すことができます。
ARCH ( 352 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シーケンスの代わりに、より新しくより高速な命令を使用するコードを生成します。	なし	ご使用のアプリケーションは、ARCH オプションで指定したものよりも低いアーキテクチャー・レベルのプロセッサで実行した場合、異常終了するおそれがあります。

表 89. 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
ARITH(EXTEND) ( 354 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 進数データ型でパフォーマンスがいくらか低下します。	どれほど低下するかは、使用する 10 進数データの量に直接左右されます。
AWO ( 355 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・レコードが入った、プログラム内のすべての QSAM ファイルに、APPLY WRITE-ONLY 節が有効になります。
BLOCK0 ( 356 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、BLOCK CONTAINS 0 節がアクティブになります。
DATA(31) ( 366 ページの『DATA』を参照)	DFSMS に、16 MB 境界より上の QSAM バッファを割り振らせる (RENT および DATA(31) コンパイラー・オプションを使用して)	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 が備わった z/OS システムでは、アプリケーションがストライプ拡張形式 QSAM データ・セットを使用する場合には、RENT および DATA(31) 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 ファイル用の入出力バッファを割り振らせるようにしてください。
DYNAM ( 373 ページの『DYNAM』を参照)	サブプログラム (CALL ステートメントによって呼び出された) が実行時に動的にロードされるようにする	サブプログラムが変更されても、アプリケーションをリンク・エディットする必要がないので、サブプログラムの保守が容易になります。	呼び出しは言語環境プログラム・ルーチンを介して行う必要があるため、ちょっとしたパフォーマンス・ペナルティがあります。	不要になった仮想記憶域を解放するには、CANCEL ステートメントを出してください。
FASTSRT ( 377 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	直接作業ファイルをソート作業ファイルとして使用する場合は、FASTSRT の使用をお勧めします。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 ( 381 ページの『HGPR』を参照)	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で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、プログラムが使用する 64 ビット GPR の高位半分を保存するため、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。 <b>例外:</b> これは、このプログラムの呼び出し元が 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 ( 384 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ジャー (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 の指定により、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ジャーをインライン化するかどうかを判断できます。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ジ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。	通常、INLINE が指定されている場合は、モジュールのサイズが増えます。共通に使用されていてもあまり実行されないプロシージャーに関しては、>>INLINE OFF ディレクティブを使用して、コンパイラーがそのプロシージャーをインライン化したりモジュール・サイズを増やしたりしないようにできます。	INLINE コンパイラー・オプションを使用すれば、プロシージャーがインライン化に適格であるとみなされるように指示できますが、特定の PERFORM ステートメントにおいてプロシージャーをインライン化するかどうかを決定するのはコンパイラーです。

表 89. 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
INITCHECK ( 382 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、コンパイル時間が長引いたりメモリー消費量が増えたりすることがあります。	INITCHECK 分析はいずれもコンパイル時のみ行われます。INITCHECK オプションは、プログラムのコンパイル後はプログラムの動作やパフォーマンスに影響しません。
MAXPCF ( 389 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( <i>n</i> ) を指定し、 <i>n</i> がゼロでない場合に、プログラム複雑度係数が <i>n</i> を超えると、OPTIMIZE(1) または OPTIMIZE(2) の指定はすべて 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 ( 395 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	COBOL プログラムで 사용되는ゾーン 10 進数 (数値 USAGE DISPLAY) データ項目、バック 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、NUMCHECK が指定されたときは、NONUMCHECK が指定されたときよりもはるかに時間がかかります。	なし
NUMPROC(PFD) ( 398 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) を使用すると、符号を「調整」するための余分のコードの生成が禁止されます。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。NUMPROC(NOPFD) を使用した場合は、余分のコードが生成されます。	NUMPROC(PFD) を使用すると、コンパイラーは、すべての 10 進数項目が優先符号値を含み、符号「修正」プロセスを迂回するものと想定し、それを必要とします。ただし、すべての外部データ・ファイルに COMP-3 または DISPLAY の数値データに適切な符号が含まれているとは限らず、プログラムは優先符号を保証しない方法で REDEFINES、グループ移動、またはパラメーター受け渡しを使用している可能性があるため、NUMPROC(PFD) は多くのプログラムで不適切であると考えられます。
OPTIMIZE(0 1 2) ( 402 ページの『OPTIMIZE』を参照)	パフォーマンスがよくなるように、生成されるコードを最適化する	一般に、もっと効率的な実行時コードが得られます。	コンパイル時間が長くなります。OPTIMIZE(1 2) は OPTIMIZE(0) に比べて、コンパイルのためにより多くの処理時間を必要とします。	OPTIMIZE(0) は一般に、頻繁にコンパイルを行う必要があるプログラム開発時に使用します。これにより、シンボリック・デバッグも可能になります。実稼働には、OPTIMIZE(1 2) の使用をお勧めします。
STGOPT ( 421 ページの『STGOPT』を参照)	DATA DIVISION でのストレージ割り振りを最適化する	一般に、ストレージ使用量が少なくなります。	なし	STGOPT は未使用データ項目を削除しますが、それは、ダンプ読み取り用マーカーとしてのみ使用されるタイム・スタンプまたはデータ項目の場合には、望ましくないことがあります。

表 89. 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
PARMCHECK ( 404 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 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。良好なパフォーマンスを得るには、NOPARMCHECK を指定する必要があります。	なし
RENT ( 409 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。	なし
RMODE(ANY) ( 410 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	なし
SSRANGE ( 420 ページの『SSRANGE』を参照)	すべてのテーブル参照および参照変更式が適切な範囲内にあるかどうかを検査する	SSRANGE は、テーブル参照を検査するための追加コードを生成します。NOSSRANGE を使用すると、コードは生成されません。	SSRANGE を指定すると、有効範囲の検査は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、テーブル参照のたびに検査する必要はなく、数度の検査だけで済む場合には、独自の検査をコーディングする方が、SSRANGE を使用するより速くなります。パフォーマンスを重視するアプリケーションについては、NOSSRANGE の使用をお勧めします。
TEST ( 423 ページの『TEST』を参照)	デバッグ・ツールの使用時に全デバッグ機能を使用可能にし、CEEDUMP 内のデータ項目のシンボリック・ダンプを取得する。CEEDUMP 内のデータ項目のシンボリック・ダンプは、NOTEST(DWARF)でも取得できます。	なし	TEST オプションを使用すると、最適化がいくらか低下します。TEST のEJPD サブオプションを使用すると、最適化がさらに低下します。	実稼働には、NOTEST または TEST(NOJPD)の使用をお勧めします。  実稼働中、プログラムの異常終了時にデータ項目のシンボリック・ダンプを定様式ダンプで取得するには、TEST または NOTEST(DWARF) を使用してコンパイルしてください。
THREAD ( 428 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 言語環境プログラム・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・ペナルティーがあります。	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・ペナルティーが発生します。
TRUNC(OPT) ( 430 ページの『TRUNC』を参照)	算術演算の受信フィールドを切り捨てるためのコードの生成を回避する	余分のコードを生成しないので、一般にパフォーマンスは向上します。	TRUNC(BIN) と TRUNC(STD) はともに、BINARY データ項目が変更されるたびに、余分のコードを生成します。TRUNC(BIN) は通常、最も低速のオプションです。	TRUNC(STD) は 85 COBOL 標準に準拠しますが、TRUNC(BIN) および TRUNC(OPT) は準拠しません。TRUNC(OPT) を使用すると、コンパイラーは、データが PICTURE および USAGE の仕様に従っていると見なします。可能な場合は、TRUNC(OPT) を使用することをお勧めします。

## 関連概念

786 ページの『最適化』

43 ページの『ストレージとそのアドレス可能性』

## 関連タスク

317 ページの『コンパイラー・メッセージのリストの生成』

- 『パフォーマンスの評価』  
 11 ページの『バッファおよび装置スペースの最適化』  
 787 ページの『パフォーマンスを向上させるコンパイラ機能の選択』  
 261 ページの『FASTSORT によるソート・パフォーマンスの向上』  
 203 ページの『ストライプ拡張形式 QSAM データ・セットの使用』  
 783 ページの『テーブルの効率的処理』

#### 関連参照

- 60 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』  
 204 ページの『QSAM ファイル用のバッファの割り振り』  
 345 ページの『第 17 章 コンパイラ・オプション』  
 349 ページの『矛盾するコンパイラ・オプション』

## パフォーマンスの評価

プログラムのパフォーマンスの評価に役立つ次のワークシートに記入してください。各質問に「はい」と答える場合は、おそらくパフォーマンスは向上しています。

パフォーマンスのトレードオフを比較検討する際には、各オプションの機能およびパフォーマンスの利点と欠点を十分に理解するようにしてください。パフォーマンスの向上よりも、機能を重視する場合も多くあります。

表 90. パフォーマンス調整のワークシート

コンパイラ・オプション	考慮事項	はい
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、最低レベルのアーキテクチャーが z10™ である場合に、ARCH(8) 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ていますか。	
BLOCK0	QSAM ファイルに BLOCK0 を使用しますか。	
DATA	QSAM ストライプ・データ・セットを使用するときは、RENT および DATA(31) オプションを使用していますか。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。	
FASTSORT	直接作業ファイルをソート作業ファイルとして使用するとき、FASTSORT オプションを使用しましたか。	
INLINE	可能な場合、INLINE 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、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。	
PARMCHK	NOPARMCHK 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	実稼働に NOTEST または TEST(NOJPD) を使用しますか。	
TRUNC	可能な場合、TRUNC(OPT) を使用していますか。	

表 90. パフォーマンス調整のワークシート (続き)

コンパイラー・オプション	考慮事項	はい
ZONEDATA	可能な場合、ZONEDATA(PFD) を使用していますか。	

#### 関連概念

43 ページの『ストレージとそのアドレス可能性』

#### 関連タスク

787 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

788 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。

**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、言語環境プログラムのライブラリー・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

501 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』

529 ページの『第 22 章 COBOL プログラムの開発 (IMS の場合)』

237 ページの『VSAM パフォーマンスの向上』

言語環境プログラム カスタマイズ

関連参照

言語環境プログラム 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。

詳しくは、560 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

560 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 37 章 コーディングの単純化

コーディング技法を使用して、生産性を向上させることができます。COPY ステートメント、形式 2 SORT ステートメント、COBOL 組み込み関数、および言語環境プログラム呼び出し可能サービスを使用することにより、反復コーディングや、多数の算術計算または他の複雑なタスクをコーディングする必要性を回避することができます。

プログラムに頻繁に使用されるコード・シーケンス (共通データ項目のブロック、入出力ルーチン、エラー・ルーチン、または COBOL プログラム全体) が含まれている場合は、それらのコード・シーケンスを一度作成し、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含めることができます。このようにコピーブックを使用することによって、コーディングの繰り返しがなくなります。

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してコーディングを簡略化することができます。この方法は、形式 1 SORT ステートメントに比べてはるかに簡単です。

COBOL は、ストリングおよび数値を扱うためのさまざまな機能を提供します。これらの機能がコーディングの単純化に役立ちます。

言語環境プログラム日時呼び出し可能サービスは、日付をフルワード・バイナリー整数として保管し、タイム・スタンプを長精度 (64 ビット) 浮動小数点値として保管します。これらの形式を使用することにより、算術計算を日時の値に基づいて単純かつ効率的に行うことができます。このような計算を実行するために、言語ライブラリーの外側でサービスを使用する特別なサブルーチンを書く必要はありません。

### 関連タスク

- 64 ページの『数字組み込み関数の使用』
- 66 ページの『数学用の呼び出し可能サービスの使用』
- 67 ページの『データ呼び出し可能サービスの使用』
- 『反復コーディングの除去』
- 130 ページの『データ項目の変換 (組み込み関数)』
- 134 ページの『データ項目の評価 (組み込み関数)』
- 797 ページの『言語環境プログラム呼び出し可能サービスの使用』

### 関連参照

- 801 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』

---

## 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。

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 UNIX**

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。

以下に例を示します。

```
COPY MEMBER1 OF COPYLIB
```

この修飾句を省略した場合、デフォルトは SYSLIB です。

**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) 扱うには、COPY ステートメントの先頭行に D を指定します。COPY ステートメント自体をデバッグ行にすることはできません。これに D が入っていても、WITH DEBUGGING モードが指定されていなければ、COPY ステートメントが処理されることはありません。

『例: COPY ステートメントの使用』

#### 関連参照

445 ページの『第 18 章 コンパイラ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 CFILEA
 COPY CFILEA.
```

このライブラリー記入項目はプログラムにコピーされ、その結果生じるプログラム・リストは次のようになります。

```
FD FILEA
 COPY CFILEA.
C BLOCK CONTAINS 20 RECORDS
C RECORD CONTAINS 120 CHARACTERS
C LABEL RECORDS ARE STANDARD
C DATA RECORD IS FILE-OUT.
C 01 FILE-OUT PIC X(120).
```

コンパイラー・ソース・リストで COPY ステートメントは別個の行に印刷され、コピーされた行の前には C が付けられます。

テキスト名 DOWORK を持つコピーブックが、以下のステートメントによって保管されているとします。

```
COMPUTE QTY-ON-HAND = TOTAL-USED-NUMBER-ON-HAND
MOVE QTY-ON-HAND to PRINT-AREA
```

DOWORK として識別されたコピーブックを取り出すには、次のようにコーディングします。

```
paragraph-name.
COPY DOWORK.
```

DOWORK プロシー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

795 ページの『反復コーディングの除去』

#### 関連参照

445 ページの『第 18 章 コンパイラー指示ステートメント』

---

## 言語環境プログラム呼び出し可能サービスの使用

言語環境プログラム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。

言語環境プログラムは、以下のタスクに役立てることができます。

- 条件の処理

言語環境プログラム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、それを言語環境プログラムに処理させるか、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。

言語環境プログラムの条件処理のサポートでは、COBOL はプロシーチャー・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。言語環境プログラム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。言語環境プログラムは、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。

- デバッグ・ツールの始動および言語環境プログラム定様式ダンプの取得

デバッグ・ツールは 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。デバッグ・ツールにより、ホストから、または IBM Developer for z Systems のデバッグ・パースペクティブを併用して Windows ベースのワークステーションから、COBOL アプリケーションをデバッグすることができます。

選択したオプションに応じて、言語環境プログラム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。すべての言語環境プログラム・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。

800 ページの『例: 言語環境プログラムの呼び出し可能サービス』

#### 関連概念

799 ページの『言語環境プログラムの呼び出し可能サービスのサンプル・リスト』

## 関連タスク

- 64 ページの『数字組み込み関数の使用』
- 66 ページの『数学用の呼び出し可能サービスの使用』
- 67 ページの『データ呼び出し可能サービスの使用』
- 800 ページの『言語環境プログラム・サービスの呼び出し』
- 568 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』

## 言語環境プログラムの呼び出し可能サービスのサンプル・リスト

次の表は、言語環境プログラム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 91. 言語環境プログラム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
日付および時刻	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	言語環境プログラムの世紀ウィンドウ (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
その他	CEE3DMP	言語環境プログラム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

言語環境プログラム プログラミング・リファレンス

## 言語環境プログラム・サービスの呼び出し

言語環境プログラム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。

```
77 argument comp-1.
77 feedback-code pic x(12) display.
77 result comp-1.
...
CALL "CEESSSQT" using argument, feedback-code, result
```

上の例では、言語環境プログラム・サービス CEESSSQT は、変数 `argument` の平方根の値を計算し、この値を変数 `result` に入れて戻します。

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。フィードバック・コードではなく `OMITTED` を指定したときに、サービスが失敗すると、言語環境プログラム条件が自動的に 言語環境プログラム 条件マネージャーに通知されます。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 言語環境プログラム 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。

言語環境プログラム 呼び出し可能サービスを呼び出し、`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。

『例: 言語環境プログラムの呼び出し可能サービス』

### 関連概念

言語環境プログラム プログラミング・ガイド (一般呼び出し可能サービス)

### 関連参照

言語環境プログラム プログラミング・リファレンス (一般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: 言語環境プログラムの呼び出し可能サービス

この例で示している COBOL プログラムは、言語環境プログラム・サービスの `CEEDAYS` および `CEEDATE` を使用して、`COBOL ACCEPT` ステートメントの結果として生じる日付をフォーマットし、表示します。

`CEEDAYS` および `CEEDATE` を使用すると、言語環境プログラム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: *
* WWWWWWWW, MMMMMMMM 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 " WWWWWWZ, MMMMMMMM 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 92. 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

表 92. 形式 1 と形式 2 の SORT ステートメントの比較 (続き)

特性	形式 1 SORT ステートメント	形式 2 SORT ステートメント
ソートのキーをテーブル定義の一部として指定できる (SEARCH ALL ステートメントでも使用可能)	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ーを使用します。	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ー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*)



---

## 第 9 部 付録



## 付録 A. 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。

### 807 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 バージョン 2 リリース 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

- 54 ページの『数値データの形式』
- 70 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

- 807 ページの『固定小数点データと中間結果』
- 813 ページの『浮動小数点データと中間結果』
- 814 ページの『非算術ステートメントの算術式』
- 354 ページの『ARITH』

---

## 中間結果用の用語

中間結果に関するこの情報を理解するには、以下の用語を理解する必要があります。

- i*** 中間結果用に保持された整数の桁数。(ROUNDED 句を使用している場合は、正確さを得るために必要なら、整数がもう 1 桁余分に保持します。)
- d*** 中間結果用に保持された小数部の桁数。(ROUNDED 句を使用している場合は、正確さを得るために必要なら、小数部がもう 1 桁余分に保持します。)

### ***dmax***

特定のステートメントで、次の項目のうち最も大きいもの。

- 最終結果フィールド (1 つ以上) に必要な小数部の桁数。
- 除数または指数を除き、オペランドに対して定義された小数部の最大桁数。
- 関数オペランドの *outer-dmax*。

### ***inner-dmax***

関数に関して、以下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

63 ページの『算術式の使用』

関連参照

806 ページの『中間結果用の用語』

---

## 固定小数点データと中間結果

コンパイラーは、中間結果における整数および小数点以下の桁数を、決定します。

### 加算、減算、乗算、および除算

次の表は、加算、減算、乗算、または除算の結果として理論上可能な精度を示しています。

演算	整数桁	小数桁
+ または -	( <i>i1</i> または <i>i2</i> ) + 1、どちらか大きい方	<i>d1</i> または <i>d2</i> 、どちらか大きい方
*	<i>i1</i> + <i>i2</i>	<i>d1</i> + <i>d2</i>
/	<i>i2</i> + <i>d1</i>	( <i>d2</i> - <i>d1</i> ) または <i>dmax</i> 、どちらか大きい方

算術ステートメントのオペランドを十分な小数桁で定義して、最終結果の正確度が希望どおりになるようにしなければなりません。

次の表は、互換モード (つまり、デフォルトのコンパイラ・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 >30	$i$ 整数桁数および $30-i$ 小数桁数 $30-dmax$ 整数桁数および $dmax$ 小数桁数

次の表は、拡張モード (つまり、コンパイラ・オプション ARITH(EXTEND) が有効である場合) で加算、減算、乗算、または除算を伴う算術演算の固定小数点中間結果においてコンパイラが保持する桁数を示しています。

$i + d$ の値	$d$ の値	$i + dmax$ の値	$ir$ 用に保持される桁数
<31 または =31	任意の値	任意の値	$i$ 整数桁数および $d$ 小数桁数
>31	< $dmax$ または = $dmax$	任意の値	$31-d$ 整数桁数および $d$ 小数桁数
	> $dmax$	<31 または =31 >31	$i$ 整数桁数および $31-i$ 小数桁数 $31-dmax$ 整数桁数および $dmax$ 小数桁数

## 指数

指数は、式  $op1 ** op2$  で表されます。  $op2$  の特性に基づいて、コンパイラは固定小数点数のべき乗計算を次の 3 つのいずれかの方法で処理します。

- $op2$  が小数部で表されている場合は、浮動小数点命令が使用されます。
- $op2$  が整数リテラルまたは定数である場合、値  $d$  は次のように計算されます。

$$d = d1 * |op2|$$

また、値  $i$  は、 $op1$  の特性に基づいて、次のように計算されます。

- $op1$  がデータ名または変数である場合は、次のように計算されます。  
 $i = i1 * |op2|$
- $op1$  がリテラルまたは定数である場合、 $i$  は、 $op1 ** |op2|$  の値の整数の桁数に等しく設定されます。

互換モード (ARITH(COMPAT) を使用してコンパイルする場合) では、コンパイラは  $i$  および  $d$  を計算し終わると、次の表に示すアクションを取り、べき乗計算の中間結果  $ir$  を処理します。

$i + d$ の値	その他の条件	取られるアクション
<30	任意	$ir$ において、 $i$ の整数桁数および $d$ の小数桁数が保持される。

$i + d$ の値	その他の条件	取られるアクション
=30	$op1$ が奇数の桁数を持つ。	$ir$ において、 $i$ の整数桁数および $d$ の小数桁数が保持される。
	$op1$ が偶数の桁数を持つ。	$op2$ が整数データ名または変数である場合と同じアクション (以下を参照)。例外: リテラル 1 の累乗に計算される 30 桁の整数の場合は、 $ir$ において、 $i$ の整数桁数および $d$ の小数桁数が保持される。
>30	任意	$op2$ が整数データ名または変数である場合と同じアクション (以下を参照)

拡張モード (ARITH(EXTEND) を使用してコンパイルする場合) では、コンパイラは  $i$  および  $d$  を計算し終わると、次の表に示すアクションを取り、指数計算の中間結果  $ir$  を処理します。

$i + d$ の値	その他の条件	取られるアクション
<31	任意	$ir$ において、 $i$ の整数桁数および $d$ の小数桁数が保持される。
=31 または >31	任意	$op2$ が整数データ名または変数である場合と同じアクション (以下を参照)。例外: リテラル 1 の累乗に計算される 31 桁の整数の場合は、 $ir$ において、 $i$ の整数桁数および $d$ の小数桁数が保持される。

$op2$  が負である場合は、1 という値が予備計算によって作成された結果で除算されます。使用される  $i$  と  $d$  の値は、上記に示された固定小数点データの除算規則に従って計算されます。

- $op2$  が整数データ名または変数である場合は、 $dmax$  の小数桁数と、 $30-dmax$  (互換モード) または  $31-dmax$  (拡張モード) 整数桁数が使用されます。 $op1$  はそれ自体によって  $(|op2| - 1)$  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。0 による除算とべき乗計算の SIZE ERROR 条件が適用されます。

9 桁を超える有効数字を持つ固定小数点の指数部は、常に 9 桁に切り捨てられます。指数がリテラルまたは定数である場合、E レベルのコンパイラ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。

810 ページの『例: 固定小数点の算術での指数』

#### 関連参照

806 ページの『中間結果用の用語』

810 ページの『中間結果での切り捨て』

811 ページの『バイナリー・データと中間結果』

813 ページの『浮動小数点データと中間結果』

811 ページの『固定小数点算術で評価される組み込み関数』

354 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、コンパイラーが、必要に応じて中間結果を保管しながら、ゼロ以外の整数累乗へのべき乗計算を一連の乗算として実行する方法を示しています。

```
COMPUTE Y = A ** B
```

B が 4 であれば、結果は次のように計算されます。使用される *i* と *d* の値は、固定小数点データおよび中間結果に関する乗算規則に従って計算されます (以下を参照してください)。

1. A に A を掛けると、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 は正であるので、*ir4* は Y に移動されます。しかし B が -4 であった場合は、さらに 5 番目のステップが実行されます。

5. 1 を *ir4* で割ると *ir5* が得られる。

*ir5* は *dmax* の小数桁数を持っており、Y に移動されます。

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

806 ページの『中間結果用の用語』

807 ページの『固定小数点データと中間結果』

## 中間結果での切り捨て

中間結果において桁数が互換モードの 30、または拡張モードの 31 を超えるときは常に、コンパイラーは、30 (互換モード) または 31 (拡張モード) 桁までに切り捨て、警告を表示します。切り捨てが実行時に起こった場合は、メッセージが出され、プログラムは実行を続けます。

固定小数点計算で発生する可能性のある中間結果の切り捨てを回避したい場合には、代わりに浮動小数点オペランド (COMP-1 または COMP-2) を使用してください。

### 関連概念

54 ページの『数値データの形式』

### 関連参照

807 ページの『固定小数点データと中間結果』

354 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

807 ページの『固定小数点データと中間結果』

354 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。

機能	<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	n/a	9
MOD	0	$\min(i1\ i2)$
ORD	n/a	3
ORD-MAX		9
ORD-MIN		9
YEAR-TO-YYYY	0	4
INTEGER		固定小数点引数の場合: 引数より 1 桁大きくなります。浮動小数点引数の場合: 30 (互換モード)、31 (拡張モード)
INTEGER-PART		固定小数点引数の場合: 引数と同じ桁数になります。浮動小数点引数の場合: 30 (互換モード)、31 (拡張モード)

## 混合関数

混合 組み込み関数とは、結果の型がその引数の型に依存する関数です。 引数がすべて数値で、どの引数も浮動小数点でない場合、その混合関数は固定小数点です。(混合関数のいずれかの引数が浮動小数点である場合、その関数は浮動小数点命令によって評価され、浮動小数点結果を戻します。) 混合関数が固定小数点算術演算で評価されたときは、引数がすべて整数であれば結果は整数になり、それ以外の場合は結果は固定小数点になります。

混合関数 MAX、MIN、RANGE、REM、および SUM の場合、*outer-dmax* は常に *inner-dmax* に等しくなります (したがって、引数がすべて整数であれば、両方とも 0 になります)。 これらの関数について戻される結果の精度を判別するためには、固定小数点算術演算および中間結果に関する規則 (以下を参照) を、アルゴリズムの各ステップに適用してください。

### MAX

1. 最初の引数を関数結果に割り当てる。
2. 残りの引数ごとに、以下のステップを実行する。
  - a. 関数結果の代数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、以下のステップを実行する。
  - 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、以下のステップを実行する。
  - a. その引数を関数結果に足す。
  - b. その和を関数結果に割り当てる。

### 関連参照

806 ページの『中間結果用の用語』

---

## 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  $dmax$  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。

長精度浮動小数点算術演算が算術式の 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算術演算が算術式の 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。

負数を小数でべき乗した値は、COBOL で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評価され、結果が未定義である可能性がある場合、指数は実行時に評価され、整数値を持つかどうかは判別されます。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

806 ページの『中間結果用の用語』

354 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。

以下に例を示します。

```
IF operand-1 = expression-1 THEN . . .
```

*operand-1* が COMP-2 と定義されるデータ名である場合は、*expression-1* には、たとえ固定小数点オペランドしか含まれていなくても、浮動小数点算術演算の規則が適用されます。というのは、これは固定小数点オペランドと比較されるためです。

- ある算術式と別のデータ項目または算術式との間の比較で、関係演算子が使用されない場合 (すなわち、明示的な比較条件がない場合) は、その算術式は、被比較数の属性を考慮に入れずに評価されます。以下に例を示します。

```
EVALUATE expression-1
 WHEN expression-2 THRU expression-3
 WHEN expression-4
 . . .
END-EVALUATE
```

上記のステートメントでは、それぞれの算術式は、その特性に応じて、固定小数点または浮動小数点算術で評価されます。

#### 関連概念

70 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

806 ページの『中間結果用の用語』

807 ページの『固定小数点データと中間結果』

813 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)



---

## 付録 B. 2 バイト文字セット (DBCS) データの変換

言語環境プログラムのサービス・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。

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

156 ページの『国別 (Unicode) 表現と間の変換』

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

『DBCS の表記』

『英数字から DBCS データへの変換 (IGZCA2D)』

820 ページの『DBCS から英数字データへの変換 (IGZCD2A)』

359 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)

言語環境プログラムの 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

- *parameter-1*、*parameter-3*、および *parameter-4* はルーチン BY REFERENCE または BY CONTENT に渡すことができますが、*parameter-2* は BY REFERENCE に渡さなければなりません。
- 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。

819 ページの『例: IGZCA2D』

### 関連参照

819 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 93. IGZCA2D 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 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。
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

817 ページの『DBCS の表記』

---

## DBCS から英数字データへの変換 (IGZCD2A)

言語環境プログラムの 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

- *parameter-1*、*parameter-3*、および *parameter-4* はルーチン BY REFERENCE または BY CONTENT に渡すことができますが、*parameter-2* は BY REFERENCE に渡さなければなりません。
- 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。

『例: IGZCD2A』

関連参照

『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 94. 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、右側に単一バイト・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、右側で切り捨てられました。 <sup>1</sup>
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

817 ページの『DBCS の表記』

---

## 付録 C. 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

『XMLPARSE(XMLSS) が有効な場合の XML PARSE 例外』

825 ページの『XMLPARSE(COMPAT) が有効な場合の XML PARSE 例外』

832 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なります。

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 System Services User's Guide and Reference」(下記参照) および「824 ページの表 95」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。

処理プロシージャーが例外イベント後にパーサーに戻ると、制御は、ON EXCEPTION 句で指定したステートメント、または XML PARSE ステートメントの最後 (ON EXCEPTION 句をコーディングしていない場合) に渡ります。

### 検証の例外:

VALIDATING 句を含む XML PARSE ステートメントをコーディングしていて、文書が妥当でないと z/OS XML System Services パーサーが判断すると、パーサーは戻りコード 24 (16 進数の 18、XRC\_NOT\_VALID) を生成します。

### Enterprise COBOL に固有の例外

一部の例外は、Enterprise COBOLに固有なものであるため、z/OS XML System Services User's Guide and Reference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)。16 進数で 800 から 899 までの範囲にある理由コードを持つ例外の戻りコードは 4 (16 進数 0004、XRC\_WARNING) です。他の例外の戻りコードは 16 (16 進数 0010、XRC\_FATAL) です。例外コード (特殊レジスター XML-CODE の値) は、この戻りコードが、次の表にあるいずれかの理由コードと連結して形成されます。

表 95. Enterprise COBOL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ーに対するメモリー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

635 ページの『XML-CODE』

634 ページの『XML イベント』

#### 関連タスク

654 ページの『XML PARSE の例外処理』

関連参照  
436 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、Enterprise COBOL ライブラリーで提供されている XML パーサーは、特殊レジスター XML-CODE に、例外を識別する値を設定します。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  
『継続を許可する XML PARSE 例外』  
829 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

XMLPARSE(COMPAT) コンパイラー・オプションが有効である場合、例外イベントの後に XML パーサーが処理を続行できるかどうかは、例外コードの値によって決まります。

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。

- 実際の文書エンコード
- 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 96. 続行可能な 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。

表 96. 続行可能な 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
2	パーサーで、エレメント内容に含まれない、処理命令、エレメント、コメント、または文書タイプ宣言の無効な開始が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
3	パーサーで、重複する属性名が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
4	パーサーで、属性値にマークアップ文字 '<' が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
5	エレメントの開始および終了タグ名が一致しません。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
6	パーサーで、エレメント内容に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
7	パーサーで、エレメント内容に、エレメント、コメント、処理命令、または CDATA セクションの無効な開始が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
8	パーサーで、エレメント内容に、一致する開始文字シーケンス '<![CDATA[' のない、CDATA 終了文字シーケンス ']]>' 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 96. 続行可能な 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
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。
13	パーサーで、16 進文字参照 (形式 &#xddd; の) 内に無効な数字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
14	パーサーで、10 進数文字参照 (形式 &#ddd; の) 内に無効な数字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
15	XML 宣言内のエンコード宣言値が小文字または大文字の A から Z で始まっていませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。

表 96. 続行可能な 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
16	文字参照が適切な XML 文字を参照しませんでした。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
17	パーサーで、エンティティ参照名に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
18	パーサーで、属性値に無効文字が見つかりました。	パーサーは、文書の終わりに到達するまで、あるいは継続不能の原因となるエラーを検出するまで、エラーの検出を継続します。パーサーでは、END-OF-DOCUMENT イベントを除き、追加の標準イベントをシグナル通知しません。
70	実際の文書エンコードは EBCDIC であり、CODEPAGE コンパイラー・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いましたが、文書エンコード宣言ではサポートされる EBCDIC コード・ページが指定されていませんでした。	パーサーは、CODEPAGE コンパイラー・オプションで指定されたエンコードを使用します。
71	実際の文書エンコードは EBCDIC であり、文書エンコード宣言ではサポートされる EBCDIC エンコードが指定されていましたが、CODEPAGE コンパイラー・オプションはサポートされる 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。
72	実際の文書エンコードは EBCDIC ですが、CODEPAGE コンパイラー・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。	パーサーは、EBCDIC コード・ページ 1140 (USA、カナダ、... ユーロ国別拡張コード・ページ) を使用します。

表 96. 続行可能な 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
73	実際の文書エンコードは EBCDIC ですが、CODEPAGE コンパイラー・オプションおよび文書エンコード宣言のどちらにもサポートされる EBCDIC コード・ページが指定されていませんでした。	パーサーは、EBCDIC コード・ページ 1140 (USA、カナダ、... ユーロ国別拡張コード・ページ) を使用します。
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。	パーサーはコード・ページ 1200 (Unicode UTF-16) を使用します。
100,001 から 165,535	CODEPAGE コンパイラー・オプションおよび文書エンコード宣言に指定された、サポートされる EBCDIC コード・ページがそれぞれ異なっていました。 XML-CODE には、エンコード宣言に 100,000 をプラスするためのコード・ページ CCSID が含まれています。	EXCEPTION イベントから戻る前に、XML-CODE をゼロに設定した場合、パーサーでは、CODEPAGE コンパイラー・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

635 ページの『XML-CODE』

649 ページの『XML 入力文書エンコード』

#### 関連タスク

654 ページの『XML PARSE の例外処理』

#### 関連参照

436 ページの『XMLPARSE』 (コンパイラー・オプション)

## 継続を許可しない XML PARSE 例外

XMLPARSE (COMPAT) コンパイラー・オプションが有効である場合、以下に記述した例外が発生すると、XML パーサーは処理を続行できません。

処理プロシージャー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 97. 継続を許可しない XML PARSE 例外(XMLPARSE(COMPAT) の場合)

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。

表 97. 継続を許可しない XML PARSE 例外(XMLPARSE(COMPAT) の場合) (続き)

例外コード (10進数)	説明
125	エレメントの先頭の属性名の先頭文字が、文字、 '_'、または ':' ではありませんでした。
126	パーサーで、エレメント名内に、またはエレメント名の後のいずれかに無効文字が見つかりました。
127	パーサーで、属性名の後に '=' 以外の文字が見つかりました。
128	パーサーで、無効な属性値区切り文字が見つかりました。
130	属性名の先頭文字が、文字、 '_'、または ':' ではありませんでした。
131	パーサーで、属性名内に、または属性名の後のいずれかに無効文字が見つかりました。
132	空のエレメント・タグが、 '/' の後に続く '>' で終了しませんでした。
133	エレメント終了タグ名の先頭文字が、文字、 '_'、または ':' ではありませんでした。
134	エレメント終了タグ名が '>' で終了しませんでした。
135	エレメント名の先頭文字が、文字、 '_'、または ':' ではありませんでした。
136	パーサーで、エレメント内容に、コメントまたは CDATA セクションの無効な開始が見つかりました。
137	パーサーで、コメントの無効な開始が見つかりました。
138	処理命令ターゲット名の先頭文字が、文字、 '_'、または ':' ではありませんでした。
139	パーサーで、処理命令ターゲット名内に、または処理命令ターゲット名の後のいずれかに無効文字が見つかりました。
140	処理命令が終了文字シーケンス '?>' で終了しませんでした。
141	パーサーで、文字参照またはエンティティー参照内の '&' の後に無効文字が見つかりました。
142	バージョン情報が XML 宣言にありませんでした。
143	XML 宣言内の 'version' の後に '=' がありませんでした。
144	XML 宣言内のバージョン宣言値が欠落しているか、または不適切に区切られています。
145	XML 宣言内のバージョン情報値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
146	パーサーで、XML 宣言内のバージョン情報値の終了区切り文字の後に無効文字が見つかりました。
147	パーサーで、XML 宣言にオプションのエンコード宣言ではない、無効な属性が見つかりました。
148	XML 宣言内の 'encoding' の後に '=' がありませんでした。
149	XML 宣言内のエンコード宣言値が欠落しているか、または不適切に区切られています。
150	XML 宣言内のエンコード宣言値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
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。

表 97. 継続を許可しない XML PARSE 例外(XMLPARSE(COMPAT) の場合) (続き)

例外コード (10進数)	説明
152	パーサーで、XML 宣言にオプションの standalone 宣言ではない、無効な属性が見つかりました。
153	XML 宣言内の standalone の後に = がありませんでした。
154	XML 宣言内の standalone 宣言値が欠落しているか、または不適切に区切られています。
155	standalone 宣言値が 'yes' または 'no' 以外の値になっていました。
156	XML 宣言内の standalone 宣言値が不適切な文字を指定したか、または開始と終了の区切り文字が一致しませんでした。
157	パーサーで、XML 宣言内の standalone 宣言値の終了区切り文字の後に無効文字が見つかりました。
158	XML 宣言が正しい文字シーケンス '?>' で終了しなかったか、無効属性が含まれていました。
159	パーサーで、ルート・エレメントの末尾の後に文書タイプ宣言の開始が見つかりました。
160	パーサーで、ルート・エレメントの末尾の後にエレメントの開始が見つかりました。
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

635 ページの『XML-CODE』

#### 関連タスク

654 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。このような例外が発生すると、ON EXCEPTION 句で指定されたステートメント、または、ON EXCEPTION 句をコーディングしていない場合には、XML GENERATE ステートメントの末尾に制御が渡されます。

表 98. 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lementまたは属性名では無効な文字を含んでいました。
402	Unicode への変換時に、DBCS データ名の先頭文字は、XML Elementまたは属性名の先頭文字としては無効なものでした。
403	OCCURS DEPENDING ON 変数の値が 16,777,215 を超えました。
410	CODEPAGE コンパイラ・オプションで指定された CCSID ページは、Unicode への変換に関してはサポートされません。
411	CODEPAGE コンパイラ・オプションで指定された CCSIDは、サポート対象の 1 バイト EBCDIC CCSIDではありません。
414	XML 文書に指定された CCSID が無効だったか、またはサポートされていませんでした。
415	受け取り側は国別でしたが、文書に対して指定されたエンコードは UTF-16 ではありませんでした。
416	XML 名前空間 ID に無効な XML 文字が含まれていました。
417	<p>Element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。文書内の「hex」が接頭部のElement・タグ名または属性名および元のデータ値を 16 進表記して、XML の生成を続行しました。</p> <p>TYPE IS CONTENT 指定はすべて無視され、項目はElementとして扱われます。</p>
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	受け取り側は英数字で、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、文書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

682 ページの『XML GENERATE 例外の処理』





## 付録 D. 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

『JSON GENERATE 例外』  
『JSON PARSE 状態、関連コード、およびランタイム・メッセージ』  
836 ページの『非例外状態とそれに対応する JSON-STATUS 値』  
836 ページの『例外状態とそれに対応する JSON-CODE 値』  
837 ページの『非例外状態ランタイム・メッセージ』  
838 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 99. 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トが格納されています。
500 - 599	内部エラー。このエラーをサービス担当者に報告してください。

## 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 100. 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	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。
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 101. 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。

表 101. JSON 例外状態の理由コード (続き)

JSON-CODE レジスターでの理由コード	説明
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	適合する 1 つ以上の JSON 名前/値ペアに値 true または false が含まれていました。
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。

## 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 名前 *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 コンパイラー・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、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。

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。



---

## 付録 E. 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。各出口モジュールの処理、出口モジュールのエラー処理、または CICS、SQL および SQLIMS ステートメントでの EXIT オプションの使用方法について詳しくは、以下のトピックを参照してください。

### 関連タスク

『ユーザー出口作業域の使用』

842 ページの『出口モジュールからの呼び出し』

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

### 関連参照

374 ページの『EXIT』

842 ページの『INEXIT の処理』

844 ページの『LIBEXIT の処理』

847 ページの『PRTEXTIT の処理』

848 ページの『ADEXIT の処理』

850 ページの『MSGEXIT の処理』

861 ページの『出口モジュールでのエラー処理』

---

## 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。

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 102. 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	PRTEXTIT
4	ADEXIT
5	(予約済み)
6	MSGEXIT

関連参照

- 『INEXIT の処理』
- 844 ページの『LIBEXIT の処理』
- 847 ページの『PRTEXTIT の処理』
- 848 ページの『ADEXIT の処理』
- 850 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。

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

出口モジュールでは、RMODE 属性が 24、AMODE 属性が ANY でなければなりません。

関連概念

- 43 ページの『ストレージとそのアドレス可能性』

INEXIT の処理

SYSIN の代わりにユーザー提供プログラム・オブジェクトからソース・コードを読み取る場合に、INEXIT 出口モジュールを使用します。

表 103. 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 104. 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= データの終わり • 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)
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

841 ページの『ユーザー出口作業域の使用』

842 ページの『出口モジュールからの呼び出し』

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

## 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 105. 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やアドレス、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せん。

*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 106. 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 ( <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。

表 106. ネストなしの **COPY** ステートメントによる **LIBEXIT** の処理 (続き)

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
<i>library-name</i> が正常にオープンされた場合に、 <b>FIND</b> 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。
<b>GET</b> 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やアドレス、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、 <b>CLOSE</b> 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 107. 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
ネストされた <b>COPY</b> ステートメントからの要求された <i>library-name</i> が前にオープンされていないならば、 <b>OPEN</b> 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 ( <b>OPEN</b> ) 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 <b>FIND</b> 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 ( <b>FIND</b> ) 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。
<b>GET</b> 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やアドレス、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 108. 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。  <ul style="list-style-type: none"> <li>• 0=OPEN</li> <li>• 1=CLOSE</li> <li>• 2=GET</li> <li>• 4=FIN</li> </ul>
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。  <ul style="list-style-type: none"> <li>• 0= 操作は成功しました</li> <li>• 4= データの終わり</li> <li>• 12= 操作は失敗しました</li> </ul>
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  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ー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、ブランクが埋め込まれます。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、ブランクが埋め込まれます。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。必要な場合にはブランクが埋め込まれ、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。

表 108. LIBEXIT 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
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。必要な場合にはブランクが埋め込まれ、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

841 ページの『ユーザー出口作業域の使用』

842 ページの『出口モジュールからの呼び出し』

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

## PRTEXT の処理

SYSPRINT データ・セットの代わりに PRTEXT 出口モジュールを使用します。

表 109. PRTEXT 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>mod3</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。
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

## PRTEXT 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 110. PRTEXT 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=PRTEXT

表 110. PRTEXTIT 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
2	命令コード	操作のタイプを示すハーフワード。 <ul style="list-style-type: none"> <li>• 0=OPEN</li> <li>• 1=CLOSE</li> <li>• 3=PUT</li> </ul>
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。 <ul style="list-style-type: none"> <li>• 0= 操作は成功しました</li> <li>• 12= 操作は失敗しました</li> </ul>
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  3 番目のワード: PRTEXTIT 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)
6	データ・バッファード または <i>str3</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・バッファード。  <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

841 ページの『ユーザー出口作業域の使用』

842 ページの『出口モジュールからの呼び出し』

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

## 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 111. 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 ( <i>op</i> 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 112. 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。 <ul style="list-style-type: none"> <li>0=OPEN</li> <li>1=CLOSE</li> <li>3=PUT</li> </ul>
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。 <ul style="list-style-type: none"> <li>0= 操作は成功しました</li> <li>12= 操作は失敗しました</li> </ul>
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

表 112. ADEXIT 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 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

841 ページの『ユーザー出口作業域の使用』

842 ページの『出口モジュールからの呼び出し』

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

#### 関連参照

350 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表示されます。

表 113. 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 ( <i>mod5</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	オプションで <i>str5</i> を処理し、OPEN 要求の状況をコンパイラーに渡します



表 113. MSGEXIT の処理 (続き)

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
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。 <ul style="list-style-type: none"> <li>• (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。</li> <li>• 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</li> <li>• (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</li> </ul>
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 114. 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。 <ul style="list-style-type: none"> <li>• 0=OPEN</li> <li>• 1=CLOSE</li> <li>• 5=MSGSEV: メッセージ重大度をカスタマイズします</li> </ul>
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: <ul style="list-style-type: none"> <li>• 0= メッセージはカスタマイズされませんでした</li> <li>• 4= メッセージが検出され、カスタマイズされました</li> <li>• 12= 操作は失敗しました</li> </ul>
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  6 番目のワード: MSGEXIT が使用
5	未使用	(他の出口が使用)

表 114. 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
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。 <ul style="list-style-type: none"> <li>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</li> <li>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</li> <li>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)</li> </ul>
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)

855 ページの『例: MSGEXIT ユーザー出口』

#### 関連タスク

841 ページの『ユーザー出口作業域の使用』

842 ページの『出口モジュールからの呼び出し』

『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

## 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。

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。
  - 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

317 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

318 ページの『コンパイラー診断メッセージの重大度コード』

『カスタマイズ可能なコンパイラー・メッセージ重大度』

855 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』

861 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ー、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 つのカテゴリーを示します。FIPS メッセージのカテゴリーは、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 115. 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
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すが、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

318 ページの『コンパイラ診断メッセージの重大度コード』

379 ページの『FLAGSTD』

855 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

852 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

318 ページの『コンパイラー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。

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 2 モディフィケーション 0 *
*
* LICENSED MATERIALS - PROPERTY OF IBM. *
*
* 5655-EC6 COPYRIGHT IBM CORP. 2017 *
* 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
* compiles 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, PRTEXTIT 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 NOFASTSORT 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 V3R4 and later compilers such as
* Enterprise COBOL Version 4 Release 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 '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.

 When(7300) When(7301) When(7302) When(7304)
 When(7307) When(7309)
 Compute EXIT-USER-SEV = -1 *> Suppress the messages

```

```

* 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(0) 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。

ユーザー出口 *exit-name* をロードしようとしたときにエラーが発生しました。

出口オープン失敗:

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。

ユーザー出口 *exit-name* をオープンしようとしたときにエラーが発生しました。

**PRTEXIT PUT** の失敗:

- メッセージ IGYSI5203-U がリストに書き込まれます。

PRTEXIT ユーザー出口への PUT 要求が失敗し、戻りコード *nn* が返されました。  
(A PUT request to the PRTEXIT 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。

PRTEXIT ユーザー出口 *exit-name* でエラーが発生しました。コンパイラーは終了しました。

**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:

レコード・アドレスが *exit-name* ユーザー出口によって設定されていません。

- IGYSI5205-U:

INEXIT ユーザー出口からの GET 要求が失敗し、戻りコード *nn* が戻されました。

- IGYSI5206-U:

レコード長が *exit-name* ユーザー出口によって設定されていません。

#### 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。

ADEXIT ユーザー出口 *exit-name* でエラーが発生しました。コンパイラーは終了しました。

- メッセージ IGYSI5226-U がリストに書き込まれます。

ADEXIT ユーザー出口への PUT 要求が失敗し、戻りコード *nn* が戻されました。

#### 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 ユーザー出口 *exit-name* で、サポートされていないメッセージ重大度のカスタマイズが指定されました。  
メッセージ番号、デフォルトの重大度、およびユーザーが指定した重大度は、*mm*、*ds*、*us* です。  
MSGEXIT ユーザー出口 *exit-name* を変更してこのエラーを訂正してください。  
(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。

MSGEXIT ユーザー出口ルーチン *exit-name* の呼び出しが失敗し、戻りコード *nn* が戻されました。  
(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

852 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

EXIT コンパイラー・オプションのサブオプションを使用してコンパイルし、EXEC CICS、EXEC SQL、または EXEC SQLIMS の各ステートメントがプログラムに含まれる場合、出口モジュールで実行できるアクションは、個別の CICS 変換プログラムおよび Db2 プリコンパイラーを使用するか、組み込み CICS 変換プログラムおよび Db2 コプロセッサを使用するかによって異なります。

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 116. 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ル	組み込みの、または分離型の CICS および Db2 変換プログラムを使用した変換	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。
LIBEXIT	組み込み	EXEC SQL INCLUDE および EXEC SQLIMS INCLUDE ステートメントによってもたらされたステートメントを LIBEXIT モジュールで処理できます。LIBEXIT モジュールで EXEC CICS ソース・ステートメントを処理できます。	EXEC SQL INCLUDE および EXEC SQLIMS INCLUDE ステートメントは、COBOL COPY ステートメントと同じように処理されます。
	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。
PRTEXIT	組み込み	PRTEXIT 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	PRTEXIT 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	PRTEXIT 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 プリコンパイラー・リストに表示されています。

#### 関連概念

508 ページの『組み込みの CICS 変換プログラム』

515 ページの『Db2 コプロセッサ』

#### 関連タスク

506 ページの『CICS オプションを使用したコンパイル』

520 ページの『SQL オプションを使用したコンパイル』

#### 関連参照

842 ページの『INEXIT の処理』

844 ページの『LIBEXIT の処理』

847 ページの『PRTEXT の処理』

848 ページの『ADEXIT の処理』

850 ページの『MSGEXIT の処理』

---

## 付録 F. JNI.cpy コピーブック

このリストは JNI.cpy コピーブックを示しています。これを使用すると、COBOL プログラムから Java Native Interface (JNI) サービスにアクセスすることができます。

JNI.cpy には、Java JNI タイプに対応するサンプル COBOL データ定義と、JNI 呼び出し可能サービスにアクセスするための関数ポインターを含む JNI 環境構造である JNINativeInterface が入っています。

JNI.cpy は、COBOL インストール・ディレクトリーの include サブディレクトリー (通常は /usr/lpp/cobol/include) にあります。の z/OS UNIX ファイル・システム にあります。JNI.cpy は、C プログラマーが JNI にアクセスするために使用するヘッダー・ファイル jni.h に類似しています。

```

* COBOL declarations for Java native method interoperation *
* * *
* To use the Java Native Interface callable services from a *
* COBOL program: *
* 1) Use a COPY statement to include this file into the *
* the Linkage Section of the program, e.g. *
* Linkage Section. *
* Copy JNI *
* 2) Code the following statements at the beginning of the *
* Procedure Division: *
* Set address of JNIEnv to JNIEnvPtr *
* Set address of JNINativeInterface to JNIEnv *

*
* Sample JNI type definitions in COBOL
*
*01 jboolean1 pic X.
* 88 jboolean1-true value X'01' through X'FF'.
* 88 jboolean1-false value X'00'.
*
*01 jbyte1 pic X.
*
*01 jchar1 pic N usage national.
*
*01 jshort1 pic s9(4) comp-5.
*01 jint1 pic s9(9) comp-5.
*01 jlong1 pic s9(18) comp-5.
*
*01 jfloat1 comp-1.
*01 jdouble1 comp-2.
*
*01 jobject1 object reference.
*01 jclass1 object reference.
*01 jstring1 object reference jstring.
*01 jarray1 object reference jarray.
*
*01 jbooleanArray1 object reference jbooleanArray.
*01 jbyteArray1 object reference jbyteArray.
*01 jcharArray1 object reference jcharArray.
*01 jshortArray1 object reference jshortArray.
*01 jintArray1 object reference jintArray.
*01 jlongArray1 object reference jlongArray.
*01 floatArray1 object reference floatArray.
```

```
*01 jdoubleArray1 object reference jdoubleArray.
*01 jobjectArray1 object reference jobjectArray.
```

```
* Possible return values for JNI functions.
```

```
01 JNI-RC pic S9(9) comp-5.
* success
 88 JNI-OK value 0.
* unknown error
 88 JNI-ERR value -1.
* thread detached from the VM
 88 JNI-EDETACHED value -2.
* JNI version error
 88 JNI-EVERSION value -3.
* not enough memory
 88 JNI-ENOMEM value -4.
* VM already created
 88 JNI-EEXIST value -5.
* invalid arguments
 88 JNI-EINVAL value -6.
```

```
* Used in ReleaseScalarArrayElements
```

```
01 releaseMode pic s9(9) comp-5.
 88 JNI-COMMIT value 1.
 88 JNI-ABORT value 2.
```

```
01 JNIenv pointer.
```

```
* JNI Native Method Interface - environment structure.
```

```
01 JNINativeInterface.
 02 pointer.
 02 pointer.
 02 pointer.
 02 pointer.
 02 GetVersion function-pointer.
 02 DefineClass function-pointer.
 02 FindClass function-pointer.
 02 FromReflectedMethod function-pointer.
 02 FromReflectedField function-pointer.
 02 ToReflectedMethod function-pointer.
 02 GetSuperclass function-pointer.
 02 IsAssignableFrom function-pointer.
 02 ToReflectedField function-pointer.
 02 Throw function-pointer.
 02 ThrowNew function-pointer.
 02 ExceptionOccurred function-pointer.
 02 ExceptionDescribe function-pointer.
 02 ExceptionClear function-pointer.
 02 FatalError function-pointer.
 02 PushLocalFrame function-pointer.
 02 PopLocalFrame function-pointer.
 02 NewGlobalRef function-pointer.
 02 DeleteGlobalRef function-pointer.
 02 DeleteLocalRef function-pointer.
 02 IsSameObject function-pointer.
 02 NewLocalRef function-pointer.
 02 EnsureLocalCapacity function-pointer.
 02 AllocObject function-pointer.
 02 NewObject function-pointer.
 02 NewObjectV function-pointer.
 02 NewObjectA function-pointer.
 02 GetObjectClass function-pointer.
 02 IsInstanceOf function-pointer.
 02 GetMethodID function-pointer.
 02 CallObjectMethod function-pointer.
 02 CallObjectMethodV function-pointer.
 02 CallObjectMethodA function-pointer.
```



02 CallBooleanMethod	function-pointer.
02 CallBooleanMethodV	function-pointer.
02 CallBooleanMethodA	function-pointer.
02 CallByteMethod	function-pointer.
02 CallByteMethodV	function-pointer.
02 CallByteMethodA	function-pointer.
02 CallCharMethod	function-pointer.
02 CallCharMethodV	function-pointer.
02 CallCharMethodA	function-pointer.
02 CallShortMethod	function-pointer.
02 CallShortMethodV	function-pointer.
02 CallShortMethodA	function-pointer.
02 CallIntMethod	function-pointer.
02 CallIntMethodV	function-pointer.
02 CallIntMethodA	function-pointer.
02 CallLongMethod	function-pointer.
02 CallLongMethodV	function-pointer.
02 CallLongMethodA	function-pointer.
02 CallFloatMethod	function-pointer.
02 CallFloatMethodV	function-pointer.
02 CallFloatMethodA	function-pointer.
02 CallDoubleMethod	function-pointer.
02 CallDoubleMethodV	function-pointer.
02 CallDoubleMethodA	function-pointer.
02 CallVoidMethod	function-pointer.
02 CallVoidMethodV	function-pointer.
02 CallVoidMethodA	function-pointer.
02 CallNonvirtualObjectMethod	function-pointer.
02 CallNonvirtualObjectMethodV	function-pointer.
02 CallNonvirtualObjectMethodA	function-pointer.
02 CallNonvirtualBooleanMethod	function-pointer.
02 CallNonvirtualBooleanMethodV	function-pointer.
02 CallNonvirtualBooleanMethodA	function-pointer.
02 CallNonvirtualByteMethod	function-pointer.
02 CallNonvirtualByteMethodV	function-pointer.
02 CallNonvirtualByteMethodA	function-pointer.
02 CallNonvirtualCharMethod	function-pointer.
02 CallNonvirtualCharMethodV	function-pointer.
02 CallNonvirtualCharMethodA	function-pointer.
02 CallNonvirtualShortMethod	function-pointer.
02 CallNonvirtualShortMethodV	function-pointer.
02 CallNonvirtualShortMethodA	function-pointer.
02 CallNonvirtualIntMethod	function-pointer.
02 CallNonvirtualIntMethodV	function-pointer.
02 CallNonvirtualIntMethodA	function-pointer.
02 CallNonvirtualLongMethod	function-pointer.
02 CallNonvirtualLongMethodV	function-pointer.
02 CallNonvirtualLongMethodA	function-pointer.
02 CallNonvirtualFloatMethod	function-pointer.
02 CallNonvirtualFloatMethodV	function-pointer.
02 CallNonvirtualFloatMethodA	function-pointer.
02 CallNonvirtualDoubleMethod	function-pointer.
02 CallNonvirtualDoubleMethodV	function-pointer.
02 CallNonvirtualDoubleMethodA	function-pointer.
02 CallNonvirtualVoidMethod	function-pointer.
02 CallNonvirtualVoidMethodV	function-pointer.
02 CallNonvirtualVoidMethodA	function-pointer.
02 GetFieldID	function-pointer.
02 GetObjectField	function-pointer.
02 GetBooleanField	function-pointer.
02 GetByteField	function-pointer.
02 GetCharField	function-pointer.
02 GetShortField	function-pointer.
02 GetIntField	function-pointer.
02 GetLongField	function-pointer.
02 GetFloatField	function-pointer.
02 GetDoubleField	function-pointer.

02 SetObjectField	function-pointer.
02 SetBooleanField	function-pointer.
02 SetByteField	function-pointer.
02 SetCharField	function-pointer.
02 SetShortField	function-pointer.
02 SetIntField	function-pointer.
02 SetLongField	function-pointer.
02 SetFloatField	function-pointer.
02 SetDoubleField	function-pointer.
02 GetStaticMethodID	function-pointer.
02 CallStaticObjectMethod	function-pointer.
02 CallStaticObjectMethodV	function-pointer.
02 CallStaticObjectMethodA	function-pointer.
02 CallStaticBooleanMethod	function-pointer.
02 CallStaticBooleanMethodV	function-pointer.
02 CallStaticBooleanMethodA	function-pointer.
02 CallStaticByteMethod	function-pointer.
02 CallStaticByteMethodV	function-pointer.
02 CallStaticByteMethodA	function-pointer.
02 CallStaticCharMethod	function-pointer.
02 CallStaticCharMethodV	function-pointer.
02 CallStaticCharMethodA	function-pointer.
02 CallStaticShortMethod	function-pointer.
02 CallStaticShortMethodV	function-pointer.
02 CallStaticShortMethodA	function-pointer.
02 CallStaticIntMethod	function-pointer.
02 CallStaticIntMethodV	function-pointer.
02 CallStaticIntMethodA	function-pointer.
02 CallStaticLongMethod	function-pointer.
02 CallStaticLongMethodV	function-pointer.
02 CallStaticLongMethodA	function-pointer.
02 CallStaticFloatMethod	function-pointer.
02 CallStaticFloatMethodV	function-pointer.
02 CallStaticFloatMethodA	function-pointer.
02 CallStaticDoubleMethod	function-pointer.
02 CallStaticDoubleMethodV	function-pointer.
02 CallStaticDoubleMethodA	function-pointer.
02 CallStaticVoidMethod	function-pointer.
02 CallStaticVoidMethodV	function-pointer.
02 CallStaticVoidMethodA	function-pointer.
02 GetStaticFieldID	function-pointer.
02 GetStaticObjectField	function-pointer.
02 GetStaticBooleanField	function-pointer.
02 GetStaticByteField	function-pointer.
02 GetStaticCharField	function-pointer.
02 GetStaticShortField	function-pointer.
02 GetStaticIntField	function-pointer.
02 GetStaticLongField	function-pointer.
02 GetStaticFloatField	function-pointer.
02 GetStaticDoubleField	function-pointer.
02 SetStaticObjectField	function-pointer.
02 SetStaticBooleanField	function-pointer.
02 SetStaticByteField	function-pointer.
02 SetStaticCharField	function-pointer.
02 SetStaticShortField	function-pointer.
02 SetStaticIntField	function-pointer.
02 SetStaticLongField	function-pointer.
02 SetStaticFloatField	function-pointer.
02 SetStaticDoubleField	function-pointer.
02 NewString	function-pointer.
02 GetStringLength	function-pointer.
02 GetStringChars	function-pointer.
02 ReleaseStringChars	function-pointer.
02 NewStringUTF	function-pointer.
02 GetStringUTFLength	function-pointer.
02 GetStringUTFChars	function-pointer.
02 ReleaseStringUTFChars	function-pointer.

02 GetArrayLength	function-pointer.
02 NewObjectArray	function-pointer.
02 GetObjectArrayElement	function-pointer.
02 SetObjectArrayElement	function-pointer.
02 NewBooleanArray	function-pointer.
02 NewByteArray	function-pointer.
02 NewCharArray	function-pointer.
02 NewShortArray	function-pointer.
02 NewIntArray	function-pointer.
02 NewLongArray	function-pointer.
02 NewFloatArray	function-pointer.
02 NewDoubleArray	function-pointer.
02 GetBooleanArrayElements	function-pointer.
02 GetByteArrayElements	function-pointer.
02 GetCharArrayElements	function-pointer.
02 GetShortArrayElements	function-pointer.
02 GetIntArrayElements	function-pointer.
02 GetLongArrayElements	function-pointer.
02 GetFloatArrayElements	function-pointer.
02 GetDoubleArrayElements	function-pointer.
02 ReleaseBooleanArrayElements	function-pointer.
02 ReleaseByteArrayElements	function-pointer.
02 ReleaseCharArrayElements	function-pointer.
02 ReleaseShortArrayElements	function-pointer.
02 ReleaseIntArrayElements	function-pointer.
02 ReleaseLongArrayElements	function-pointer.
02 ReleaseFloatArrayElements	function-pointer.
02 ReleaseDoubleArrayElements	function-pointer.
02 GetBooleanArrayRegion	function-pointer.
02 GetByteArrayRegion	function-pointer.
02 GetCharArrayRegion	function-pointer.
02 GetShortArrayRegion	function-pointer.
02 GetIntArrayRegion	function-pointer.
02 GetLongArrayRegion	function-pointer.
02 GetFloatArrayRegion	function-pointer.
02 GetDoubleArrayRegion	function-pointer.
02 SetBooleanArrayRegion	function-pointer.
02 SetByteArrayRegion	function-pointer.
02 SetCharArrayRegion	function-pointer.
02 SetShortArrayRegion	function-pointer.
02 SetIntArrayRegion	function-pointer.
02 SetLongArrayRegion	function-pointer.
02 SetFloatArrayRegion	function-pointer.
02 SetDoubleArrayRegion	function-pointer.
02 RegisterNatives	function-pointer.
02 UnregisterNatives	function-pointer.
02 MonitorEnter	function-pointer.
02 MonitorExit	function-pointer.
02 GetJavaVM	function-pointer.
02 GetStringRegion	function-pointer.
02 GetStringUTFRegion	function-pointer.
02 GetPrimitiveArrayCritical	function-pointer.
02 ReleasePrimitiveArrayCritical	function-pointer.
02 GetStringCritical	function-pointer.
02 ReleaseStringCritical	function-pointer.
02 NewWeakGlobalRef	function-pointer.
02 DeleteWeakGlobalRef	function-pointer.
02 ExceptionCheck	function-pointer.

#### 関連タスク

333 ページの『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コンパイル』

743 ページの『JNI サービスへのアクセス』



---

## 付録 G. 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・リストの代わりに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。

873 ページの『例: SYSADATA』

### 関連参照

350 ページの『ADATA』

『SYSADATA ファイルに影響するコンパイラー・オプション』

872 ページの『SYSADATA レコード・タイプ』

874 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ー・オプション

いくつかのコンパイラー・オプションは、SYSADATA ファイルの内容に影響を与える可能性があります。

### 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、コンパイラーは、生成されたシーケンス番号ではなく、ソース・レコードのカラム 1 から 6 の内容を行番号に使用します。無効 (非数値)、または順不同の番号は、直前のレコードより 1 だけ大きな数値で置き換えられます。

**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

タイプ	フィールド	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、コンパイラー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ー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。

残りのコンパイラー・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。

873 ページの『例: SYSADATA』

#### 関連参照

『SYSADATA レコード・タイプ』

362 ページの『COMPILE』

374 ページの『EXIT』

385 ページの『LANGUAGE』

394 ページの『NUMBER』

423 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 117. SYSADATA レコード・タイプ

レコード・タイプ	アクション
877 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
878 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

表 117. SYSADATA レコード・タイプ (続き)

レコード・タイプ	アクション
878 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
878 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ー・オプションを記述します
890 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
891 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
906 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
920 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
920 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
921 ページの『COPY REPLACING レコード: X'0039'』	コピーブック内のテキストとの、 COPY.?.?.REPLACING <i>operand-1</i> の突き合わせの結果として、テキスト置換のインスタンスを記述します
922 ページの『記号レコード: X'0042'』	プログラムで定義されてい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。
933 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
934 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
935 ページの『ライブラリー・レコード: X'0060'』	各ライブラリーで使用されるライブラリー・ファイルとメンバーを記述します
936 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
936 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、レコードの先頭に標準 ADATA ヘッダーが追加され、EVENTS レコード・データの長さを示すフィールドが追加されています。

## 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  
000006  
000007  
000008

WORKING-STORAGE SECTION.  
77 COMP3-FLD2 pic S9(3)v9.  
PROCEDURE DIVISION.  
STOP RUN.

AD000060  
AD000070  
AD000080

タイプ	説明
X'0120'	EVENTS タイム・スタンプ・レコード
X'0120'	EVENTS プロセッサ・レコード
X'0120'	EVENTS ファイル ID レコード
X'0120'	EVENTS プログラム・レコード
X'0001'	ADATA 識別レコード
X'0000'	ジョブ識別レコード
X'0010'	オプション・レコード
X'0038'	ステートメント 1 のソース・レコード
X'0038'	ステートメント 2 のソース・レコード
X'0038'	ステートメント 3 のソース・レコード
X'0038'	ステートメント 4 のソース・レコード
X'0038'	ステートメント 5 のソース・レコード
X'0038'	ステートメント 6 のソース・レコード
X'0038'	ステートメント 7 のソース・レコード
X'0038'	ステートメント 8 のソース・レコード
X'0020'	AD04202 の外部シンボル・レコード
X'0044'	STOP の記号相互参照レコード
X'0044'	COMP3-FLD2 の記号相互参照レコード
X'0044'	AD04202 の記号相互参照レコード
X'0042'	AD04202 の記号レコード
X'0042'	COMP3-FLD2 の記号レコード
X'0090'	統計レコード
X'0120'	EVENTS ファイル終わりレコード

#### 関連参照

『SYSADATA レコード記述』

---

## SYSADATA レコード記述

関連データ・ファイルに書き込まれるレコードのフォーマットは、以下の関連参照に示されています。

それぞれのレコード・タイプに記述されたフィールドは、以下のシンボルで表されます。

- C 文字 (EBCDIC または ASCII) データを表す
- H 2 バイトの 2 進整数データを表す
- F 4 バイトの 2 進整数データを表す
- A 4 バイトの 2 進整数アドレスとオフセット・データを表す



X 16 進数 (ビット) データまたは 1 バイトの 2 進整数データを表す

データ型には、境界合わせは一切含まれていません。したがって、上記の暗黙の長さは、長さ指標 (Ln) を含めることで変更される可能性があります。整数データはすべて、ヘッダー・フラグ・バイト内の指標ビットに応じて、ビッグ・エンディアン形式またはリトル・エンディアン形式になっています。ビッグ・エンディアン形式では、ビット 0 が常に最上位ビットで、ビット *n* が最下位ビットであることを意味します。リトル・エンディアンとは、Intel プロセッサで見られる「逆順バイト」整数のことです。

未定義フィールドおよび未使用値はすべて、予約済みです。

#### 関連参照

『共通ヘッダー・セクション』

877 ページの『ジョブ識別レコード: X'0000'』

878 ページの『ADATA 識別レコード: X'0001'』

878 ページの『コンパイル単位の開始|終了レコード: X'0002'』

878 ページの『オプション・レコード: X'0010'』

890 ページの『外部シンボル・レコード: X'0020'』

891 ページの『構文解析ツリー・レコード: X'0024'』

906 ページの『トークン・レコード: X'0030'』

920 ページの『ソース・エラー・レコード: X'0032'』

920 ページの『ソース・レコード: X'0038'』

921 ページの『COPY REPLACING レコード: X'0039'』

922 ページの『記号レコード: X'0042'』

933 ページの『記号相互参照レコード: X'0044'』

934 ページの『ネストされたプログラム・レコード: X'0046'』

935 ページの『ライブラリー・レコード: X'0060'』

936 ページの『統計レコード: X'0090'』

936 ページの『EVENTS レコード: X'0120'』

---

## 共通ヘッダー・セクション

次の表は、すべてのレコード・タイプに共通するヘッダー・セクションの形式を示しています。MVS および VSE の場合、各レコードの前に 4 バイトの RDW (レコード記述子ワード) が置かれます。この RDW は通常、アクセス方式によってのみ使用され、ダウンロード・ユーティリティによって、ストリップされます。

表 118. 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	<b>16</b> 高水準アセンブラー
		<b>17</b> すべてのプラットフォームの COBOL
		<b>40</b> サポートされるプラットフォームの PL/I

表 118. 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	<p>以下のいずれかのレコード・タイプ。</p> <p><b>X'0000'</b> ジョブ識別レコード <sup>1</sup></p> <p><b>X'0001'</b> ADATA 識別レコード</p> <p><b>X'0002'</b> コンパイル単位の開始/終了レコード</p> <p><b>X'0010'</b> オプション・レコード <sup>1</sup></p> <p><b>X'0020'</b> 外部シンボル・レコード</p> <p><b>X'0024'</b> 構文解析ツリー・レコード</p> <p><b>X'0030'</b> トークン・レコード</p> <p><b>X'0032'</b> ソース・エラー・レコード</p> <p><b>X'0038'</b> ソース・レコード</p> <p><b>X'0039'</b> COPY REPLACING レコード</p> <p><b>X'0042'</b> 記号レコード</p> <p><b>X'0044'</b> 記号相互参照レコード</p> <p><b>X'0046'</b> ネストされたプログラム・レコード</p> <p><b>X'0060'</b> ライブラリー・レコード</p> <p><b>X'0090'</b> 統計レコード <sup>1</sup></p> <p><b>X'0120'</b> EVENTS レコード</p>
関連データ・アーキテクチャー・レベル	XL1	<b>3</b> ヘッダー構造の定義レベル
フラグ	XL1	<p><b>.... ..1.</b> ADATA レコード整数はリトル・エンディアン (Intel) 形式です</p> <p><b>.... ....1</b> このレコードは、次のレコードに続行されます</p> <p><b>1111 11..</b> 将来の利用のために予約済み</p>
関連データ・レコード・エディション・レベル	XL1	特定のレコード・タイプの新規形式を表すために使用され、通常は 0 です
予約済み	CL4	将来の利用のために予約済み
関連データ・フィールド長	HL2	ヘッダーの後に置かれるデータの長さ (バイト単位)

表 118. SYSADATA 共通ヘッダー・セクション (続き)

フィールド	サイズ	説明
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 119. 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・プロシーチャーの MVS プロシーチャー・ステップ名
入力ファイル数 <sup>1</sup>	HL2	このレコードに記録した入力ファイルの数  以下の、7 つのフィールドのグループでは、このフィールドの値に従って、 <i>n</i> 回発生します。
...入力ファイル番号	HL2	ファイルの割り当てシーケンス番号
...入力ファイル名長	HL2	次の入力ファイル名の長さ
...ボリューム通し番号長	HL2	ボリューム通し番号の長さ
...メンバー名長	HL2	メンバー名の長さ
...入力ファイル名	CL( <i>n</i> )	コンパイルの入力ファイルの名前
...ボリューム通し番号	CL( <i>n</i> )	入力ファイルが常駐する (最初の) ボリュームのボリューム通し番号
...メンバー名	CL( <i>n</i> )	該当する場合には、入力ファイル内のメンバーの名前
1. 入力ファイル数が、関連データ・ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。入力ファイルの現行数 (そのレコードの) はレコードに保管され、レコードは関連データ・ファイルに書き込まれます。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 120. 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 <sup>1</sup>	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 121. 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 122. SYSADATA オプション・レコード

フィールド	サイズ	説明
オプション・バイト 0	XL1	1111 1111 将来の利用のために予約済み

表 122. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 1	XL1	<p><b>1... ....</b> ビット 1 = DECK、ビット 0 = NODECK</p> <p><b>.1.. ....</b> ビット 1 = ADATA、ビット 0 = NOADATA</p> <p><b>..1. ....</b> ビット 1 = COLLSEQ(EBCDIC)、ビット 0 = COLLSEQ(LOCALE BINARY) (AIX のみ)</p> <p><b>...1 ....</b> ビット 1 = SEPOBJ、ビット 0 = NOSEPOBJ (AIX のみ)</p> <p><b>.... 1...</b> ビット 1 = NAME、ビット 0 = NONAME</p> <p><b>.... .1..</b> ビット 1 = OBJECT、ビット 0 = NOOBJECT</p> <p><b>.... ..1.</b> ビット 1 = SQL、ビット 0 = NOSQL</p> <p><b>.... ...1</b> ビット 1 = CICS、ビット 0 = NOCICS</p>
オプション・バイト 2	XL1	<p><b>1... ....</b> ビット 1 = OFFSET、ビット 0 = NOOFFSET</p> <p><b>.1.. ....</b> ビット 1 = MAP、ビット 0 = NOMAP</p> <p><b>..1. ....</b> ビット 1 = LIST、ビット 0 = NOLIST</p> <p><b>...1 ....</b> ビット 1 = DBCSXREF、ビット 0 = NODBCSXREF</p> <p><b>.... 1...</b> ビット 1 = XREF(SHORT)、ビット 0 = XREF(SHORT) ではない。このフラグは、ビット 7 のフラグと組み合わせて使用します。このフラグによって、XREF(FULL) はオフ状態で示され、ビット 7 のフラグはオン状態で示されます。</p> <p><b>.... .1..</b> ビット 1 = SOURCE、ビット 0 = NOSOURCE</p> <p><b>.... ..1.</b> ビット 1 = VBREF、ビット 0 = NOVBREF</p> <p><b>.... ...1</b> ビット 1 = XREF、ビット 0 = XREF ではない。ビット 4 以降のフラグも参照。</p>

表 122. **SYSADATA** オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 3	XL1	<p><b>1...</b> ....  ビット 1 = 指定される FLAG 組み込み診断レベル (FLAG(x,y) のように値 y が指定されます)</p> <p><b>.1..</b> ....  ビット 1 = FLAGSTD、ビット 0 = NOFLAGSTD</p> <p><b>..1.</b> ....  ビット 1 = NUM、ビット 0 = NONUM</p> <p><b>...1</b> ....  ビット 1 = SEQUENCE、ビット 0 = NOSEQUENCE</p> <p><b>.... 1...</b>  ビット 1 = SOSI、ビット 0 = NOSOSI (AIX のみ)</p> <p><b>.... .1..</b>  ビット 1 = NSYMBOL(NATIONAL)、ビット 0 = NSYMBOL(DBCS)</p> <p><b>.... ..1.</b>  ビット 1 = PROFILE、ビット 0 = NOPROFILE (AIX のみ)</p> <p><b>.... ...1</b>  ビット 1 = WORD、ビット 0 = NOWORD</p>
オプション・バイト 4	XL1	<p><b>1...</b> ....  ビット 1 = ADV、ビット 0 = NOADV</p> <p><b>.1..</b> ....  ビット 1 = APOST、ビット 0 = QUOTE</p> <p><b>..1.</b> ....  ビット 1 = DYNAM、ビット 0 = NODYNAM</p> <p><b>...1</b> ....  ビット 1 = AWO、ビット 0 = NOAWO</p> <p><b>.... 1...</b>  ビット 1 = 指定済み RMODE、ビット 0 = RMODE(AUTO)</p> <p><b>.... .1..</b>  ビット 1 = RENT、ビット 0 = NORENT</p> <p><b>.... ..1.</b>  ビット 1 = RES: COBOL の場合、このフラグは常にオンに設定されます。</p> <p><b>.... ...1</b>  ビット 1 = RMODE(24)、ビット 0 = RMODE(ANY)</p>

表 122. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 5	XL1	<p><b>1...</b> .... ビット 1 = SQLCCSID、ビット 0 = NOSQLCCSID</p> <p><b>.1..</b> .... ビット 1 = OPT(1 2)、ビット 0 = OPT(0)</p> <p><b>..1.</b> .... ビット 1 = SQLIMS、ビット 0 = NOSQLIMS</p> <p><b>...1</b> .... ビット 1 = DBCS、ビット 0 = NODBCS</p> <p><b>.... 1...</b> ビット 1 = AFP(VOLATILE)、ビット 0 = AFP(NOVOLATILE)</p> <p><b>.... .1..</b> ビット 1 = SSRANGE、ビット 0 = NOSSRANGE</p> <p><b>.... ..1.</b> ビット 1 = TEST、ビット 0 = NOTEST</p> <p><b>.... ...1</b> ビット 1 = PROBE、ビット 0 = NOPROBE (Windows のみ)</p>
オプション・バイト 6	XL1	<p><b>1...</b> .... ビット 1 = SRCFORMAT(EXTEND)、ビット 0 = SRCFORMAT(COMPAT)</p> <p><b>..1.</b> .... ビット 1 = NUMPROC(PFD)、ビット 0 = NUMPROC(NOPFD)</p> <p><b>...1</b> .... ビット 1 = NUMCLS(ALT)、ビット 0 = NUMCLS(PRIM)</p> <p><b>.... .1..</b> ビット 1 = BINARY(S390)、ビット 0 = BINARY(NATIVE) (AIX のみ)</p> <p><b>.... ..1.</b> ビット 1 = TRUNC(STD)、ビット 0 = TRUNC(OPT)</p> <p><b>.... ...1</b> ビット 1 = ZWB、ビット 0 = NOZWB</p> <p><b>.1.. 1...</b> 将来の利用のために予約済み</p>

表 122. **SYSADATA** オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト 7	XL1	<p><b>1... ....</b> ビット 1 = ALLOWCBL、ビット 0 = NOALLOWCBL</p> <p><b>.1.. ....</b> ビット 1 = TERM、ビット 0 = NOTERM</p> <p><b>..1. ....</b> ビット 1 = DUMP、ビット 0 = NODUMP</p> <p><b>.... ..1.</b> ビット 1 = CURRENCY、ビット 0 = NOCURRENCY</p> <p><b>...1 11.1</b> 将来の利用のために予約済み</p>
オプション・バイト 8	XL1	<p><b>1... ....</b> ビット 1 = RULES、ビット 0 = NORULES</p> <p><b>.1.. ....</b> ビット 1 = OPTFILE、ビット 0 = not OPTFILE</p> <p><b>..1. ....</b> ビット 1 = ADDR(64)、ビット 0 = ADDR(32) (AIX のみ)</p> <p><b>.... 1...</b> ビット 1 = BLOCK0、ビット 0 = NOBLOCK0</p> <p><b>.... ..1.</b> ビット 1 = DISPSIGN(SEP)、ビット 0 = DISPSIGN(COMPAT)</p> <p><b>.... ...1</b> ビット 1 = STGOPT、ビット 0 = NOSTGOPT</p> <p><b>1..1 .1..</b> 将来の利用のために予約済み</p>
オプション・バイト 9	XL1	<p><b>1... ....</b> ビット 1 = DATA(24)、ビット 0 = DATA(31)</p> <p><b>.1.. ....</b> ビット 1 = FASTSRT、ビット 0 = NOFASTSRT</p> <p><b>.... .1..</b> ビット 1 = THREAD、ビット 0 = NOTHREAD</p> <p><b>..11 1.11</b> 将来の利用のために予約済み</p>



表 122. 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 A	XL1	<p><b>1... ....</b> ビット 1 = HGPR(PRESERVE)、ビット 0 = HGPR(NOPRESERVE)</p> <p><b>.1.. ....</b> ビット 1 = XMLPARSE(XMLSS)、ビット 0 = XMLPARSE(COMPAT)</p> <p><b>..1. ....</b> ビット 1 = MAP(DEC)、ビット 0 = MAP(HEX)</p> <p><b>...1....</b> 将来の利用のために予約済み</p> <p><b>....1...</b> ビット 1 = SUPPRESS、ビット 0 = NOSUPPRESS</p> <p><b>.....1..</b> ビット 1 = VSAMOPENFS(SUCC)、ビット 0 = VSAMOPENFS(COMPAT)</p> <p><b>.....11</b> 将来の利用のために予約済み</p>
オプション・バイト B	XL1	<p><b>1111 1111</b> 将来の利用のために予約済み</p>
オプション・バイト C	XL1	<p><b>1... ....</b> ビット 1 = NCOLLSEQ(LOCALE) (AIX のみ)</p> <p><b>.1.. ....</b> 将来の利用のために予約済み</p> <p><b>..1. ....</b> ビット 1 = INTDATE(LILIAN)、ビット 0 = INTDATE(ANSI)</p> <p><b>...1 ....</b> ビット 1 = NCOLLSEQ(BINARY) (AIX のみ)</p> <p><b>.... 1...</b> ビット 1 = CHAR(EBCDIC)、ビット 0 = CHAR(NATIVE) (AIX のみ)</p> <p><b>.... .1..</b> ビット 1 = FLOAT(HEX)、ビット 0 = FLOAT(NATIVE) (AIX のみ)</p> <p><b>.... ..1.</b> ビット 1 = COLLSEQ(BINARY) (AIX のみ)</p> <p><b>.... ....1</b> ビット 1 = COLLSEQ(LOCALE) (AIX のみ)</p>

表 122. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
オプション・バイト D	XL1	<p><b>1... ....</b> ビット 1 = DLL、ビット 0 = NODLL</p> <p><b>.1.. ....</b> ビット 1 = EXPORTALL、ビット 0 = NOEXPORTALL</p> <p><b>..1. ....</b> ビット 1 = CODEPAGE</p> <p><b>...1 ....</b> ビット 1 = SOURCEFORMAT(EXTEND)、ビット 0 = SOURCEFORMAT(COMPAT) (AIX のみ)</p> <p><b>.... ..1.</b> ビット 1 = WSCLEAR、ビット 0 = NOWSCLEAR (AIX のみ)</p> <p><b>.... ...1</b> ビット 1 = BEOPT、ビット 0 = NOBEOPT (AIX のみ)</p> <p><b>.... 11..</b> 将来の利用のために予約済み</p>
オプション・バイト E	XL1	<p><b>1.....</b> ビット 1 = VLR(COMPAT)、ビット 0 = VLR(STANDARD)</p> <p><b>.1.. ....</b> ビット 1 = DIAGTRUNC、ビット 0 = NODIAGTRUNC</p> <p><b>.... .1..</b> ビット 1 = LSTFILE(UTF-8)、ビット 0 = LSTFILE(LOCALE) (AIX のみ)</p> <p><b>.... ..1.</b> ビット 1 = MDECK、ビット 0 = NOMDECK</p> <p><b>.... ...1</b> ビット 1 = MDECK(NOCOMPILE)</p> <p><b>..11 1...</b> 将来の利用のために予約済み</p>

表 122. 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	<p><b>1... ....</b> ビット 1 = DIVIDE(S390)、ビット 0 = DIVIDE(NATIVE) (AIX のみ)</p> <p><b>.1... ....</b> ビット 1 = COPYRIGHT、ビット 0 = NOCOPYRIGHT</p> <p><b>..1. ....</b> ビット 1 = QUALIFY(EXTEND)、ビット 0 = QUALIFY(COMPAT)</p> <p><b>...1 ....</b> ビット 1 = SERVICE、ビット 0 = NOSERVICE</p> <p><b>.... 1...</b> ビット 1 = ZONEDATA(MIG)</p> <p><b>.... .1..</b> ビット 1 = ZONEDATA(NOPFD)</p> <p><b>.... ..1.</b> ビット 1 = NUMCHECK(ZON PAC BIN ABD MSG)、 ビット 0 = NONUMCHECK</p> <p><b>.... ...1</b> ビット 1 = PARMCHECK(ABD MSG)、ビット 0 = NOPARMCHECK</p>
オプション・バイト G	XL1	<p><b>1... ....</b> ビット 1 = NUMCHECK(ZON)、ビット 0 = NUMCHECK(NOZON)</p> <p><b>.1... ....</b> ビット 1 = NUMCHECK(PAC)、ビット 0 = NUMCHECK(NOPAC)</p> <p><b>..1. ....</b> ビット 1 = NUMCHECK(BIN)、ビット 0 = NUMCHECK(NOBIN)</p> <p><b>...1 ....</b> ビット 1 = NUMCHECK(MSG)、ビット 0 = NUMCHECK(ABD)</p> <p><b>.... 1...</b> ビット 1 = NUMCHECK(ZON(NOALPHNUM))、ビット 0 = NUMCHECK(ZON(ALPHNUM))</p> <p><b>.... .111</b> 将来の利用のために予約済み</p>

表 122. 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 H	XL1	<b>1... ....</b> ビット 1 = PARMCHECK(ABD)、ビット 0 = PARMCHECK(MSG) (PARMCHECK が有効になっている 場合)  <b>.1... ....</b> 将来の利用のために予約済み
フラグ・レベル	XL1	<b>X'00'</b> フラグ (I) <b>X'04'</b> フラグ (W) <b>X'08'</b> フラグ (E) <b>X'0C'</b> フラグ (S) <b>X'10'</b> フラグ (U) <b>X'FF'</b> Noflag
組み込み診断レベル	XL1	<b>X'00'</b> フラグ (I) <b>X'04'</b> フラグ (W) <b>X'08'</b> フラグ (E) <b>X'0C'</b> フラグ (S) <b>X'10'</b> フラグ (U) <b>X'FF'</b> Noflag
FLAGSTD (FIPS) 指定	XL1	<b>1... ....</b> 最小  <b>.1... ....</b> 中間  <b>..1. ....</b> 高  <b>...1 ....</b> IBM 拡張  <b>.... 1...</b> レベル 1 セグメンテーション  <b>.... .1..</b> レベル 2 セグメンテーション  <b>.... ..1.</b> デバッグ  <b>.... ...1</b> 廃止
フラグ用に予約済み	XL1	<b>1111 1111</b> 将来の利用のために予約済み

表 122. 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	<b>X'00'</b> 無条件 Nocompile、Nocompile(I) <b>X'04'</b> Nocompile(W) <b>X'08'</b> Nocompile(E) <b>X'0C'</b> Nocompile(S) <b>X'FF'</b> コンパイル
スペース値	CL1	
3 値オプション用のデータ	XL1	<b>1... ....</b> NAME(ALIAS) 指定 <b>.1... ....</b> 将来の利用のために予約済み <b>..1. ....</b> TRUNC(BIN) 指定 <b>...1 ....</b> PARMCHECK(ABD) (PARMCHECK が有効になっている場合) <b>.... 1111</b> 将来の利用のために予約済み
TEST サブオプション	XL1	<b>1... ....</b> TEST(EJPD) <b>.1... ....</b> TEST(SOURCE) <b>..1. ....</b> TEST NOTEST(SEPARATE) <b>...1 ....</b> NOTEST(DWARF) <b>.... 1...</b> TEST NOTEST(SEPARATE(DSNAME)) <b>.... .111</b> 将来の利用のために予約済み
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	プロダクト・ビルド・レベル情報 (形式: PYYMMDD)

表 122. 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	<p>1... .... ビット 1 = PGMNAME (COMPAT)</p> <p>.1... .... ビット 1 = PGMNAME (LONGUPPER)</p> <p>..1. .... ビット 1 = PGMNAME (LONGMIXED)</p> <p>...1 1111 将来の利用のために予約済み</p>
記入項目インターフェース・サブオプション	XL1	<p>1... .... ビット 1 = EntryInterface(System) (Windows のみ)</p> <p>.1... .... ビット 1 = EntryInterface(OptLink) (Windows のみ)</p> <p>..11 1111 将来の利用のために予約済み</p>
CALLINTERFACE サブオプション	XL1	<p>1... .... ビット 1 = CALLINTERFACE (DLL)</p> <p>.1... .... ビット 1 = CALLINTERFACE (DYNAMIC)</p> <p>..11 1111 将来の利用のために予約済み</p>
ARITH サブオプション	XL1	<p>1... .... ビット 1 = ARITH (COMPAT)</p> <p>.1... .... ビット 1 = ARITH (EXTEND)</p> <p>..11 1111 将来の利用のために予約済み</p>
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	このフィールドには、LANGUAGE オプションからの 2 文字の省略形 (EN、UE、JA、または JP のいずれか) が保持されます。
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

表 122. SYSADATA オプション・レコード (続き)

フィールド	サイズ	説明
LIBEXIT 名長	HL2	Library ユーザー出口名の長さ
ADEXIT 名長	HL2	ADATA ユーザー出口名の長さ
CURROPT	CL5	CURRENCY オプション値
ARCH	XL1	ARCH レベル番号
予約済み	CL2	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	<p><b>1... ....</b> ビット 1 = ユーザー領域内の IGYCLIBR</p> <p><b>.1.. ....</b> ビット 1 = ユーザー領域内の IGYCSCAN</p> <p><b>..1. ....</b> ビット 1 = ユーザー領域内の IGYCDSCN</p> <p><b>...1 ....</b> ビット 1 = ユーザー領域内の IGYCGROU</p> <p><b>.... 1...</b> ビット 1 = ユーザー領域内の IGYCPSCN</p> <p><b>.... .1..</b> ビット 1 = ユーザー領域内の IGYCPANA</p> <p><b>.... ..1.</b> ビット 1 = ユーザー領域内の IGYCFGEN</p> <p><b>.... ...1</b> ビット 1 = ユーザー領域内の IGYCPGEN</p>
フェーズ常駐ビット バイト 2	XL1	<p><b>.1.. ....</b> ビット 1 = ユーザー領域内の IGYCLSTR</p> <p><b>..1. ....</b> ビット 1 = ユーザー領域内の IGYCXREF</p> <p><b>...1 ....</b> ビット 1 = ユーザー領域内の IGYCDMAP</p> <p><b>.... ..1.</b> ビット 1 = ユーザー領域内の IGYCDIAG</p> <p><b>.... ...1</b> ビット 1 = ユーザー領域内の IGYCDGEN</p> <p><b>1... 11..</b> 将来の利用のために予約済み</p>

表 122. 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	CL8	将来の利用のために予約済み
OUTDD 名	CL(n)	OUTDD 名
RWT	CL(n)	予約語テーブル ID
DBCS ORDPGM	CL(n)	DBCS 配列プログラム名
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 123. SYSADATA 外部シンボル・レコード

フィールド	サイズ	説明
セクション・タイプ	XL1	<p><b>X'00'</b> PROGRAM-ID 名 (メインエントリー・ポイント名)</p> <p><b>X'01'</b> ENTRY 名 (2 次エントリー・ポイント名)</p> <p><b>X'02'</b> 外部参照 (参照先の外部エントリー・ポイント)</p> <p><b>X'04'</b> COBOL の場合は適用外</p> <p><b>X'05'</b> COBOL の場合は適用外</p> <p><b>X'06'</b> COBOL の場合は適用外</p> <p><b>X'0A'</b> COBOL の場合は適用外</p> <p><b>X'12'</b> 内部参照 (参照先の内部サブプログラム)</p> <p><b>X'C0'</b> 外部クラス名 (OO COBOL クラス定義)</p> <p><b>X'C1'</b> METHOD-ID 名 (OO COBOL メソッド定義)</p> <p><b>X'C6'</b> メソッド参照 (OO COBOL メソッド参照)</p> <p><b>X'FF'</b> COBOL の場合は適用外</p> <p>タイプ X'12'、X'C0'、X'C1'、および X'C6' は、COBOL のみ対応です。</p>
フラグ	XL1	COBOL の場合は適用外
予約済み	HL2	将来の利用のために予約済み
記号 ID	FL4	参照を含むプログラムの記号 ID (タイプ x'02' および x'12' の場合のみ)
行番号	FL4	参照を含むステートメントの行番号 (タイプ x'02' および x'12' の場合のみ)
セクション長	FL4	COBOL の場合は適用外



表 123. SYSADATA 外部シンボル・レコード (続き)

フィールド	サイズ	説明
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。

表 124. 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: <b>001</b> プログラム <b>002</b> クラス <b>003</b> メソッド
		<b>101</b> IDENTIFICATION DIVISION <b>102</b> ENVIRONMENT DIVISION <b>103</b> データ部 <b>104</b> 手続き部 <b>105</b> 終了プログラム/メソッド/クラス
		<b>201</b> 宣言本文 <b>202</b> 非宣言本文
		<b>301</b> セクション <b>302</b> プロシージャラー・セクション
		<b>401</b> 段落 <b>402</b> 手順段落
		<b>501</b> 文 <b>502</b> ファイル定義 <b>503</b> ソート・ファイル定義 <b>504</b> プログラム名 <b>505</b> プログラム属性 <b>508</b> ENVIRONMENT DIVISION 節 <b>509</b> CLASS 属性 <b>510</b> METHOD 属性 <b>511</b> USE ステートメント

表 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		<b>601</b> ステートメント <b>602</b> データ記述節 <b>603</b> データ入力項目 <b>604</b> ファイル記述節 <b>605</b> データ入力項目名 <b>606</b> データ入力項目レベル <b>607</b> EXEC 記入項目
		<b>701</b> EVALUATE サブジェクト句 <b>702</b> EVALUATE WHEN 句 <b>703</b> EVALUATE WHEN OTHER 句 <b>704</b> SEARCH WHEN 句 <b>705</b> INSPECT CONVERTING 句 <b>706</b> INSPECT REPLACING 句 <b>707</b> INSPECT TALLYING 句 <b>708</b> PERFORM UNTIL 句 <b>709</b> PERFORM VARYING 句 <b>710</b> PERFORM AFTER 句 <b>711</b> ステートメント・ブロック <b>712</b> 範囲終了符号 <b>713</b> INITIALIZE REPLACING 句 <b>714</b> EXEC CICS コマンド <b>715</b> INITIALIZE WITH FILLER <b>716</b> INITIALIZE TO VALUE <b>717</b> INITIALIZE TO DEFAULT <b>718</b> ALLOCATE INITIALIZED <b>719</b> ALLOCATE LOC <b>720</b> DATA DIVISION 句

I

表 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		<b>801</b> 句 <b>802</b> ON 句 <b>803</b> NOT 句 <b>804</b> THEN 句 <b>805</b> ELSE 句 <b>806</b> 条件 <b>807</b> 式 <b>808</b> 相対索引付け <b>809</b> EXEC CICS オプション <b>810</b> 予約語 <b>811</b> INITIALIZE REPLACING カテゴリー
		<b>901</b> セクションまたは段落名 <b>902</b> ID <b>903</b> 英字名 <b>904</b> クラス名 <b>905</b> 条件名 <b>906</b> ファイル名 <b>907</b> 指標名 <b>908</b> 簡略名 <b>910</b> シンボリック文字 <b>911</b> リテラル <b>912</b> 関数 ID <b>913</b> データ名 <b>914</b> 特殊レジスター <b>915</b> プロシーチャー参照 <b>916</b> 算術演算子 <b>917</b> 全プロシーチャー <b>918</b> INITIALIZE リテラル (トークンなし) <b>919</b> ALL リテラルまたは形象定数 <b>920</b> キーワード・クラス・テスト名 <b>921</b> ID レベルの予約語 <b>922</b> 単項演算子 <b>923</b> 比較演算子
		<b>1001</b> 添え字 <b>1002</b> 参照変更

表 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
ノード・サブタイプ	HL2	<p>ノードのサブタイプ。</p> <p>セクション・タイプの場合:</p> <p>0001 CONFIGURATION セクション</p> <p>0002 INPUT-OUTPUT セクション</p> <p>0003 FILE SECTION</p> <p>0004 WORKING-STORAGE SECTION</p> <p>0005 LINKAGE SECTION</p> <p>0006 LOCAL-STORAGE SECTION</p> <p>0007 REPOSITORY セクション</p>
		<p>段落タイプの場合:</p> <p>0001 PROGRAM-ID 段落</p> <p>0002 AUTHOR 段落</p> <p>0003 INSTALLATION 段落</p> <p>0004 DATE-WRITTEN 段落</p> <p>0005 SECURITY 段落</p> <p>0006 SOURCE-COMPUTER 段落</p> <p>0007 OBJECT-COMPUTER 段落</p> <p>0008 SPECIAL-NAMES 段落</p> <p>0009 FILE-CONTROL 段落</p> <p>0010 I-O-CONTROL 段落</p> <p>0011 DATE-COMPILED 段落</p> <p>0012 CLASS-ID 段落</p> <p>0013 METHOD-ID 段落</p> <p>0014 REPOSITORY 段落</p>
		<p>ENVIRONMENT DIVISION 節タイプの場合:</p> <p>0001 WITH DEBUGGING MODE</p> <p>0002 MEMORY-SIZE</p> <p>0003 SEGMENT-LIMIT</p> <p>0004 CURRENCY-SIGN</p> <p>0005 DECIMAL POINT</p> <p>0006 PROGRAM COLLATING SEQUENCE</p> <p>0007 ALPHABET</p> <p>0008 SYMBOLIC-CHARACTER</p> <p>0009 CLASS</p> <p>0010 ENVIRONMENT NAME</p> <p>0011 SELECT</p> <p>0012 XML-SCHEMA</p>

表 124. **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:
		<b>0001</b> BLANK WHEN ZERO
		<b>0002</b> DATA-NAME OR FILLER
		<b>0003</b> JUSTIFIED
		<b>0004</b> OCCURS
		<b>0005</b> PICTURE
		<b>0006</b> REDEFINES
		<b>0007</b> RENAMES
		<b>0008</b> SIGN
		<b>0009</b> SYNCHRONIZED
		<b>0010</b> USAGE
		<b>0011</b> VALUE
		<b>0012</b> VOLATILE
		<b>0023</b> GLOBAL
		<b>0024</b> EXTERNAL

表 124. **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:
		<b>0001</b> FILE STATUS
		<b>0002</b> ORGANIZATION
		<b>0003</b> ACCESS MODE
		<b>0004</b> RECORD KEY
		<b>0005</b> ASSIGN
		<b>0006</b> RELATIVE KEY
		<b>0007</b> PASSWORD
		<b>0008</b> PROCESSING MODE
		<b>0009</b> RECORD DELIMITER
		<b>0010</b> PADDING CHARACTER
		<b>0011</b> BLOCK CONTAINS
		<b>0012</b> RECORD CONTAINS
		<b>0013</b> LABEL RECORDS
		<b>0014</b> VALUE OF
		<b>0015</b> DATA RECORDS
		<b>0016</b> LINAGE
		<b>0017</b> ALTERNATE KEY
		<b>0018</b> LINES AT TOP
		<b>0019</b> LINES AT BOTTOM
		<b>0020</b> CODE-SET
		<b>0021</b> RECORDING MODE
		<b>0022</b> RESERVE
		<b>0023</b> GLOBAL
		<b>0024</b> EXTERNAL
		<b>0025</b> LOCK

表 124. 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:
		<b>0002</b> NEXT SENTENCE
		<b>0003</b> ACCEPT
		<b>0004</b> ADD
		<b>0005</b> ALTER
		<b>0006</b> CALL
		<b>0007</b> CANCEL
		<b>0008</b> CLOSE
		<b>0009</b> COMPUTE
		<b>0010</b> CONTINUE
		<b>0011</b> DELETE
		<b>0012</b> DISPLAY
		<b>0013</b> DIVIDE (INTO)
		<b>0113</b> DIVIDE (BY)
		<b>0014</b> ENTER
		<b>0015</b> ENTRY
		<b>0016</b> EVALUATE
		<b>0017</b> EXIT
		<b>0018</b> GO
		<b>0019</b> GOBACK
		<b>0020</b> IF
		<b>0021</b> INITIALIZE
		<b>0022</b> INSPECT

表 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		<b>0023</b> INVOKE
		<b>0024</b> MERGE
		<b>0025</b> MOVE
		<b>0026</b> MULTIPLY
		<b>0027</b> OPEN
		<b>0028</b> PERFORM
		<b>0029</b> READ
		<b>0030</b> READY
		<b>0031</b> RELEASE
		<b>0032</b> RESET
		<b>0033</b> RETURN
		<b>0034</b> REWRITE
		<b>0035</b> SEARCH
		<b>0036</b> SERVICE
		<b>0037</b> SET
		<b>0038</b> SORT
		<b>0039</b> START
		<b>0040</b> STOP
		<b>0041</b> STRING
		<b>0042</b> SUBTRACT
		<b>0043</b> UNSTRING
		<b>0044</b> EXEC SQL
		<b>0144</b> EXEC CICS
		<b>0045</b> WRITE
		<b>0046</b> XML
		<b>0047</b> ALLOCATE
		<b>0048</b> FREE
		<b>0049</b> JSON



表 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		句タイプの場合:
		0001 INTO
		0002 DELIMITED
		0003 INITIALIZE. . .REPLACING
		0004 INSPECT. . .ALL
		0005 INSPECT. . .LEADING
		0006 SET . . .TO
		0007 SET . . .UP
		0008 SET . . .DOWN
		0009 PERFORM . . .TIMES
		0010 DIVIDE . . .REMAINDER
		0011 INSPECT. . .FIRST
		0012 SEARCH. . .VARYING
		0013 MORE-LABELS
		0014 SEARCH ALL
		0015 SEARCH. . .AT END
		0016 SEARCH. . .TEST INDEX
		0017 GLOBAL
		0018 LABEL
		0019 DEBUGGING
		0020 SEQUENCE
		0021 将来の利用のために予約済み
		0022 将来の利用のために予約済み
		0023 将来の利用のために予約済み
		0024 TALLYING
		0025 将来の利用のために予約済み
		0026 ON SIZE ERROR
		0027 ON OVERFLOW
		0028 ON ERROR
		0029 AT END
		0030 INVALID KEY
		0031 END-OF-PAGE
		0032 USING

表 124. 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 KEY
		0042 DELIMITER
		0043 POINTER
		0044 COUNT
		0045 METHOD
		0046 PROGRAM
		0047 INPUT
		0048 OUTPUT
		0049 I-O
		0050 EXTEND
		0051 RELOAD
		0052 ASCENDING
		0053 DESCENDING
		0054 DUPLICATES
		0055 NATIVE (USAGE)
		0056 INDEXED
		0057 FROM
		0058 FOOTING
		0059 LINES AT BOTTOM
		0060 LINES AT TOP
		0061 XML ENCODING
		0062 XML GENERATE XML-DECLARATION
		0063 XML GENERATE ATTRIBUTES
		0064 XML GENERATE NAMESPACE
		0065 XML PARSE PROCESSING
		0066 XML PARSE VALIDATING
		0067 XML GENERATE NAME
		0068 XML GENERATE TYPE
		0069 XML GENERATE SUPPRESS

表 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		関数 ID タイプの場合:
		0001 COS
		0002 LOG
		0003 MAX
		0004 MIN
		0005 MOD
		0006 ORD
		0007 REM
		0008 SIN
		0009 SUM
		0010 TAN
		0011 ACOS
		0012 ASIN
		0013 ATAN
		0014 CHAR
		0015 MEAN
		0016 SQRT
		0017 LOG10
		0018 RANGE
		0019 LENGTH
		0020 MEDIAN
		0021 NUMVAL
		0022 RANDOM
		0023 ANNUITY
		0024 INTEGER
		0025 ORD-MAX
		0026 ORD-MIN
		0027 REVERSE
		0028 MIDRANGE
		0029 NUMVAL-C
		0030 VARIANCE
		0031 FACTORIAL
		0032 LOWER-CASE

表 124. 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
		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 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		特殊レジスター・タイプの場合:  <b>0001</b> ADDRESS OF <b>0002</b> LENGTH OF
		キーワード・クラス・テスト名タイプの場合:  <b>0001</b> ALPHABETIC <b>0002</b> ALPHABETIC-LOWER <b>0003</b> ALPHABETIC-UPPER <b>0004</b> DBCS <b>0005</b> KANJI <b>0006</b> NUMERIC <b>0007</b> NEGATIVE <b>0008</b> POSITIVE <b>0009</b> ZERO
		予約語タイプの場合:  <b>0001</b> TRUE <b>0002</b> FALSE <b>0003</b> ANY <b>0004</b> THRU
		ID、データ名、索引名、条件名、または簡略名タイプの場合:  <b>0001</b> REFERENCED <b>0002</b> CHANGED <b>0003</b> REFERENCED & CHANGED
		初期化リテラル・タイプの場合:  <b>0001</b> ALPHABETIC <b>0002</b> ALPHANUMERIC <b>0003</b> NUMERIC <b>0004</b> ALPHANUMERIC-EDITED <b>0005</b> NUMERIC-EDITED <b>0006</b> DBCS/EGCS <b>0007</b> NATIONAL <b>0008</b> NATIONAL-EDITED
		プロシージャー名タイプの場合:  <b>0001</b> SECTION <b>0002</b> PARAGRAPH

表 124. 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      ON
		0004      オフ
		0005      SIZE
		0006      DATE
		0007      DAY
		0008      DAY-OF-WEEK
		0009      TIME
		0010      WHEN-COMPILED
		0011      PAGE
		0012      DATE YYYYMMDD
		0013      DAY YYYYDDD
		0014      属性
		0015      エレメント
		0016      内容
		0017      数字
		0018      非数値
		0019      間隔
		0020      タイミング
		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

表 124. SYSADATA 構文解析ツリー・レコード (続き)

フィールド	サイズ	説明
		関係演算子タイプの場合:  <b>0008</b> LESS <b>0009</b> LESS OR EQUAL <b>0010</b> EQUAL <b>0011</b> NOT EQUAL <b>0012</b> GREATER <b>0013</b> GREATER OR EQUAL <b>0014</b> AND <b>0015</b> OR <b>0016</b> CLASS CONDITION <b>0017</b> NOT CLASS CONDITION
親ノード番号	FL4	ノードの親のノード番号
左方兄弟ノード番号	FL4	ノードの左方兄弟のノード番号 (ある場合)。なしの場合、値はゼロ。
記号 ID	FL4	以下のタイプのいずれかのユーザー名の場合は、ノードの記号 ID。 <ul style="list-style-type: none"> <li>データ入力項目</li> <li>ID</li> <li>ファイル名</li> <li>指標名</li> <li>プロシージャ名。</li> <li>条件名</li> <li>簡略名</li> </ul> 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。  <b>X'80'</b> 予約済み <b>X'40'</b> 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 125. 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	コンパイラが生成するソース・ファイル内のトークン番号。1 から開始。ソースにはコピーブックを組み込み済みです。



表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明																																																
トークン・コード	HL2	<p>トークンの型 (ユーザー名、リテラル、予約語など)。</p> <p>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。</p> <p>PICTURE スtringの場合は、特別コード 0000 が使用されます。</p> <p>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。</p> <p>その他の場合は、以下のコードが使用されます。</p> <table><tr><td>0001</td><td>ACCEPT</td></tr><tr><td>0002</td><td>ADD</td></tr><tr><td>0003</td><td>ALTER</td></tr><tr><td>0004</td><td>CALL</td></tr><tr><td>0005</td><td>CANCEL</td></tr><tr><td>0007</td><td>CLOSE</td></tr><tr><td>0009</td><td>COMPUTE</td></tr><tr><td>0011</td><td>DELETE</td></tr><tr><td>0013</td><td>DISPLAY</td></tr><tr><td>0014</td><td>DIVIDE</td></tr><tr><td>0017</td><td>READY</td></tr><tr><td>0018</td><td>END-PERFORM</td></tr><tr><td>0019</td><td>ENTER</td></tr><tr><td>0020</td><td>ENTRY</td></tr><tr><td>0021</td><td>EXIT</td></tr><tr><td>0022</td><td>EXEC</td></tr><tr><td></td><td>EXECUTE</td></tr><tr><td>0023</td><td>GO</td></tr><tr><td>0024</td><td>IF</td></tr><tr><td>0025</td><td>INITIALIZE</td></tr><tr><td>0026</td><td>INVOKE</td></tr><tr><td>0027</td><td>INSPECT</td></tr><tr><td>0028</td><td>MERGE</td></tr><tr><td>0029</td><td>MOVE</td></tr></table>	0001	ACCEPT	0002	ADD	0003	ALTER	0004	CALL	0005	CANCEL	0007	CLOSE	0009	COMPUTE	0011	DELETE	0013	DISPLAY	0014	DIVIDE	0017	READY	0018	END-PERFORM	0019	ENTER	0020	ENTRY	0021	EXIT	0022	EXEC		EXECUTE	0023	GO	0024	IF	0025	INITIALIZE	0026	INVOKE	0027	INSPECT	0028	MERGE	0029	MOVE
0001	ACCEPT																																																	
0002	ADD																																																	
0003	ALTER																																																	
0004	CALL																																																	
0005	CANCEL																																																	
0007	CLOSE																																																	
0009	COMPUTE																																																	
0011	DELETE																																																	
0013	DISPLAY																																																	
0014	DIVIDE																																																	
0017	READY																																																	
0018	END-PERFORM																																																	
0019	ENTER																																																	
0020	ENTRY																																																	
0021	EXIT																																																	
0022	EXEC																																																	
	EXECUTE																																																	
0023	GO																																																	
0024	IF																																																	
0025	INITIALIZE																																																	
0026	INVOKE																																																	
0027	INSPECT																																																	
0028	MERGE																																																	
0029	MOVE																																																	

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0030 MULTIPLY
		0031 OPEN
		0032 PERFORM
		0033 READ
		0035 RELEASE
		0036 RETURN
		0037 REWRITE
		0038 SEARCH
		0040 SET
		0041 SORT
		0042 START
		0043 STOP
		0044 STRING
		0045 SUBTRACT
		0048 UNSTRING
		0049 USE
		0050 WRITE
		0051 CONTINUE
		0052 END-ADD
		0053 END-CALL
		0054 END-COMPUTE
		0055 END-DELETE
		0056 END-DIVIDE
		0057 END-EVALUATE
		0058 END-IF
		0059 END-MULTIPLY
		0060 END-READ
		0061 END-RETURN
		0062 END-REWRITE
		0063 END-SEARCH
		0064 END-START
		0065 END-STRING
		0066 END-SUBTRACT
		0067 END-UNSTRING
		0068 END-WRITE
		0069 GOBACK

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0070 EVALUATE
		0071 RESET
		0072 SERVICE
		0073 END-INVOKE
		0074 END-EXEC
		0075 XML
		0076 END-XML
		0077 ALLOCATE
		0078 FREE
		0079 JSON
		0080 END-JSON
		0099 FOREIGN-VERB
		0101 DATA-NAME
		0105 DASHED-NUM
		0106 DECIMAL
		0107 DIV-SIGN
		0108 EQ
		0109 EXPONENTIATION
		0110 GT
		0111 INTEGER
		0112 LT
		0113 LPAREN
		0114 MINUS-SIGN
		0115 MULT-SIGN
		0116 NONUMLIT
		0117 PERIOD
		0118 PLUS-SIGN
		0121 RPAREN
		0122 SIGNED-INTEGER
		0123 QUID
		0124 COLON
		0125 IEOF
		0126 EGCS-LIT
		0127 COMMA-SPACE
		0128 SEMICOLON-SPACE
		0129 PROCEDURE-NAME
		0130 FLT-POINT-LIT
		0131 言語環境プログラム

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0132 GE
		0133 IDREF
		0134 EXPREF
		0136 CICS
		0137 NEW
		0138 NATIONAL-LIT
		0200 ADDRESS
		0201 ADVANCING
		0202 AFTER
		0203 ALL
		0204 ALPHABETIC
		0205 ALPHANUMERIC
		0206 ANY
		0207 AND
		0208 ALPHANUMERIC-EDITED
		0209 BEFORE
		0210 BEGINNING
		0211 FUNCTION
		0212 CONTENT
		0213 CORR
		CORRESPONDING
		0214 DAY
		0215 DATE
		0216 DEBUG-CONTENTS
		0217 DEBUG-ITEM
		0218 DEBUG-LINE
		0219 DEBUG-NAME
		0220 DEBUG-SUB-1
		0221 DEBUG-SUB-2
		0222 DEBUG-SUB-3
		0223 DELIMITED
		0224 DELIMITER
		0225 DOWN

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0226 NUMERIC-EDITED
		0227 XML-EVENT
		0228 END-OF-PAGE
		EOP
		0229 EQUAL
		0230 ERROR
		0231 XML-NTEXT
		0232 EXCEPTION
		0233 EXTEND
		0234 FIRST
		0235 FROM
		0236 GIVING
		0237 GREATER
		0238 I-O
		0239 IN
		0240 INITIAL
		0241 INTO
		0242 INVALID
		0243 SQL
		0244 LESS
		0245 LINAGE-COUNTER
		0246 XML-TEXT
		0247 LOCK
		0248 GENERATE
		0249 NEGATIVE
		0250 NEXT
		0251 NO
		0252 NOT
		0253 NUMERIC
		0254 KANJI
		0255 OR
		0256 OTHER
		0257 OVERFLOW
		0258 PAGE
		0259 CONVERTING

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0260 POINTER
		0261 POSITIVE
		0262 DBCS
		0263 PROCEDURES
		0264 PROCEED
		0265 REFERENCES
		0266 DAY-OF-WEEK
		0267 REMAINDER
		0268 REMOVAL
		0269 REPLACING
		0270 REVERSED
		0271 REWIND
		0272 ROUNDED
		0273 RUN
		0274 SENTENCE
		0275 STANDARD
		0276 RETURN-CODE
		SORT-CORE-SIZE
		SORT-FILE-SIZE
		SORT-MESSAGE
		SORT-MODE-SIZE
		SORT-RETURN
		TALLY
		XML-CODE
		0277 TALLYING
		0278 SUM
		0279 TEST
		0280 THAN
		0281 UNTIL
		0282 UP
		0283 UPON
		0284 VARYING
		0285 RELOAD
		0286 TRUE

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0287 THEN
		0288 RETURNING
		0289 ELSE
		0290 SELF
		0291 SUPER
		0292 WHEN-COMPILED
		0293 ENDING
		0294 FALSE
		0295 REFERENCE
		0296 NATIONAL-EDITED
		0297 COM-REG
		0298 ALPHABETIC-LOWER
		0299 ALPHABETIC-UPPER
		0301 REDEFINES
		0302 OCCURS
		0303 SYNC
		SYNCHRONIZED
		0304 MORE-LABELS
		0305 JUST
		JUSTIFIED
		0306 SHIFT-IN
		0307 BLANK
		0308 VALUE
		0309 COMP
		COMPUTATIONAL
		0310 COMP-1
		COMPUTATIONAL-1
		0311 COMP-3
		COMPUTATIONAL-3
		0312 COMP-2
		COMPUTATIONAL-2
		0313 COMP-4
		COMPUTATIONAL-4
		0314 DISPLAY-1
		0315 SHIFT-OUT

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0316 INDEX
		0317 USAGE
		0318 SIGN
		0319 LEADING
		0320 SEPARATE
		0321 INDEXED
		0322 LEFT
		0323 RIGHT
		0324 PIC
		PICTURE
		0325 VALUES
		0326 GLOBAL
		0327 EXTERNAL
		0328 BINARY
		0329 PACKED-DECIMAL
		0330 EGCS
		0331 PROCEDURE-POINTER
		0332 COMP-5
		COMPUTATIONAL-5
		0333 FUNCTION-POINTER
		0334 TYPE
		0335 JNIEVPTR
		0336 NATIONAL
		0337 GROUP-USAGE
		0342 VOLATILE
		0401 HIGH-VALUE
		HIGH-VALUES
		0402 LOW-VALUE
		LOW-VALUES
		0403 QUOTE
		QUOTES
		0404 SPACE
		SPACES
		0405 ZERO



表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0406 ZEROES
		ZEROS
		0407 NULL
		NULLS
		0501 BLOCK
		0502 BOTTOM
		0505 CHARACTER
		0506 CODE
		0507 CODE-SET
		0514 FILLER
		0516 FOOTING
		0520 LABEL
		0521 LENGTH
		0524 LINAGE
		0526 OMITTED
		0531 RENAMES
		0543 TOP
		0545 TRAILING
		0549 RECORDING
		0601 INHERITS
		0603 RECURSIVE
		0701 ACCESS
		0702 ALSO
		0703 ALTERNATE
		0704 AREA
		AREAS
		0705 ASSIGN
		0707 COLLATING
		0708 COMMA
		0709 CURRENCY
		0710 CLASS
		0711 DECIMAL-POINT
		0712 DUPLICATES
		0713 DYNAMIC
		0714 EVERY

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0716 MEMORY
		0717 MODE
		0718 MODULES
		0719 MULTIPLE
		0720 NATIVE
		0721 オフ
		0722 OPTIONAL
		0723 ORGANIZATION
		0724 POSITION
		0725 PROGRAM
		0726 RANDOM
		0727 RELATIVE
		0728 RERUN
		0729 RESERVE
		0730 SAME
		0731 SEGMENT-LIMIT
		0732 SELECT
		0733 SEQUENCE
		0734 SEQUENTIAL
		0736 SORT-MERGE
		0737 STANDARD-1
		0738 TAPE
		0739 WORDS
		0740 PROCESSING
		0741 APPLY
		0742 WRITE-ONLY
		0743 COMMON
		0744 ALPHABET
		0745 PADDING
		0746 SYMBOLIC
		0747 STANDARD-2
		0748 OVERRIDE
		0750 PASSWORD
		0751 XML-SCHEMA

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0801 ARE
		IS
		0802 ASCENDING
		0803 AT
		0804 BY
		0805 CHARACTERS
		0806 CONTAINS
		0808 COUNT
		0809 DEBUGGING
		0810 DEPENDING
		0811 DESCENDING
		0812 DIVISION
		0814 FOR
		0815 ORDER
		0816 INPUT
		0817 REPLACE
		0818 KEY
		0819 LINE
		LINES
		0820 XML-INFORMATION
		0821 OF
		0822 ON
		0823 OUTPUT
		0825 RECORD
		0826 RECORDS
		0827 REEL
		0828 SECTION
		0829 SIZE
		0830 STATUS
		0831 THROUGH
		THRU
		0832 TIME
		0833 TIMES
		0834 TO
		0836 UNIT

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		0837 USING
		0838 WHEN
		0839 WITH
		0840 SQLIMS
		0841 DEFAULT
		0901 PROCEDURE
		0902 DECLARATIVES
		0903 END
		1001 DATA
		1002 FILE
		1003 FD
		1004 SD
		1005 WORKING-STORAGE
		1006 LOCAL-STORAGE
		1007 LINKAGE
		1101 ENVIRONMENT
		1102 CONFIGURATION
		1103 SOURCE-COMPUTER
		1104 OBJECT-COMPUTER
		1105 SPECIAL-NAMES
		1106 REPOSITORY
		1107 INPUT-OUTPUT
		1108 FILE-CONTROL
		1109 I-O-CONTROL
		1201 ID
		IDENTIFICATION
		1202 PROGRAM-ID
		1203 AUTHOR
		1204 INSTALLATION
		1205 DATE-WRITTEN
		1206 DATE-COMPILED
		1207 SECURITY
		1208 CLASS-ID
		1209 METHOD-ID
		1210 METHOD
		1211 FACTORY

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
		<b>1212</b> OBJECT <b>2020</b> TRACE <b>2046</b> SUPPRESS <b>3000</b> DATADEF <b>3001</b> F-NAME <b>3002</b> UPSI-SWITCH <b>3003</b> CONDNAME <b>3004</b> CONDDVAR <b>3005</b> BLOB <b>3006</b> CLOB <b>3007</b> DBCLOB <b>3008</b> BLOB-LOCATOR <b>3009</b> CLOB-LOCATOR <b>3010</b> DBCLOB-LOCATOR <b>3011</b> BLOB-FILE <b>3012</b> CLOB-FILE <b>3013</b> DBCLOB-FILE <b>3014</b> DFHRESP <b>5001</b> PARSE <b>5002</b> AUTOMATIC <b>5003</b> PREVIOUS <b>5004</b> ENCODING <b>5005</b> NAMESPACE <b>5006</b> NAMESPACE-PREFIX <b>5007</b> XML-DECLARATION <b>5008</b> ATTRIBUTES <b>5009</b> VALIDATING <b>5010</b> UNBOUNDED <b>5011</b> ATTRIBUTE <b>5012</b> ELEMENT <b>5013</b> NONNUMERIC <b>5014</b> NAME <b>5015</b> CYCLE <b>5016</b> PARAGRAPH <b>5020</b> AS <b>5021</b> INITIALIZED <b>9999</b> COBOL
トークン長	HL2	トークンの長さ
トークン列	FL4	ソース・リスト内のトークンの開始列番号
トークン行	FL4	ソース・リスト内のトークンの行番号

表 125. SYSADATA トークン・レコード (続き)

フィールド	サイズ	説明
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。  <b>X'80'</b> トークンは継続  <b>X'40'</b> 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。
予約済み	CL7	将来の利用のために予約済み
トークン・テキスト	CL(n)	実トークン・String

## ソース・エラー・レコード: X'0032'

次の表に、ソース・エラー・レコードの内容を示します。

表 126. SYSADATA ソース・エラー・レコード

フィールド	サイズ	説明
ステートメント番号	FL4	エラーのあるステートメントのステートメント番号
エラー ID	CL16	エラー・メッセージ ID (左寄せ、ブランク埋め込み)
エラーの重大度	HL2	エラーの重大度
エラー・メッセージ長	HL2	エラー・メッセージ・テキストの長さ
行位置	XL1	FIPS メッセージに指定される行位置標識
予約済み	CL7	将来の利用のために予約済み
エラー・メッセージ	CL(n)	エラー・メッセージ・テキスト

## ソース・レコード: X'0038'

次の表に、ソース・レコードの内容を示します。

表 127. 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 1 次入力ファイルからのものである場合は、入力ファイルの割り当て済みシーケンス番号。(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済みシーケンス番号。(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。

表 127. SYSADATA ソース・レコード (続き)

フィールド	サイズ	説明
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ものである場合は、親ファイルの割り当て済みシーケンス番号。(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済みシーケンス番号。(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。)
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( <i>n</i> )	

## COPY REPLACING レコード: X'0039'

REPLACING アクションが実行されるごとに、1 つの COPY REPLACING タイプのレコードが出力されます。すなわち、REPLACING 句の *operand-1* がコピーブックのテキストと一致するごとに、COPY REPLACING TEXT レコードが書き込まれます。

次の表に、COPY REPLACING レコードの内容を示します。

表 128. 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 129. SYSADATA 記号レコード

フィールド	サイズ	説明
記号 ID	FL4	固有の記号 ID
行番号	FL4	記号が定義または宣言されるソース・レコードのリスト行番号
レベル	XL1	記号の真のレベル番号 (または構造内のデータ項目の相対レベル番号)。 COBOL の場合、これは、01 から 49 の範囲、66 (RENAMES 項目の場合)、77、または 88 (条件項目の場合) になります。
修飾標識	XL1	<b>X'00'</b> 固有の名前。修飾は不要。 <b>X'01'</b> 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	<b>X'68'</b> クラス名 (クラス ID) <b>X'58'</b> メソッド名 <b>X'40'</b> データ名 <b>X'20'</b> プロシージャ名。 <b>X'10'</b> 簡略名 <b>X'08'</b> プログラム名 <b>X'81'</b> 予約済み  以下は、上記タイプの ORed です (該当する場合)。 <b>X'04'</b> 外部 <b>X'02'</b> グローバル



表 129. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
記号属性	XL1	<p><b>X'01'</b> 数字</p> <p><b>X'02'</b> 次のいずれかのクラスの基本文字:</p> <ul style="list-style-type: none"> <li>• 英字</li> <li>• 英数字</li> <li>• DBCS</li> <li>• 国別</li> </ul> <p><b>X'03'</b> グループ</p> <p><b>X'04'</b> ポインター</p> <p><b>X'05'</b> 指標データ項目</p> <p><b>X'06'</b> 指標名</p> <p><b>X'07'</b> 条件</p> <p><b>X'0F'</b> ファイル</p> <p><b>X'10'</b> ソート・ファイル</p> <p><b>X'17'</b> クラス名 (リポジトリ)</p> <p><b>X'18'</b> オブジェクト参照</p> <p><b>X'19'</b> 通貨記号</p> <p><b>X'1A'</b> XML スキーマ名</p>

表 129. **SYSADATA** 記号レコード (続き)

フィールド	サイズ	説明
節	XL1	<p>記号定義で指定されている文節。</p> <p>数値 (X'01'), 基本文字 (X'02'), グループ (X'03'), ポインター (X'04'), 指標データ項目 (X'05'), またはオブジェクト参照 (X'18') の記号属性を持つ記号の場合:</p> <p><b>1... ....</b> 値</p> <p><b>.1... ....</b> 索引付き</p> <p><b>..1. ....</b> 再定義</p> <p><b>...1 ....</b> 名前変更</p> <p><b>.... 1...</b> 発生</p> <p><b>.... .1..</b> Occurs キーあり</p> <p><b>.... ..1.</b> ケースにより発生</p> <p><b>.... ...1</b> 親で発生</p> <p>ファイル・タイプの場合:</p> <p><b>1... ....</b> 選択</p> <p><b>.1... ....</b> 割り当て</p> <p><b>..1. ....</b> 再実行</p> <p><b>...1 ....</b> 同一領域</p> <p><b>.... 1...</b> 同一レコード域</p> <p><b>.... .1..</b> 記録モード</p> <p><b>.... ..1.</b> 予約済み</p> <p><b>.... ...1</b> レコード</p>

表 129. 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:
		<b>01</b> CSP
		<b>02</b> C01
		<b>03</b> C02
		<b>04</b> C03
		<b>05</b> C04
		<b>06</b> C05
		<b>07</b> C06
		<b>08</b> C07
		<b>09</b> C08
		<b>10</b> C09
		<b>11</b> C10
		<b>12</b> C11
		<b>13</b> C12
		<b>14</b> S01
		<b>15</b> S02
		<b>16</b> S03
		<b>17</b> S04
		<b>18</b> S05
		<b>19</b> CONSOLE
		<b>20</b> SYSIN   SYSIPT
		<b>22</b> SYSOUT   SYSLST   SYSLIST
		<b>24</b> SYSPUNCH   SYSPCH
		<b>26</b> UPSI-0
		<b>27</b> UPSI-1
		<b>28</b> UPSI-2
		<b>29</b> UPSI-3
		<b>30</b> UPSI-4
		<b>31</b> UPSI-5
		<b>32</b> UPSI-6
		<b>33</b> UPSI-7
		<b>34</b> AFP-5A

表 129. **SYSADATA** 記号レコード (続き)

フィールド	サイズ	説明
データ・フラグ 1	XL1	<p>ファイル・タイプの場合、および数値 (X'01'), 基本文字 (X'02'), グループ (X'03'), ポインター (X'04'), 指標データ項目 (X'05'), またはオブジェクト参照 (X'18') の記号属性を持つ記号の場合:</p> <p><b>1... ....</b> 再定義</p> <p><b>.1... ....</b> 名前変更</p> <p><b>..1. ....</b> 同期化</p> <p><b>...1 ....</b> 暗黙的に再定義</p> <p><b>.... 1...</b> 揮発性</p> <p><b>.... .1..</b> 暗黙の再定義</p> <p><b>.... ..1.</b> FILLER</p> <p><b>.... ...1</b> レベル 77</p>

表 129. 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	<p>数値 (X'01') の記号属性を持つ記号の場合:</p> <p>1... .... 2 進数</p> <p>.1... .... 外部浮動小数点 (USAGE DISPLAY または USAGE NATIONAL の)</p> <p>..1. .... 内部浮動小数点</p> <p>...1 .... 圧縮</p> <p>.... 1... 外部 10 進数 (USAGE DISPLAY または USAGE NATIONAL の)</p> <p>.... .1.. 負の位取り</p> <p>.... ..1.. 数字編集 (USAGE DISPLAY または USAGE NATIONAL の)</p> <p>.... ...1 将来の利用のために予約済み</p> <p>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:</p> <p>1... .... 英字</p> <p>.1... .... 英数字</p> <p>..1. .... 編集英数字</p> <p>...1 .... グループは独自の ODO オブジェクトを含む</p> <p>.... 1... DBCS 項目</p> <p>.... .1.. グループ可変長</p> <p>.... ..1.. EGCS 項目</p> <p>.... ...1 編集 EGCS</p>

表 129. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
		<p>ファイル・タイプの場合:</p> <p><b>1... ....</b> レコード内の ODO のオブジェクト</p> <p><b>.1.. ....</b> レコード内の ODO のサブジェクト</p> <p><b>..1. ....</b> 順次アクセス</p> <p><b>...1 ....</b> ランダム・アクセス</p> <p><b>.... 1...</b> 動的アクセス</p> <p><b>.... .1..</b> 位置指定モード</p> <p><b>.... ..1.</b> レコード域</p> <p><b>.... ...1</b> 将来の利用のために予約済み</p>
データ・フラグ 3	XL1	<p><b>1... ....</b> すべてのレコードが同じ長さ</p> <p><b>.1.. ....</b> 固定長</p> <p><b>..1. ....</b> 可変長</p> <p><b>...1 ....</b> 未定義</p> <p><b>.... 1...</b> スパン</p> <p><b>.... .1..</b> ブロック</p> <p><b>.... ..1.</b> 書き込みのみ適用</p> <p><b>.... ...1</b> 同一ソート・マージ領域</p>

表 129. 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	<p><b>1...</b> .... 物理順次 (ホスト上、QSAM)</p> <p><b>.1..</b> .... ASCII</p> <p><b>..1.</b> .... 標準ラベル</p> <p><b>...1</b> .... ユーザー・ラベル</p> <p><b>.... 1...</b> 順次編成</p> <p><b>.... .1..</b> 索引編成</p> <p><b>.... ..1.</b> 相対編成</p> <p><b>.... ...1</b> 行順次</p>
USAGE 節	FL1	<p><b>X'00'</b> USAGE IS DISPLAY</p> <p><b>X'01'</b> USAGE IS COMP-1</p> <p><b>X'02'</b> USAGE IS COMP-2</p> <p><b>X'03'</b> USAGE IS PACKED-DECIMAL または USAGE IS COMP-3</p> <p><b>X'04'</b> USAGE IS BINARY、USAGE IS COMP、USAGE IS COMP-4、または USAGE IS COMP-5</p> <p><b>X'05'</b> USAGE IS DISPLAY-1</p> <p><b>X'06'</b> USAGE IS POINTER</p> <p><b>X'07'</b> USAGE IS INDEX</p> <p><b>X'08'</b> USAGE IS PROCEDURE-POINTER</p> <p><b>X'09'</b> USAGE IS OBJECT-REFERENCE</p> <p><b>X'0A'</b> FUNCTION-POINTER</p> <p><b>X'0B'</b> NATIONAL</p>
記号節	FL1	<p><b>X'00'</b> SIGN 節なし</p> <p><b>X'01'</b> SIGN IS LEADING</p> <p><b>X'02'</b> SIGN IS LEADING SEPARATE CHARACTER</p> <p><b>X'03'</b> SIGN IS TRAILING</p> <p><b>X'04'</b> SIGN IS TRAILING SEPARATE CHARACTER</p>
標識	FL1	<p><b>X'01'</b> JUSTIFIED 節あり。右寄せ属性が有効。</p> <p><b>X'02'</b> BLANK WHEN ZERO 節あり。</p>

表 129. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
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。
ストレージ・タイプ	FL1	<p>00 該当なし</p> <p>01 ファイル</p> <p>02 作業用ストレージ</p> <p>03 リンケージ・セクション</p> <p>05 特殊レジスター</p> <p>07 変数別索引付き</p> <p>10 UPSI スイッチ</p> <p>13 可変位置項目</p> <p>14 EXTERNAL データ</p> <p>15 英数字 FUNC</p> <p>16 英数字 EVAL</p> <p>17 オブジェクト・データ</p> <p>19 ローカル・ストレージ</p> <p>20 ファクトリー・データ</p> <p>21 XML-TEXT および XML-NTEXT</p>
日付形式	FL1	将来の利用のために予約済み
データ・フラグ 4	XL1	<p>数値 (X'01') の記号属性を持つ記号の場合:</p> <p>1... .... 数値国別</p> <p>基本文字 (X'02') の記号属性を持つ記号の場合:</p> <p>1... .... 国別</p> <p>.1... .... 国別編集</p> <p>グループ (X'03') の記号属性を持つ記号の場合:</p> <p>1... .... 国別グループ (Group-Usage National)</p> <p>.1... .... 無制限長グループ</p>



表 129. 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 5	XL1	OCCURS フラグ:  1... .. UNBOUNDED
ベース・ロケーター・セル	FL2	ベース・ロケーター・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	FL4	OCCURS の最小値
段落 ID		段落名の PROC 名 ID
OCCURS 最大	FL4	OCCURS の最大値
セクション ID		セクション名の PROC 名 ID
寸法	FL4	寸法の数
大/小文字ビット・ベクトル	XL4	シンボル名の文字の大/小文字は、1 文字につき 1 ビットで表現されます。それぞれのビットには次の意味があります。  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。
予約済み	CL8	将来の利用のために予約済み
値の組のカウント	HL2	値の組のカウント
記号名の長さ	HL2	記号名の文字数

表 129. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
データ名のピクチャー・データ長  または  ファイル名の割り当て名の長さ	HL2	ピクチャー・データ内の文字の数: 関連した PICTURE 節を記号が持たない場合はゼロ。(PICTURE フィールドの長さ。) 長さは、ソース入力内で検出されたときのフィールドを表します。この長さは、複製係数を含んでいる PICTURE 項目の拡張フィールドを表すものではありません。 PICTURE スtringの最大 COBOL 長は、50 バイトです。このフィールドがゼロの場合は、PICTURE が指定されていないことを示します。  これがファイル名の場合は、外部ファイル名の文字数。これは、割り当て名の DD 名部分です。ファイル名および ASSIGN USING が指定されている場合は、ゼロ。
データ名の初期値の長さ  CLASS-ID の外部クラス名の長さ	HL2	記号値の文字の数。記号に初期値がない場合はゼロ。  CLASS-ID の外部クラス名の文字数
データ名の ODO 記号名 ID  ファイル名の場合、ASSIGN データ名の ID	FL4	データ名の場合、ODO 記号名の ID。ODO が指定されていない場合はゼロ。  ファイル名の場合、ASSIGN USING データ名の記号 ID。ASSIGN TO が指定されている場合はゼロ。
キー・カウント	HL2	定義されたキーの数
索引カウント	HL2	索引記号 ID のカウント。何も指定しない場合はゼロ
記号名	CL(n)	
データ名のピクチャー・データ・String  または  ファイル名の割り当て名	CL(n)	ユーザーが入力したとおりの PICTURE 文字String。文字Stringには、全記号、括弧、およびレプリカの生成係数が含まれます。  これがファイル名の場合は外部ファイル名。これは、割り当て名の DD 名部分です。
索引 ID リスト	(n)FL4	各索引記号名の ID
キー	(n)XL8	このフィールドには、配列に対して指定するキーを記述するデータが含まれます。以下の 3 つのフィールドは、「キー・カウント」フィールドに指定されている回数だけ繰り返されます。
...キー・シーケンス	FL1	昇順または降順の標識。  X'00'    DESCENDING  X'01'    ASCENDING
...充てん文字	CL3	予約済み
...キー ID	FL4	配列内のキー・フィールドであるデータ項目の記号 ID

表 129. SYSADATA 記号レコード (続き)

フィールド	サイズ	説明
データ名の初期値データ  CLASS-ID の外部クラス名	CL(n)	このフィールドには、この記号の INITIAL VALUE 節で指定するデータが含まれます。以下の 4 つの適切なサブフィールドは、「値の組カウント」フィールド内のカウントに応じて繰り返されます。このフィールドのデータの全長は、「初期値の長さ」フィールドに入れます。  CLASS-ID の外部クラス名。
...第 1 値の長さ	HL2	最初の値の長さ
...第 1 値のデータ	CL(n)	最初の値。  このフィールドには、ソース・ファイルの VALUE 節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始/終了区切り文字、組み込み引用符、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、そのワードに関連付けられた値ではなく、実際の予約語が入ります。
...第 2 値の長さ	HL2	2 番目の値の長さ — THRU 値の組でない場合は、ゼロ
...第 2 値のデータ	CL(n)	2 番目の値。  このフィールドには、ソース・ファイルの VALUE 節に指定されたとおりに、リテラル (または形象定数) が入ります。このフィールドには、開始/終了区切り文字、組み込み引用符、および SHIFT IN/SHIFT OUT 文字が入ります。リテラルが複数行にわたる場合には、行は 1 つの長いストリングに連結されます。形象定数が指定された場合には、このフィールドには、そのワードに関連付けられた値ではなく、実際の予約語が入ります。

## 記号相互参照レコード: X'0044'

次の表に、記号相互参照レコードの内容を示します。

表 130. SYSADATA 記号相互参照レコード

フィールド	サイズ	説明
記号長	HL2	記号の長さ
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号  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 <sup>1</sup>	HL2	このレコード内の、後に続く記号への参照の数

表 130. SYSADATA 記号相互参照レコード (続き)

フィールド	サイズ	説明
相互参照タイプ	XL1	<b>X'01'</b> プログラム <b>X'02'</b> プロシージャー <b>X'03'</b> ステートメント <b>X'04'</b> 記号またはデータ名 <b>X'05'</b> メソッド <b>X'06'</b> 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: <b>C' '</b> ブランクは参照のみの意味 <b>C'M'</b> 修正参照フラグ  プロシージャー型記号参照の場合: <b>C'A'</b> ALTER (procedure-name) <b>C'D'</b> GO TO (procedure-name) DEPENDING ON <b>C'E'</b> (PERFORM) から (procedure-name) までの範囲の 終わり <b>C'G'</b> GO TO (procedure-name) <b>C'P'</b> PERFORM (procedure-name) <b>C'T'</b> (ALTER) TO PROCEED TO (procedure-name) <b>C'U'</b> デバッグ用に使用 (procedure-name)
...ステートメント番号	XL4	記号またはステートメントが参照されているステートメント番号
<p>1. 参照フラグ・フィールドおよびステートメント番号フィールドは、参照フィールドで指示される回数分だけ、発生します。例えば、参照フィールドの数に 10 の値が入っているとき、データ名、プロシージャー、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。</p> <p>参照数が、SYSADATA ファイルのレコード・サイズを超える場合、レコードは次のレコードに続行されます。レコードの共通ヘッダー・セクションに、継続フラグが設定されます。</p>		

## ネストされたプログラム・レコード: X'0046'

次の表に、ネストされたプログラム・レコードの内容を示します。

表 131. SYSADATA ネストされたプログラム・レコード

フィールド	サイズ	説明
ステートメント定義	FL4	記号が定義または宣言されているステートメント番号
ネスト・レベル	XL1	プログラム・ネスト・レベル

表 131. SYSADATA ネストされたプログラム・レコード (続き)

フィールド	サイズ	説明
プログラム属性	XL1	<b>1... ..</b> 初期  <b>.1... ..</b> 共通  <b>...1. ....</b> PROCEDURE DIVISION using  <b>...1 1111</b> 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 132. 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 <sup>1</sup>	HL2	このレコードに記述される COPY/INCLUDE コード・メンバーの数のカウント
ライブラリー名長	HL2	ライブラリー名の長さ
ライブラリー・ボリューム長	HL2	ライブラリー・ボリューム ID の長さ
連結番号	XL2	ライブラリーの連結番号
ライブラリー DD 名長	HL2	ライブラリー DD 名の長さ
予約済み	CL4	将来の利用のために予約済み
ライブラリー名	CL(n)	COPY/INCLUDE メンバーが取り出された元のライブラリーの名前
ライブラリー・ボリューム	CL(n)	ライブラリーが常駐するボリュームのボリューム識別
ライブラリー DD 名	CL(n)	このライブラリーに使用される DD 名 (または同義)
...COPY/BASIS メンバー・ファイル ID <sup>2</sup>	HL2	... の後の名前のライブラリー・ファイル ID
...COPY/BASIS 名長	HL2	... の後の名前の長さ
...COPY/BASIS 名	CL(n)	使用されてきた COPY/BASIS メンバーの名前

表 132. SYSADATA ライブラリー・レコード (続き)

フィールド	サイズ	説明
1. ライブラリーから COPY メンバーが 10 メンバー取り出された場合には、「メンバーの数」フィールドに 10 が入り、「COPY/BASIS メンバー・ファイル ID」フィールド、「COPY/BASIS 名長」フィールド、および「COPY/BASIS 名」フィールドのオカレンスが 10 回出現します。		
2. COPY/BASIS メンバーが別のライブラリーから取り出された場合には、それぞれの一意のライブラリーごとに、ライブラリー・レコードは SYSADATA ファイルに書き込まれます。		

## 統計レコード: X'0090'

次の表に、統計レコードの内容を示します。

表 133. 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	<b>1... ....</b> ジョブ終了標識  <b>.1... ....</b> クラス定義標識  <b>..11 1111</b> 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 ファイルに含められます。

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 134. **SYSADATA EVENTS TIMESTAMP** レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ TIMESTAMP のレコード	CL12	C'TIMESTAMP'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
日付	XL8	YYYYMMDD
時間	XL2	HH
分	XL2	MI
秒	XL2	SS

表 135. **SYSADATA EVENTS PROCESSOR** レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ PROCESSOR のレコード	CL9	C'PROCESSOR'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
出力ファイル ID	XL1	
ブランク・セパレータ ー	CL1	
行クラス標識	XL1	

表 136. **SYSADATA EVENTS FILE END** レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)

表 136. SYSADATA EVENTS FILE END レコードのレイアウト (続き)

フィールド	サイズ	説明
EVENTS レコード・ タイプ FILE END の レコード	CL7	C'FILEEND'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
入力ファイル ID	XL1	
ブランク・セパレータ ー	CL1	
拡張標識	XL1	

表 137. SYSADATA EVENTS PROGRAM レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ PROGRAM のレコード	CL7	C'PROGRAM'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	
ブランク・セパレータ ー	CL1	
出力ファイル ID	XL1	
ブランク・セパレータ ー	CL1	
プログラム入力レコー ド番号	XL1	

表 138. SYSADATA EVENTS FILE ID レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・ タイプ FILE ID のレ コード	CL7	C'FILEID'
ブランク・セパレータ ー	CL1	
改訂レベル	XL1	



表 138. SYSADATA EVENTS FILE ID レコードのレイアウト (続き)

フィールド	サイズ	説明
ブランク・セパレーター	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	
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 139. SYSADATA EVENTS ERROR レコードのレイアウト

フィールド	サイズ	説明
ヘッダー	CL12	標準 ADATA レコード・ヘッダー
レコード長	HL2	続く EVENTS レコード・データの長さ (このハーフワードは除く)
EVENTS レコード・タイプ ERROR のレコード	CL5	C'ERROR'
ブランク・セパレーター	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	

表 139. SYSADATA EVENTS ERROR レコードのレイアウト (続き)

フィールド	サイズ	説明
エラー・トークン開始 番号	XL1	エラー・トークン開始の桁番号
ブランク・セパレータ ー	CL1	
エラー終了行番号	XL10	
ブランク・セパレータ ー	CL1	
エラー・トークン終了 番号	XL1	エラー・トークン終了の桁番号
ブランク・セパレータ ー	CL1	
エラー・メッセージの ID 番号	XL9	
ブランク・セパレータ ー	CL1	
エラー・メッセージの 重大度コード	XL1	
ブランク・セパレータ ー	CL1	
エラー・メッセージの 重大度レベル番号	XL2	
ブランク・セパレータ ー	CL1	
エラー・メッセージ長	HL3	
ブランク・セパレータ ー	CL1	
エラー・メッセージ・ テキスト	CL(n)	

---

## 付録 H. 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、言語環境プログラム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

『IGYTCARA: バッチ・アプリケーション』

945 ページの『IGYTCARB: 対話式プログラム』

948 ページの『IGYTSALE: ネストされたプログラム・アプリケーション』

---

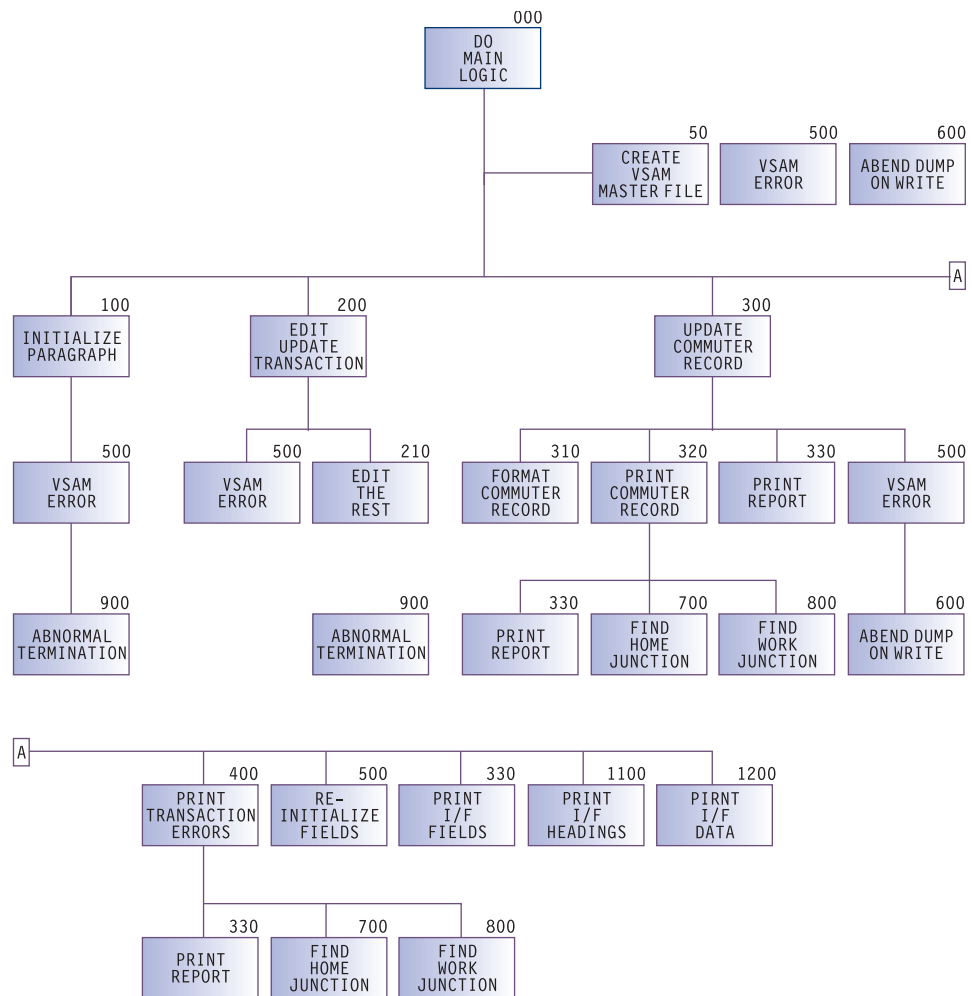
## IGYTCARA: バッチ・アプリケーション

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・ファイル記入項目を妥当性検査し (QSAM 順次ファイル処理)、マスター・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

944 ページの『IGYTCARA の実行準備』

#### 関連参照

『IGYTCARA の入力データ』

943 ページの『IGYTCARA によって作成される報告書』

956 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、会社は関与する従業員から情報を収集し、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
5. 通勤者名
6. 自宅の住所
7. 自宅の電話番号
8. 勤務先電話
9. 居住地域コード
10. 勤務地域コード
11. 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 PAGE #: 1
-PROGRAM #: IGYTCAR1 RUN TIME: 01:40 RUN DATE: 11/24/2003

TRANS CORD HOME CODE COMMUTER HOME HOME PHONE HOME LOCATION JUNCTION STA-
CODE TYPE WORK CODE NAME ADDRESS ADDRESS WORK PHONE WORK LOCATION JUNCTION TUS

A NEW 1 01 11 ROBERTS AB 1021 CRYSTAL COURT (415) 555-0190 RODNEY/CRYSTAL D
SAN FRANCISCO CA 99901 (415) 555-1387 BAYFAIR PLAZA

A NEW 2 02 12 KAHN DE 789 EMILY LANE (415) 555-1890 COYOTE D
SAN FRANCISCO CA 99921 (415) 555-2589 14TH STREET/166TH AVENUE

P 4 88 99 (000) 555-7890 HOME CODE ' ' NOT FOUND. T
99 ASDFG (123) ASD-FGHJ WORK CODE ' ' NOT FOUND.

TRANSACT. 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 (415) 555-0190 RODNEY/CRYSTAL D
SAN FRANCISCO CA 99901 (415) 555-1387 BAYFAIR PLAZA
NEW 1 01 11 ROBERTS AB 1221 CRYSTAL COURT (415) 555-0190 RODNEY/CRYSTAL D

```

SAN FRANCISCO CA 99901 (415) 555-1387 BAYFAIR PLAZA									
A		2	02	12	KAHN	DE 789 EMILY LANE SAN FRANCISCO	CA 99921	(415) 555-1890 COYOTE (415) 555-2589 14TH STREET/166TH AVENUE	D DUPLICATE REC.
D	OLD	2	02	12	KAHN	DE 789 EMILY LANE SAN FRANCISCO	CA 99921	(415) 555-1890 COYOTE (415) 555-2589 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	CA 99921	(415) 555-1890 COYOTE (415) 555-2589 14TH STREET/166TH AVENUE	D
A	NEW	1	01	11	BONNICK	FD 1025 FIFTH AVENUE SAN FRANCISCO	CA 99905	(415) 555-9590 RODNEY (415) 555-7895 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.V6R2M0.SIGYSAMP データ・セット内にあります。

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

941 ページの『IGYTCARA: バッチ・アプリケーション』

### 関連タスク

『IGYTCARA の実行』

### 関連参照

942 ページの『IGYTCARA の入力データ』

943 ページの『IGYTCARA によって作成される報告書』

956 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

以下の手順で、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。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。

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。

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.V6R2M0.SIGYSAMP,DISP=SHR
| //COBOL.SYSIN DD DSN=IGY.V6R2M0.SIGYSAMP(IGYTCARA),DISP=SHR
| //GO.SYSOUT DD SYSOUT=A
| //GO.COMMUTR DD DSN=your-prefix.IGYTCAR.MASTFILE,DISP=SHR
| //GO.LOCCODE DD DSN=IGY.V6R2M0.SIGYSAMP(IGYTCODE),DISP=SHR
| //GO.UPDTRANS DD DSN=IGY.V6R2M0.SIGYSAMP(IGYTRANX),DISP=SHR
| //GO.UPDPRINT DD SYSOUT=A,DCB=BLKSIZE=133
| //
```

関連タスク

209 ページの『第 10 章 VSAM ファイルの処理』

関連参照

292 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)』

---

## 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

『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) 下で 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.V6R2M0.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。

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

945 ページの『IGYTCARB: 対話式プログラム』

関連タスク

947 ページの『IGYTCARB の実行』

関連参照

956 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。

z/OS で IGYTCARB を実行するには、次の手順を行います。

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 ¬sym.= '.')
&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(ISRPCC)'
7, 'PGM(ISRYXDR) NOCHECK'
8, 'PANEL(ISRLPRIM)'
C, 'PGM(IGYTCARB)' (2)
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. IGYTCARB 内のシーケンス行 IB2200 にコメントを加え、シーケンス行 IB2210 のコメントを外します。(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 の LOADLIB のライブラリー名です。DSN2 は、インストールされた 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.V6R2M0.SIGYSAMP(IGYTRANB)') SHR REUSE
```

7. 次のコマンドを使用して、ISPLLIB を割り振ります。

```
ALLOCATE FILE(ISPLLIB) 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。

#### 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

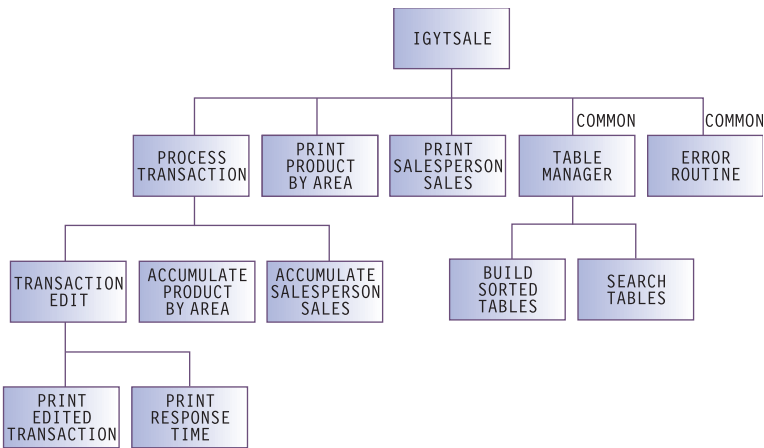
アプリケーション IGYTSALE は、スポーツ用品販売店の商品売上と売上手数料を追跡します。

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。

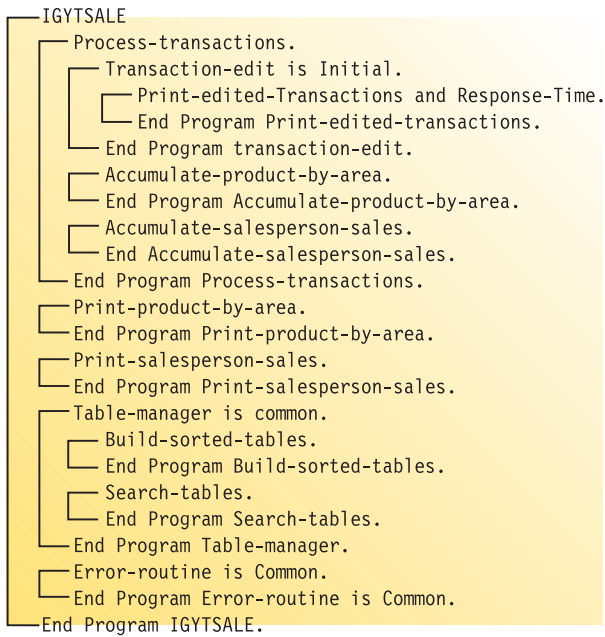
1. 製品種目、顧客、および販売員の人数を記録します。このデータは、IGYTABLE と呼ばれるファイルに保管されます。

- 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。
- 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。
- 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。
- 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

955 ページの『IGYTSale の実行準備』

## 関連参照

『IGYTSALE の入力データ』

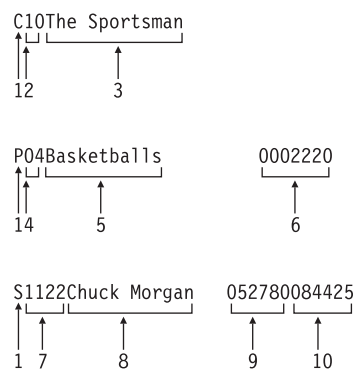
952 ページの『IGYTSALE によって作成される報告書』

956 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	8
↑	↑
9	8
↑	↑
9	8

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。

『例: IGYTSALE のトランザクション・エラー』  
『例: IGYTSALE の製品別および区域別の販売分析』  
953 ページの『例: IGYTSALE の販売と歩合』  
954 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 Order	Inv. Items	Sales Time Stamp	Sales Area	Sales Pers	Cust. Code	Product And Quantity Sold	Ship Date Stamp	
-----								
	4	19990324591515	SAN DIEGO	116	15	60200132200110522045100		Error Descriptions -Sales order number is missing -Date of sale time stamp is invalid -Salesperson number not numeric -Product code not in product-table -Date of ship time stamp is 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 -Invoiced items is invalid -Product and quantity not checked -Date of ship time stamp is invalid
B11124	2	19901313510000	SAN DIEGO	1166	11	1 0200042200120a22141100		Error Descriptions -Date of sale time stamp is invalid -Product code is invalid -Date of ship time stamp is invalid
133	81119110000	LOS ANGELES	1166	10	040112110210160321251104			Error Descriptions -Sales order number is invalid -Invoiced items is invalid -Date of sale time stamp is invalid -Product and quantity not checked -Date of ship time stamp is invalid
C11133	4	1990111944		1166	10	040112110210160321251104		Error Descriptions -Date of sale time stamp is invalid -Sales area is missing -Date of ship time stamp is invalid
C11138	4	19901117091530	LOS ANGELES	1155		113200102010260321250004	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								
Product Codes	CNTRL COAST	CNTRL VALLEY	LOS ANGELES	NORTH COAST	SAN DIEGO	SF BAY AREA	Product Totals	
=====								
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		9900	2120	11	2700		14731
Unit Price		88.30	88.30	88.30	88.30		
Amount of Sale		\$874,170.00	\$187,196.00	\$971.30	\$238,410.00		\$1,300,747.30
Product Number 06							
Basketball Uniform							
Units Sold				990	200	200	1390
Unit Price				42.20	42.20	42.20	
Amount of Sale				\$41,778.00	\$8,440.00	\$8,440.00	\$58,658.00
Product Number 10							
Baseball Cage							
Units Sold	45		3450	16	200	3320	7031
Unit Price	890.00		890.00	890.00	890.00	890.00	
Amount of Sale	\$40,050.00		\$3,070,500.00	\$14,240.00	\$178,000.00	\$2,954,800.00	\$6,257,590.00
Product Number 11							
Baseball Uniform							
Units Sold	10003		3578		2922	2746	19249
Unit Price	45.70		45.70		45.70	45.70	
Amount of Sale	\$457,137.10		\$163,514.60		\$133,535.40	\$125,492.20	\$879,679.30
Product Number 12							
Softballs							
Units Sold	10	137	2564	13	2200	22	4946
Unit Price	1.40	1.40	1.40	1.40	1.40	1.40	
Amount of Sale	\$14.00	\$191.80	\$3,589.60	\$18.20	\$3,080.00	\$30.80	\$6,924.40
Product Number 13							
Softball Bats							
Units Sold	3227		3300	1998	5444	99	14068
Unit Price	12.60		12.60	12.60	12.60	12.60	
Amount of Sale	\$40,660.20		\$41,580.00	\$25,174.80	\$68,594.40	\$1,247.40	\$177,256.80
Product Number 14							
Softball Gloves							
Units Sold	1155		136	3119	3833	5152	13395
Unit Price	12.00		12.00	12.00	12.00	12.00	
Amount of Sale	\$13,860.00		\$1,632.00	\$37,428.00	\$45,996.00	\$61,824.00	\$160,740.00
Product Number 15							
Softball Cage							
Units Sold	997	99	2000		2400		5496
Unit Price	890.00	890.00	890.00		890.00		
Amount of Sale	\$887,330.00	\$88,110.00	\$1,780,000.00		\$2,136,000.00		\$4,891,440.00
Product Number 16							
Softball Uniform							
Units Sold	44		465	16	6165	200	6890
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	21823
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	2052
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 *	119210 *
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
Salesperson: Billy Jim Bob		Sales and Commission Report					
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 Orders	Products Ordered	Total for Order	Discount (if any)	Discount Amount	Commission 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.V6R2M0.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。

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

948 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

『IGYTSALE の実行』

関連参照

- 950 ページの『IGYTSALE の入力データ』
- 952 ページの『IGYTSALE によって作成される報告書』
- 956 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。

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.V6R2M0.SIGYSAMP,DISP=SHR
| //COBOL.SYSIN DD DSN=IGY.V6R2M0.SIGYSAMP(IGYTSALE),DISP=SHR
| //GO.SYSOUT DD SYSOUT=A
| //GO.IGYTABLE DD DSN=IGY.V6R2M0.SIGYSAMP(IGYTABLE),DISP=SHR
| //GO.IGYTRANS DD DSN=IGY.V6R2M0.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

言語エレメントまたは概念	シーケンス・ストリング
呼び出し可能サービス (言語環境プログラム):	
1. CEEDATM: 日付および時刻出力の形式設定	1. IS0875, IS2575
2. CEEDCOD: フィードバック・コード検査	2. IS0905
3. CEEGMTO: 地方時からの UTC オフセット	3. IS0904
4. CEELOCT: ローカル日付および時刻	4. IS0850
5. CEESECS: タイム・スタンプから秒への変換	5. IS2350, IS2550
ファイルの CLOSE	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
Do-until (PERFORM . . . TEST AFTER)	IA4900-5010, IA7690-7770
Do-while (PERFORM . . . TEST BEFORE)	IS1660
END-ADD	IS2900
END-COMPUTE	IS4510
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

言語エレメントまたは概念	シーケンス・ストリング
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
DATA DIVISION 内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
組み込み関数: 1. CURRENT-DATE 2. MAX 3. MEAN 4. MEDIAN 5. MIN 6. STANDARD-DEVIATION 7. UPPER-CASE 8. VARIANCE 9. WHEN-COMPILED	1. IA9005 2. IA9235 3. IA9215 4. IA9220 5. IA9240 6. IA9230 7. IA9015 8. IA9225 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

言語エレメントまたは概念	シーケンス・ストリング
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 (Do-until)	IA4900-5010, IA7690-7770
PERFORM . . . WITH TEST BEFORE (Do-while)	IS1660
PERFORM . . . UNTIL	IS5000
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、順次テーブル探索	IA7690-7770
SEARCH を使用した、順次テーブル探索	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
VALUE ZERO (S) (ES)	IS0600

言語エレメントまたは概念	シーケンス・ストリング
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

---

## 付録 I. 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 のアクセシビリティ機能は、Enterprise COBOL for z/OS のアクセスを支援します。

### アクセシビリティ機能

z/OS は、以下のような主要アクセシビリティ機能を備えています。

- 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、US Section 508 (<http://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards>) および Web Content Accessibility Guidelines (WCAG) 2.0 (<http://www.w3.org/TR/WCAG20/>) に確実に準拠するために、最新の W3C 標準である WAI-ARIA 1.0 (<http://www.w3.org/TR/wai-aria/>) を使用します。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 の Enterprise COBOL for z/OS オンライン製品資料は、アクセシビリティに対応しています。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 Developer for z Systems 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>)
- z/OS TSO/E User's Guide (<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>)
- IBM Developer for z Systems Knowledge Center ([http://www.ibm.com/support/knowledgecenter/SSQ2R2/rdz\\_welcome.html?lang=en](http://www.ibm.com/support/knowledgecenter/SSQ2R2/rdz_welcome.html?lang=en))

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。

## インターフェース情報

Enterprise COBOL for z/OS のオンライン製品資料は、IBM Knowledge Center で入手でき、標準の Web ブラウザーで表示できます。

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。

支援技術製品は、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技術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

標準の IBM ヘルプ・デスクとサポート Web サイトに加え、IBM は、聴覚が不自由なお客様が営業やサポート・サービスにアクセスするために使用できる TTY 電話サービスを立ち上げました。

TTY サービス

800-IBM-3383 (800-426-3383)

(北米内)

## IBM およびアクセシビリティ

IBM のアクセシビリティへの取り組みについて詳しくは、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。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒103-8510

東京都中央区日本橋箱崎町19番21号

日本アイ・ビー・エム株式会社

法務・知的財産

知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

Intellectual Property Dept. for Rational Software  
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 사용할 수 있지만、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については検証できません。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・プログラムは特定物として現存するままの状態を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・プログラムから取られています。© Copyright IBM Corp. 1991, 2019.

プライバシー・ポリシーに関する考慮事項:

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。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。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、『IBM オンラインでのプライバシー・ステートメント』(<http://www.ibm.com/privacy/details/jp/ja/>) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および「IBM Software Products and Software-as-a-Service Privacy Statement」(<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)<sup>®</sup> は、世界の多くの国で登録された International Business Machines Corporation の商標です。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。現時点での IBM の商標リストについては、[www.ibm.com/legal/copytrade.shtml](http://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。

## 用語集

この用語集に記載されている用語は、COBOL における意味に従って定義されています。これらの用語は、他の言語では同じ意味を持つことも、持たないこともあります。

この用語集には、以下の資料からの用語および定義が記載されています。

- 「ANSI INCITS 23-1985, *Programming languages - COBOL*」(「ANSI INCITS 23a-1989, *Programming Languages - COBOL - Intrinsic Function Module for COBOL*」および「ANSI INCITS 23b-1993, *Programming Languages - Correction Amendment for COBOL*」で改訂)
- 「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。

#### 異常終了 (**abend**)

プログラムの異常終了。

#### 16 MB 境界より上 (**above the 16 MB line**)

いわゆる 16 MB 境界より上で、2 GB バイより下のストレージ。このストレージは、31 ビット・モードでのみアドレス可能である。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。24 ビット・モードでコンパイルされたプログラムは、架空のストレージ境界線の下に保持されているかのように、16MB のスペースに対してのみアドレッシングが可能です。VS COBOL II 以降、31 ビット・モードでコンパイルされたプログラムは、16 MB 境界より上に配置することができます。

#### 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。

#### 実際の文書エンコード (**actual document encoding**)

XML 文書のエンコード・カテゴリーで、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 段落のユーザー定義語であり、特定の文字セットまたは照合シーケンス (あるいはその両方) に名前を割り当てるもの。

英字 (\* **alphabetic character**)  
英字またはスペース文字。

| 英数字位置 (**alphanumeric character position**)  
| 「文字位置 (*character position*)」を参照。

英字データ項目 (**alphabetic data item**)  
記号 A のみを含む PICTURE 文字ストリングが記述されたデータ項目。英字データ項目は USAGE DISPLAY を持ちます。

英数字 (\* **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 (**alphanumeric data item**)  
暗黙的または明示的に USAGE DISPLAY として記述された、カテゴリー英数字、英数字編集、または数字編集を持つデータ項目を指す一般的な呼び方。

英数字編集データ項目 (**alphanumeric-edited data item**)  
少なくとも 1 つの記号 A または X のインスタンスおよび少なくとも 1 つの単純挿入記号 B、0、または / を含んでいる、PICTURE 文字ストリングで記述されたデータ項目。英数字編集データ項目は USAGE DISPLAY を持ちます。

英数字関数 (\* **alphanumeric function**)  
コンピューターの英数字セットからの 1 つ以上の文字のストリングで値が構成されている関数。

英数字グループ項目 (**alphanumeric group item**)  
GROUP-USAGE NATIONAL 節なしで定義されたグループ項目。INSPECT、STRING、および UNSTRING などの操作の場合、英数字グループ項目は、実際のグループの内容にかかわらず、その内容すべてが USAGE DISPLAY として記述されているかのように処理されます。グループ内の基本項目を処理する必要がある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、または INITIALIZE など) の場合、英数字グループ項目はグループ・セマンティクスを使用して処理されます。

英数字リテラル (**alphanumeric literal**)  
次のセットからの開始区切り文字を有するリテラル。'、"、X'、X"、Z'、または Z"。この文字ストリングには、コンピュ

ーターの有する文字セットの任意の文字を含めることができる。

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。

米国規格協会 (**American National Standards Institute: ANSI**)  
米国で認定された組織が自発的工業規格を作成して維持する手順を設定する組織であり、製造業者、消費者、および一般の利害関係者で構成される。

引数 (**argument**)  
(1) ID、リテラル、算術式、または関数 ID で、これにより関数の評価に使用する値を指定する。(2) CALL または INVOKE ステートメントの USING 句のオペランドであり、呼び出されたプログラムまたは起動されたメソッドに値を渡すのに使用されます。

| 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**)  
次に示す集合に属する 1 文字、または 2 文字で構成された固定した組み合わせ。

文字	意味
+	加算
-	減算
*	乗算
/	除算
**	指数

算術ステートメント (\* **arithmetic statement**)  
算術演算を実行させるステートメント。算術ステートメントには、ADD、COMPUTE、DIVIDE、MULTIPLY、および SUBTRACT の各ステートメントがある。

## 配列 (array)

データ・オブジェクトで構成される集合体。それぞれのオブジェクトは添え字付けによって一意的に参照できる。配列は、COBOL ではテーブルに類似する。

## 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。7 ビットのコード化文字をベースとする 1 つのコード化文字セット (パリティ・チェックを含む 8 ビット) を使用する標準コードであり、データ処理システム、データ通信システム、および関連装置の間での情報交換に使用される。ASCII セットは、制御文字と図形文字から構成されている。

IBM は、ASCII に対する拡張 (文字 128 から 255) を定義している。

### | ASCII DBCS

| 「2 バイト ASCII (double-byte ASCII)」を  
| 参照。

## 割り当て名 (assignment-name)

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 (AT END condition)

次のような特定の条件のもとで、READ、RETURN、または SEARCH ステートメントを実行した場合に引き起こされる条件。

- 順次アクセス・ファイルに対して READ ステートメントを実行中に、そのファイル内に次の論理レコードが存在しない場合、または相対レコード番号中の有効数字の桁数が相対キー・データ項目のサイズより大きい場合、またはオプションの入力ファイルが使用可能でない場合。

- RETURN ステートメントの実行中に、関連するソート・ファイルまたはマージ・ファイルについての次の論理レコードが存在しない場合。
- SEARCH ステートメントの実行中に、関連する WHEN 句のいずれかで指定された条件を満足することなく、検索操作が終了した場合。

## B

### | 基本文字セット (basic character set)

| 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。基本文字セットは 1 バイトの  
| EBCDICでインプリメントされる。拡張文  
| 字セットには DBCS 文字が含まれる。こ  
| れは、コメント、リテラル、およびユーザ  
| 一定義語で使用する。

| 85 COBOL 標準 における「COBOL 文字  
| セット (COBOL character set)」と同義。

### ビッグ・エンディアン (big-endian)

メインフレームおよび AIX ワークステーションがバイナリー・データおよび UTF-16 文字を保存するときに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 (binary item)

2 進表記 (基数 2 の数体系) で表される数値データ項目。等価の 10 進数は、10 進数字 0 から 9 に演算符号を加えたもので構成される。項目の左端のビットは、演算符号。

### 二分探索 (binary search)

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 (\* block)

通常は 1 つ以上の論理レコードで構成される物理的データ単位。大容量記憶ファイルの場合、ある論理レコードの一部がブロックに入ることがある。ブロックのサイズは、そのブロックが含まれているファイルのサイズと直接関係はなく、そのブ

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 (boolean condition)**

|       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 (boolean literal)**

|       `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。

**停止点 (breakpoint)**

通常は命令によって指定されるコンピューター・プログラムの場所であり、プログラムの実行は外部からの介入またはモニター・プログラムによって割り込まれる場合がある。

**バッファー (buffer)**

入力データまたは出力データを一時的に保持するために使用されるストレージの一部分。

**組み込み関数 (built-in function)**

組み込み関数 (*intrinsic function*) を参照。

**ビジネス・メソッド (business method)**

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)

**バイト (byte)**

特定の数のビット (通常 8 ビット) から成るストリングであり、1 つの単位として処理され、1 つの文字または制御機能を表す。

**バイト・オーダー・マーク (BOM) (byte order mark (BOM))**

UTF-16 または UTF-32 テキストの先頭に使用して、後続テキストのバイト・オーダーを示す Unicode 文字。バイト・オーダーには、「ビッグ・エンディアン (*big-endian*)」または「リトル・エンディアン (*little-endian*)」がある。

**バイトコード (bytecode)**

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 (callable services)**

言語環境プログラムでは、従来の言語環境プログラム定義の呼び出しインターフェースを使用して COBOL プログラムが呼び出すことができる一連のサービス。言語環境プログラムの規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 (called program)**

CALL ステートメントの対象となるプログラム。呼び出し先プログラムと呼び出し側プログラムが実行時に結合されて、1 つの実行単位が作成される。

**呼び出し側プログラム (\* calling program)**

別のプログラムへの CALL を実行するプログラム。

| **標準分解 (canonical decomposition)**

|       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。通常、標準分解は、ラテン語文字と発音区別符号が個別に表されるように発音区別符号付きのラテン語文字を分離するために使用される。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 (case structure)**

結果として生じた多数のアクションの中から選択を行うために、一連の条件をテストするプログラム処理ロジック。

**カタログ式プロシージャ (cataloged procedure)**

プロシージャ・ライブラリー (SYS1.PROCLIB) と呼ばれる区分データ・セットに置かれた一連のジョブ制御ステートメント。カタログ式プロシージャを使用すると、JCL をコーディングする時間を節約して、エラーを減らすことができる。

**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 (century window)**

2 桁年号が固有に決まる 100 年間のこ



と。COBOL プログラマーが使用できる世紀ウィンドウには、いくつかのタイプがある。

- ウィンドウ操作組み込み関数  
DATE-TO-YYYYMMDD、DAY-TO-YYYYDDD、  
および YEAR-TO-YYYY については、  
argument-2 によって世紀ウィンドウを  
指定する。
- 言語環境プログラム呼び出し可能サー  
ビスについては、CEESCEN で世紀ウ  
ィンドウを指定する。

#### 文字 (\* character)

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 (character encoding unit)

コード化文字セット内の 1 つのコード・ポイントに相当するデータの単位。1 つ以上の文字エンコード・ユニットを使用して、コード化文字セットの文字が表現される。エンコード・ユニット と呼ばれる。

USAGE NATIONAL の場合、文字エンコード・ユニットは、UTF-16 の 2 バイト・コード・ポイントに対応している。

USAGE DISPLAY の場合、文字エンコード・ユニットは、1 つのバイトに対応している。

USAGE DISPLAY-1 の場合、文字エンコード・ユニットは、DBCS 文字セットの 2 バイト・コード・ポイントに対応している。

#### 文字位置 (character position)

1 文字を保持または表示するために必要な物理ストレージまたは表示スペースの量。この用語はどのような文字のクラスにも適用される。文字の特定のクラスについては、以下の用語が適用される。

- 英数字文字位置。USAGE DISPLAY を使用して表される DBCS 文字。
- DBCS 文字位置。USAGE DISPLAY-1 を使用して表される DBCS 文字。
- 国別文字位置。USAGE NATIONAL を使用して表される文字。UTF-16 の文字エンコード・ユニット と同義。

#### 文字セット (character set)

テキスト情報を表すために使用されるエレメントの集合。ただし、コード化表現は想定されていません。コード化文字セット (coded character set) も参照。

#### 文字ストリング (character string)

COBOL ワード、リテラル、PICTURE 文字ストリング、またはコメント記入項目を形成する一連の連続した文字。文字ストリングは区切り文字で区切らなければならない。

#### チェックポイント (checkpoint)

ジョブ・ステップを後で再始動することができるように、ジョブとシステムの状態に関する情報を記録しておくことができる場所。

#### クラス (\* class)

ゼロ、1 つ、または複数のオブジェクトの共通の動作およびインプリメンテーションを定義するエンティティ。同じ具体化を共用するオブジェクトは、同じクラスのオブジェクトとみなされる。クラスは階層として定義でき、あるクラスを別のクラスから継承することができる。

#### クラス (オブジェクト指向) (class (object-oriented))

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 (class hierarchy)

オブジェクト・クラス間の関係を示すツリーのような構造。最上部に 1 つのクラスが置かれ、その下に 1 つ以上のクラスの層が置かれる。「継承階層 (inheritance hierarchy)」と同義。

## クラス識別記入項目 (\* class identification entry)

IDENTIFICATION DIVISION の CLASS-ID 段落内の記入項目であり、クラス名を指定する節と、選択した属性をクラス定義に割り当てる節を含む。

## クラス名 (オブジェクト指向) (class-name (object-oriented))

オブジェクト指向 COBOL クラス定義の名前。

## クラス名 (データの) (\* class-name (of data))

ENVIRONMENT DIVISION の SPECIAL-NAMES 段落で定義されるユーザー定義語であり、真理値を定義できる命題に名前を割り当てる。データ項目の内容は、クラス名の定義にリストされている文字だけで構成される。

## クラス・オブジェクト (class object)

クラスを表す実行時オブジェクト。

## 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 (client)

オブジェクト指向プログラミングにおいて、クラス内の 1 つ以上のメソッドからサービスを要求するプログラムまたはメソッド。

## COBOL 文字セット ( COBOL character set)

COBOL 構文を作成する際に使用される文字セット。完全な COBOL 文字セットは、以下の文字で構成される。

文字	意味
0,1, . . . ,9	数字
A,B, . . . ,Z	英大文字
a,b, . . . ,z	英小文字
	スペース
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
\$	通貨符号
,	コンマ
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符

文字	意味
'	アポストロフィ
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロン
_	下線

## COBOL ワード (\* COBOL word)

ワード (*word*) を参照。

## コード・ページ (code page)

すべてのコード・ポイントに図形文字および制御機能の意味を割り当てるもの。例えば、あるコード・ページでは、8 ビット・コードに対して 256 コード・ポイントに文字と意味を割り当て、別のコード・ページでは、7 ビット・コードに対して 128 コード・ポイントに文字と意味を割り当てることができる。ワークステーション上の英語の IBM コード・ページは IBM-1252 で、ホストは IBM-1047 である。コード化文字セット (*coded character set*)。

## コード・ポイント (code point)

コード化文字セット (コード・ページ) に定義する固有のビット・パターン。コード・ポイントには、グラフィック・シンボルおよび制御文字が割り当てられる。

## コード化文字セット (coded character set)

文字セットを設定し、その文字セットの文字とコード化表現との間の関係を設定する明確な規則の集まり。コード化文字セットの例として、ASCII もしくは EBCDIC コード・ページで、または Unicode 対応の UTF-16 エンコード・スキームで表す文字セットがある。

## コード化文字セット ID (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 (\* column)

印刷行または参照形式行におけるバイト位置。列は、行の左端の位置から始めて行の右端の位置まで、1 から 1 ずつ増やして番号が付けられる。列は 1 つの 1 バイト文字を保持する。

## 複合条件 (\* combined condition)

2 つ以上の条件を AND または OR 論理演算子で結合した結果生じる条件。条件 (condition) および 複合否定条件 (negated combined condition) も参照。

## 結合文字 (combining characters)

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)

ドキュメンテーションに使用される IDENTIFICATION DIVISION 内の項目で、実行に影響しない。

## コメント行 (comment line)

行の標識区域のアスタリスク (\*) と、または、プログラム・テキスト区域 (区域 A および区域 B) の最初の文字ストリングとしてのアスタリスクと大なり記号 (\*>) と、その行の区域 A および区域 B に続くコンピューターの文字セットの任意の文字によって表されるソース・プログラム行。コメント行は、文書化にのみ役立つ。行の標識区域では斜線 (/)、そしてその行の区域 A および B ではコンピューター文字セットの任意の文字で表される特殊形式のコメント行があると、コメントの印刷前に改ページが行われる。

## 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 (\* 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。

## コンパイル変数 (compilation variable)

ある特定のリテラル値のシンボル名、あるいは 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演算式のシンボル名。

## 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。

## コンパイル時演算式 (compile-time arithmetic expression)

DEFINE ディレクティブおよび EVALUATE ディレクティブに、または定数条件式に指定されてい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 (compiler)

高水準言語で記述されたソース・コードをマシン言語のオブジェクト・コードに変換するプログラム。

## コンパイラー指示ステートメント

### (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。標準コンパイラー指示ステートメントには、COPY、REPLACE、および USE がある。

### 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 (complex ODO)

次のような OCCURS DEPENDING ON 節の特定の形式。

- 可変位置項目またはグループ:  
DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、非従属データ項目またはグループが続く。グループは英数字グループでも国別グループでも構いません。
- 可変位置テーブル: DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、OCCURS 節によって記述された非従属データ項目が続く。
- 可変長エレメントを持つテーブル:  
OCCURS 節によって記述されたデータ項目に、DEPENDING ON オプションを指定した OCCURS 節によって記述された従属データ項目が含まれている。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

### コンポーネント (component)

(1) 関連ファイルからなる機能グループ化。(2) オブジェクト指向プログラミングでは、特定の機能を実行し、他のコンポーネントやアプリケーションと連携するように設計されている、再使用可能なオブジェクトまたはプログラム。JavaBeans は、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。

### 条件 (例外) (condition (exception))

言語環境プログラム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 (式) (condition (expression))

ある真の値が決定される実行時のデータの状態。条件という用語が本書で一般形式の「条件」(条件-1、条件-2、...) の中、またはこれに関連して使用された場合は、...) 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 ステートメントの中で指定される単純条件または複合条件。単純条件 (simple condition) および 複合条件 (complex condition) も参照。

### 条件句 (\* 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

ENVIRONMENT DIVISION のセクションであり、ソース・プログラムとオブジェクト・プログラムの全体的な仕様およびクラス定義を記述する。

## CONSOLE

オペレーター・コンソールに関連する COBOL 環境名。

## 定数条件式 (constant conditional expression)

IF ディレクティブで、または EVALUATE ディレクティブの WHEN 句で使用される可能性があ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はない場合、関係演算子は "IS EQUAL TO"、"IS NOT EQUAL TO"、"IS ="、"IS NOT ="、または "IS <>" でなければなりません。

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。

## 含まれているプログラム (contained program)

別の COBOL プログラムにネストされている COBOL プログラム。

## 連続項目 (\* contiguous items)

DATA DIVISION 内の連続する記入項目によって記述され、相互に一定の階層関係を持っている項目。

## コピーブック (copybook)

一連のコードが含まれたファイルまたはライブラリー・メンバーであり、コンパイル時に COPY ステートメントを使用してソース・プログラムに組み込まれる。ファイルはユーザーが作成する場合、COBOL によって提供される場合、または他の製品によって供給される場合とがある。「コピー・ファイル (copy file)」と同義。

## 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。

## 通貨記号値 (currency-sign value)

数字編集項目に保管される通貨単位を識別する文字ストリング。典型的な例としては、\$、USD、EUR などがある。通貨記号値は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 (\$) がデフォルトの通貨記号値として使用される。通貨記号 (currency symbol) も参照。

## 通貨記号 (currency symbol)

数字編集項目内の通貨記号値の部分を示すために、PICTURE 節で使用される文字。

通貨記号は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 (\$) がデフォルトの通貨記号値および通貨記号として使用される。通貨記号と通貨符号値は複数定義可能。通貨記号値 (*currency sign value*) も参照。

#### 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 のデータ記述記入項目に現れる節で、データ項目の特定の属性を記述する情報を提供する。

#### データ記述項目 (\* **data description entry**)

COBOL プログラムの DATA DIVISION 内の記入項目であり、レベル番号の後に必要に応じてデータ名が続き、その後に必要に応じて一連のデータ節で構成されるもの。

## DATA DIVISION

COBOL プログラムまたはメソッドの 1 つの部 (division)。使用するファイルとファイルに含まれるレコード、必要となる内部 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なデータなど、プログラムまたはメソッドで処理するデータを記述する。

#### 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 (**data set**)

「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 (**DBCS character**)

IBM の 2 バイト文字セットで定義された任意の文字。

#### DBCS 文字位置 (**DBCS character position**)

文字位置 (*character position*) を参照。

#### DBCS データ項目 (**DBCS data item**)

少なくとも 1 つの記号 G または少なくとも 1 つの記号 N (NSYMBOL (DBCS) コンパイラー・オプションが有効なとき) を含んでいる PICTURE 文字ストリングで記述されたデータ項目。DBCS データ項目は USAGE DISPLAY-1 を持っています。

#### 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 ステートメントが含まれているセクション。

#### 宣言文 (\* **declarative sentence**)

区切り記号のピリオドによって終了する 1 つの USE ステートメントから構成されるコンパイラー指示文。

#### 宣言部分 (\* **declaratives**)

PROCEDURE DIVISION の先頭に書き込まれた 1 つ以上の特殊目的セクションの集合であり、その先頭にはキーワード DECLARATIVE が付き、その最後にはキーワード END DECLARATIVES が続いている。宣言部分は、セクション・ヘッダー、USE コンパイラー指示文、および 0 個、1 個、または複数個の関連する段落で構成される。

#### 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。

定義済み条件 (**defined condition**)  
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。定義済み条件は、IF ディレクティブで、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 (**dependent region**)

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・ユーティリティを含む MVS 仮想記憶領域。

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 (**digit**)

0 から 9 までの任意の数字。COBOL では、この用語を用いて他の記号を参照することはない。

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 (**display floating-point data item**)

暗黙的または明示的に USAGE DISPLAY として記述されており、外部浮動小数点デー

タ項目を記述する PICTURE 文字ストリングを持っている、データ項目。

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。

部の見出し (**\* division header**)

ワードとその後に続く、部の先頭を示す分離文字ピリオドの組み合わせ。部のヘッダーは次のとおり。

IDENTIFICATION DIVISION.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
PROCEDURE DIVISION.

**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 (**DLL application**)

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** リンケージ (**DLL linkage**)

DLL 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 CALL。CALL は、別個のモジュール内のエクスポートされた名前に解決されるか、または、別個のモジュールに定義されたメソッドの INVOKE に解決される。

**Do** 構造 (**do construct**)

構造化プログラミングでは、DO ステートメントを使えば、プロシーチャー内の複数のステートメントをグループ化できる。COBOL では、インライン PERFORM ステートメントが同様に機能する。

**do-until**

構造化プログラミングにおいて、do-until ループは、少なくとも 1 回は実行され、所定の条件が真になるまで実行される。COBOL では、TEST AFTER 句を PERFORM ステートメントで使用すれば、同様に機能する。

## do-while

構造化プログラミングにおいて、do-while ループは、所定の条件が真である場合、および真である間に実行される。COBOL では、TEST BEFORE 句を PERFORM ステートメントで使用すれば、同様に機能する。

## 文書タイプ宣言 (document type declaration)

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 エレメント。この文法は、文書タイプ定義または DTD とも呼ばれる。

## 文書タイプ定義 (document type definition (DTD))

XML 文書のクラスの文法。文書タイプ宣言を参照。

### 2 バイト ASCII (double-byte ASCII)

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」とも呼ばれる)。

### 2 バイト EBCDIC (double-byte 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」とも呼ばれる)。

## 2 バイト文字セット (double-byte character set (DBCS))

それぞれの文字が 2 バイトで表現される 1 組の文字。256 個のコード・ポイントで表現される記号より多くの記号を含んでいる言語 (日本語、中国語、および韓国語など) は、2 バイト文字セットを必要とする。各文字に 2 バイトが必要なため、DBCS 文字の入力、表示、および印刷には、DBCS を受け入れ可能なハードウェアおよびサポートされるソフトウェアが必要。

## 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『DWARF program information』を参照。

## 動的アクセス (\* dynamic access)

1 つの OPEN ステートメントの実行範囲内において、特定の論理レコードを、大容量記憶ファイルからは順次アクセス以外の方法で取り出したりそのファイルに入れたりでき、またファイルからは順次アクセスの方法で取り出せるアクセス・モード。

## 動的 CALL (dynamic CALL)

DYNAM オプションおよび NODLL オプションを使用してコンパイルされたプログラム内の CALL *literal* ステートメント、または NODLL オプションを使用してコンパイルされたプログラム内の CALL *identifier* ステートメント。

## ダイナミック・リンク・ライブラリー (DLL) (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可能ファイルの一部ではないが、実行可能ファイルを正しく実行するためには必要となる可能性がある。

## 動的ストレージ域 (dynamic storage area (DSA))

動的に獲得されるストレージであり、レジスター保管域、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、言語環境プログラム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 進コード) (\* EBCDIC (Extended Binary-Coded Decimal Interchange Code))

8 ビット・コード化文字をベースとするコード化文字セット。



## EBCDIC 文字 (EBCDIC character)

EBCDIC (拡張 2 進化 10 進コード) セットに含まれているいずれかの記号。

### | EBCDIC DBCS

| 「2 バイト EBCDIC (double-byte  
| EBCDIC)」を参照。

## 編集データ項目 (edited data item)

0 の抑止または編集文字の挿入、あるいはその両方を行うことによって変更されたデータ項目。

## 編集用文字 (\* editing character)

次に示す集合に属する 1 文字、または 2 文字で構成される固定した組み合わせ。

文字	意味
	スペース
0	ゼロ
+	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 (EJB container)

J2EE アーキテクチャの 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 server)

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 (encapsulation)

オブジェクト指向プログラミングでは、オブジェクトの固有の詳細を隠すのに使用される技法。オブジェクトは、基礎構造を露出しなくても、データの照会と操作を行うインターフェースを提供する。「情報隠蔽 (*information hiding*)」と同義。

## エンクレーブ (enclave)

言語環境プログラム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 および C の `system()` 関数の使用によって、他のエンクレーブを作成できる。

## エンコード・ユニット (encoding unit)

文字エンコード・ユニット (*character encoding unit*) を参照。

## END CLASS マーカー (end class marker)

語の組み合わせに分離文字ピリオドが続いたもので、COBOL クラス定義の終わりを示す。クラス終了マーカーは次のとおり。

END CLASS *class-name*.

## メソッド終了マーカー (end method marker)

語の組み合わせに分離文字ピリオドが続いたもので、COBOL メソッド定義の終わりを示す。メソッド終了マーカーは次のとおり。

END METHOD *method-name*.

## PROCEDURE DIVISION の終わり (\* end of PROCEDURE DIVISION) (\* end of PROCEDURE DIVISION)

COBOL ソース・プログラムにおいて、それ以後にはプロシー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* .

### Enterprise 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)

### Enterprise 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。

### 項目 (\* entry)

分離文字ピリオドで終了させられる連続する節の記述セットであり、COBOL プログラムの IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、または DATA DIVISION に書き込まれる。

### 環境節 (\* environment clause)

ENVIRONMENT DIVISION 記入項目の一部として現れる節。

### 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 (environment-name)

IBM が指定する名前であり、システム論理装置、プリンターおよびカード穿孔装置の制御文字、報告書コード、またはプログラム・スイッチ、あるいはそれらの組み合わせを識別する。環境名が ENVIRONMENT DIVISION の簡略名と関連付けられている場合は、その簡略名を、置換が有効な任意の形式で置き換えることができる。

### 環境変数 (environment variable)

コンピューター環境の一部の局面を定義する多数の変数のいずれかであり、その環境で動作するプログラムからアクセス可能。

環境変数は、動作環境に依存するプログラムの動作に影響を与える。

### 実行時 (execution time)

実行時 (*run time*) を参照。

### 実行時環境 (execution-time environment)

ランタイム環境 (*runtime environment*) を参照。

### 明示的範囲終了符号 (\* explicit scope terminator)

特定の PROCEDURE DIVISION ステートメントの有効範囲を終わらせる予約語。

### 指数 (exponent)

別の数 (底) をべき乗する指数を示す数。正の指数は乗算を示し、負の指数は除算を示し、小数の指数は数量の根を示す。COBOL では、指数式は記号 \*\* の後に指数を付けて表す。

### 式 (\* expression)

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 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。

### 拡張図形文字セット (extended graphic character set) (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。

### Extensible Markup Language

XML を参照。

### 拡張 (extensions)

85 COBOL 標準 に記述されているものの他に、IBM コンパイラーでサポートされる COBOL 構文およびセマンティクス。

### 外部コード・ページ (external code page)

XML 文書では、CODEPAGE コンパイラー・オプションによって指定された値。

### 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 進数データ項目 (external decimal data item)

ゾーン 10 進数データ項目 (*zoned decimal data item*) および 国別 10 進数データ項目 (*national decimal data item*) を参照。

#### 外部ファイル結合子 (\* external file connector)

実行単位の 1 つ以上のオブジェクト・プログラムにアクセス可能なファイル結合子。

#### 外部浮動小数点データ項目 (external floating-point data item)

表示浮動小数点データ項目 (*display floating-point data item*) および 国別浮動小数点データ項目 (*national floating-point data item*) を参照。

#### 外部プログラム (external program)

最外部プログラム。 ネストされていないプログラム。

#### 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 (factory data)

いったんクラスに割り振られ、クラスのすべてのインスタンスに共用されるデータ。ファクトリー・データは、クラス定義の FACTORY 段落の DATA DIVISION の WORKING-STORAGE SECTION 内に宣言される。Java *private* 静的データと同義。

#### ファクトリー・メソッド (factory method)

オブジェクト・インスタンスとは無関係に、クラスによってサポートされるメソッド。ファクトリー・メソッドは、クラス定義の FACTORY 段落に宣言される。Java *public* 静的メソッドと同義。これらは通常、オブジェクトの作成をカスタマイズする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 (\* file)

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。

#### ファイル節 (\* file clause)

DATA DIVISION の記入項目であるファイル記述項目 (FD 記入項目) およびソート・マージ・ファイル記述項目 (SD 記入項目) のいずれかの一部として現れる節。

#### ファイル結合子 (\* 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 (FILE-CONTROL paragraph)

ENVIRONMENT DIVISION 内の段落であり、

この中では、特定のソース単位で使用されるデータ・ファイルが宣言される。

#### ファイル記述項目 (\* file description entry)

DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 FD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

#### ファイル名 (\* file-name)

DATA DIVISION の FILE SECTION の中のファイル記述項目またはソート・マージ・ファイル記述項目で記述されるファイル結合子に名前を付けるユーザー定義語。

#### ファイル編成 (\* 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 (file position indicator)

概念的エンティティーであり、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 (file system)

データ・レコードおよびファイル記述プロトコルの特定のセットに準拠するファイルの集合、およびこれらのファイルを管理する一連のプログラム。

#### 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 (\* fixed-length record)

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。

#### 固定小数点項目 (fixed-point item)

PICTURE 節で定義される数値データ項目であり、オプションの符号の位置、その中に含まれる桁数、およびオプションの小数点の位置を指定するもの。2 進数、パック 10 進数、または外部 10 進数のいずれかのフォーマットをとることができる。

#### 浮動コメント標識 (\*>) (floating comment indicators (\*>))

浮動コメント標識がプログラムのテキスト領域 (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 (floating point)

実数を 1 対の数表示で表す、数を表記するための形式。浮動小数点表記では、固定小数点部分 (最初の数表示) と、暗黙浮動小数点の底を指数で表される数だけ累乗して得られる値 (2 番目の数表示) との積が、実数になります。例えば、数値 0.0001234 の浮動小数点表記は 0.1234 -3 です (ここで、0.1234 は小数部であり、-3 は指数です)。

#### 浮動小数点データ項目 (floating-point data item)

小数部と指数が入っている数値データ項目。その値は、小数部に、指数で指定されただけ累乗された数値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 (\* format)

データの集合の特定の配列。

#### 機能 (\* function)

ステートメントの実行中に参照された時点で決定される値を持つ、一時的なデータ項目。

## 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。

## 機能名 (function-name)

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 (function-pointer data item)

入り口点を指すポインターを保管できるデータ項目。USAGE IS FUNCTION-POINTER 節で定義されるデータ項目に、関数入り口点のアドレスが含まれる。一般的に、C および Java プログラムと通信するために使用される。

## G

### ガーベッジ・コレクション (garbage collection)

参照されなくなったオブジェクト用のメモリーが 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 (global reference)

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 (group item)

(1) 複数の従属データ項目で構成されるデータ項目。英数字グループ項目 (alphanumeric group item) および 国別グ

ループ項目 (national group item) を参照。

(2) 国別グループまたは英数字グループとして明示的に (またはコンテキストで) 限定されていない場合、この用語は一般のグループを指します。

### グループ区切り文字 (grouping separator)

読みやすさのために数値を何桁かまとめて区切るのに使用される文字。デフォルトの文字はコンマです。

## H

### ヘッダー・ラベル (header label)

(1) 記録メディア・ユニットのデータ・レコードの前にある、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。

### 隠蔽 (メソッド) (hide (a method))

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。

### 高位終了 (\* high-order end)

文字ストリングの左端の文字。

### ハイパースペース (hiperspace)

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 拡張部分 (IBM COBOL extension)

85 COBOL 標準 に記述されているものの他に、IBM コンパイラーでサポートされる 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、またはメソッドを識別する。IDENTIFICATION DIVISION には、作成者名、インストール、または日付を含めることができる。

### ID (\* identifier)

データ項目に名前を付けるための文字ストリングと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 リリース 1 のブートストラップ・ルーチン。これは、COBOL/370 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。

#### IGZCBSO

COBOL (MVS および VM 版) リリース 2、COBOL (OS/390 および VM 版)、および Enterprise COBOL のブートストラップ・ルーチン。これは、COBOL (MVS および VM 版) リリース 2、COBOL (OS/390 および VM 版) または Enterprise COBOL 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。

#### 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。

#### 暗黙の範囲終了符号 (\* implicit scope terminator)

終了していないステートメントが前にある

場合、その範囲を区切る分離文字ピリオド。または、前にある句の中に含まれるステートメントがある場合、そのステートメントの範囲の終わりをそれが現れることによって示すステートメントの句。

**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。

#### 指標 (\* index)

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。

#### 指標データ項目 (\* 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。

#### 指標付き編成 (\* 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 (indexing)

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 (inheritance)

クラスのインプリメンテーションを、別のクラスを基にして使用するメカニズム。定義により、継承するクラスは継承されるクラスに準拠する。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。

### 継承階層 (inheritance hierarchy)

クラス階層 (class hierarchy) を参照。

### 初期設定プログラム (\* 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 (inline)

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 (inline comments)

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 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 または TO OFF 句付き)、START、および WRITE がある。

### 入力プロシージャ (\* input procedure)

ソートすべき特定のレコードの解放を制御する目的で、形式 1 SORT ステートメントの実行時に制御が渡されるステートメントの集合。

### インスタンス・データ (instance data)

オブジェクトの状態を定義するデータ。クラスによって導入されるインスタンス・データは、クラス定義の OBJECT 段落の DATA DIVISION の WORKING-STORAGE SECTION に定義される。オブジェクトの状態には、クラスが導入した、現行クラスによって継承されているインスタンス変数の状態も含まれる。インスタンス・データの個々のコピーは、各オブジェクト・インスタンスごとに作成される。

### 整数 (\* integer)

(1) 小数点の右側に桁位置がない数値リテラル。(2) DATA DIVISION に定義される数値データ項目であり、小数点の右側に桁位置を含まないもの。(3) 関数の起こりうるすべての評価の戻り値で、小数点の右側の桁がすべてゼロであることが定義されている数字関数。

### 整数関数 (integer function)

カテゴリーが数字で、その定義が小数点の右側に桁位置を持たない関数。

### 対話式システム生産性向上機能 (ISPF)

### (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・ユーティリティー、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 (intermediate result)

連続して行われる算術演算の結果を収める中間フィールド。

### 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 進数データ項目 (internal decimal data item)

USAGE PACKED-DECIMAL または USAGE COMP-3 として記述されており、項目を数値として定義する PICTURE 文字ストリング (記号 9、S、P、または V の有効な組み合わせ) を持っている、データ項目。「パック 10 進数データ項目 (packed-decimal data item)」と同義。

### 内部ファイル結合子 (\* internal file connector)

実行単位内にあるただ 1 つのオブジェクト・プログラムのみがアクセスできるファイル結合子。

### 内部浮動小数点データ項目 (internal floating-point data item)

USAGE COMP-1 または USAGE COMP-2 として記述されているデータ項目。COMP-1 は、単精度浮動小数点データ項目を定義します。COMP-2 は、倍精度浮動小数点データ項目を定義します。内部浮動小数点データ項目に関連した PICTURE 節はありません。

### レコード内データ構造 (\* intrarecord data structure)

連続したデータ記述記入項目のサブセットによって定義される、1 つの論理レコードから得られるグループ・データ項目および基本データ項目の集合全体。これらのデータ記述記入項目には、レコード内データ構造を記述している最初のデータ記述記入項目のレベル番号より大きいレベル番号を持つすべての記入項目が含まれる。

### 組み込み関数 (intrinsic function)

よく使用される算術関数のような事前定義関数で、組み込み関数参照によって呼び出される。

### 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 記入項目 (\* I-O-CONTROL entry)

ENVIRONMENT DIVISION の I-O-CONTROL 段落内の記入項目であり、プログラム実行中に指定のファイルへのデータの伝送と処理を行うために必要な情報を提供する節が入っている。

### 入出力モード (\* I-O mode)

ファイルに対する I-O 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

### 入出力状況 (\* I-O status)

入出力操作の結果としての状況を示す 2 文字の値を収める概念上のエンティティ。この値は、そのファイルについてのファイル制御記入項目で FILE STATUS 節を使用することによって、プログラムに使用可能にされる。

**is-a** 継承階層におけるクラスおよびサブクラスの特徴を表す関係。あるクラスに対して is-a 関係を持つサブクラスは、そのクラスから継承する。

**ISPF** 対話式システム生産性向上機能 (ISPF) (Interactive System Productivity Facility (ISPF)) を参照。

### 反復構造 (iteration structure)

ある条件が真である間、あるいはある条件



が真になるまで、一連のステートメントが繰り返して実行されるプログラムの処理ロジック。

## 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)

| **Java Batch Launcher and Toolkit for 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) (Java batch-processing program (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 batch-processing region)**  
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 Database Connectivity (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) (Java message-processing program (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 message-processing region)**  
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) (Java virtual machine (JVM))**  
コンパイル済みの Java プログラムを実行する中央演算処理装置のソフトウェア・インプリメンテーション。

**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)

**JBP** *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。

**JVM** *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 (\* key)

レコードの位置を識別するデータ項目、またはデータの順序付けを識別するための一連のデータ項目。

### 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 (\* keyword)

| 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。  
|

### キロバイト (KB) (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。

### | Language Environment

| z/OS 言語環境プログラム 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。言語環境プログラム に準拠す  
| るコンパイラでコンパイルされたプログラ  
| ムおよび、Java アプリケーションに必  
| 要です。  
|

### 言語環境プログラム 準拠 (Language Environment-conforming)

言語環境プログラム の規約に準拠したオブジェクト・コードを生成するコンパイラ製品 (Enterprise COBOL、COBOL for OS/390 & VM、COBOL for MVS & VM、C/C++ for MVS & VM、PL/I for MVS & VM など) の特性。

### 最後に使われた状態 (last-used state)

内部値がプログラム終了時と同じままで、初期値にリセットされない、プログラムの状態を言う。

### 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 内のレベル標識には、CD、FD、および 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 (Lilian date)

グレゴリオ暦の開始以降の日数。第 1 日は 1582 年 10 月 15 日、金曜日。リリアン日フォーマットは、グレゴリオ暦の考案者であるルイジ・リリオにちなんだ名称。

### \* 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 (link)

(1) リンク接続 (伝送メディア) と、それぞれがリンク接続の終端にある 2 つのリンク・ステーションの組み合わせ。1 つのリンクは、マルチポイントまたはトークンリング構成において、複数のリンク間で共用できる。(2) データ項目あるいは 1 つ以上のコンピューター・プログラムの部分を相互接続すること。例えば、リンケージ・エディターによってオブジェクト・プログラムをリンクして実行可能ファイルを作成すること。

## 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 (linker)

z/OS バインダー (リンケージ・エディター) を指す用語。

## リテラル (literal)

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 (little-endian)

Intel プロセッサが 2 進データおよび UTF-16 文字を保管する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 (local reference)

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 (locale)

プログラム実行環境の一連の属性であり、文化的に重要な考慮事項を示す。例えば、文字コード・ページ、照合シーケンス、日時形式、通貨表記、数値表記、または言語など。

## \* 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。

## 下位終了 (\* low-order end)

文字ストリングの右端の文字。

## M

## メインプログラム (main program)

プログラムとサブルーチンからなる階層において、プロセス内でプログラムが実行されたときに最初に制御を受け取るプログラム。

## Make ファイル (makefile)

アプリケーションに必要なファイルのリストが収められたテキスト・ファイル。make ユーティリティはこのファイルを使用して、ターゲット・ファイルを最新の変更で更新する。

## 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、MB (\* megabyte (MB))

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まり。マージ・ファイルは、マージ機能により作成され、マージ機能によってのみ使用できる。

## メッセージ処理プログラム (MPP)

### (message-processing program (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 (message queue)

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 (method)

オブジェクトによってサポートされる操作の 1 つを定義し、そのオブジェクトに対する INVOKE ステートメントによって実行されるプロシーチャー・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。

## メソッド見出し記入項目 (\* method identification entry)

IDENTIFICATION DIVISION の METHOD-ID 段落内の記入項目。この記入項目には、メソッド名を指定する節が入っている。

## メソッドの起動 (method invocation)

あるオブジェクトから別のオブジェクトへの通信で、受信オブジェクトにメソッドを実行するように要求するもの。

## メソッド名 (method-name)

オブジェクト指向操作の名前。メソッドを起動するのに使用する場合、名前は、英数字リテラル、国別リテラル、カテゴリー英数字データ項目、またはカテゴリー国別データ項目にすることができます。メソッドを定義する METHOD-ID 段落で使用する場合、名前は英数字リテラルまたは国別リテラルにする必要があります。

## メソッド隠蔽 (method hiding)

「隠蔽 (hide)」を参照。

## メソッド多重定義 (method overloading)

「多重定義 (overload)」を参照。

## メソッドのオーバーライド (method overriding)

「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 (module definition file)

プログラム・オブジェクト内のコード・セグメントを記述するファイル。

## 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 (multitasking)

2 つ以上のタスクの並行実行またはインターリーブ実行を可能にする操作モード。

## マルチスレッド化 (multithreading)

コンピュータ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 (name)

COBOL オペランドを定義する 30 文字を超えないで構成されたワード。

### ネーム・スペース (namespace)

XML 名前空間 (XML namespace) を参照。

### 国別文字 (national character)

(1) 国別リテラルまたは USAGE NATIONAL の UTF-16 文字。(2) UTF-16 で表される任意の文字。

### 国別文字データ (national character data)

UTF-16 で表されるデータの一般参照。

### 国別文字位置 (national character position)

文字位置 (character position) を参照。

### 国別データ (national data)

「国別文字データ (national character data)」を参照。

### 国別データ項目 (national data item)

カテゴリー国別、国別編集、または USAGE NATIONAL の数字編集のデータ項目。

国別 10 進数データ項目 (**national decimal data item**) 暗黙的または明示的に USAGE NATIONAL として記述されており、PICTURE の記号 9、S、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。

国別編集データ項目 (**national-edited data item**) 少なくとも 1 つの N のインスタンスおよび単純挿入記号 B、0、または / の少なくとも 1 つを含んでいる PICTURE 文字ストリングで記述されている、データ項目。  
国別編集データ項目は USAGE NATIONAL を持ちます。

国別浮動小数点データ項目 (**national floating-point data item**)

暗黙的または明示的に USAGE NATIONAL として記述されており、浮動小数点データ項目を記述する PICTURE 文字ストリングを持っている、外部浮動小数点データ項目。

国別グループ項目 (**national group item**)

明示的または暗黙的に GROUP-USAGE NATIONAL 節で記述されたグループ項目。  
国別グループ項目は、INSPECT、STRING、および UNSTRING などの操作で、カテゴリー国別の基本データ項目として定義されているかのように処理されます。英数字グループ項目内で USAGE NATIONAL データ項目を定義するのとは対照的に、この処理により、国別文字の埋め込みおよび切り捨てが確実に正しく行われます。グループ内の基本項目を処理する必要がある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、および INITIALIZE など) の場合、国別グループはグループ・セマンティクスを使用して処理されます。

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 (**native method**)

COBOL などの別のプログラム言語で記述されたインプリメンテーションを備える Java メソッド。

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。

次の実行可能なステートメント (\* **next executable statement**)

現在のステートメントの実行完了後に制御が移される次のステートメント。

次のレコード (\* **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**)

WORKING-STORAGE SECTION および LINKAGE SECTION 内の基本データ項目で、他のデータ項目と階層上の関係を持たないもの。

| 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  
| カテゴリーの非数字項目は、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 (**null**)

無効なアドレスの値をポインター・データ項目に割り当てるために使用される形象定

数。 NULL を使えるところならばどこでも、NULLS を使用できる。

#### 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 (**numeric data item**)

(1) 記述により内容が数字 0 から 9 より選ばれた文字で表される値に制限されるデータ項目。符号付きである場合、この項目は +、-、または他の表記の演算符号も含むことができます。(2) カテゴリー数値、内部浮動小数点、または外部浮動小数点のデータ項目。数値データ項目は、USAGE DISPLAY、NATIONAL、PACKED-DECIMAL、BINARY、COMP、COMP-1、COMP-2、COMP-3、COMP-4、または COMP-5 を持つことができます。

#### 数字編集データ項目 (**numeric-edited data item**)

印刷出力の際に使用するのに適したフォーマットの数値データを含むデータ項目。データ項目は、外部 10 進数字の 0 から 9 の数字、小数点、コンマ、通貨符号、符号制御文字、その他の編集記号から構成される。数字編集項目は、USAGE DISPLAY または USAGE NATIONAL のいずれかで表すことができる。

#### 数字関数 (\* **numeric function**)

クラスとカテゴリーは数字だが、考えられる評価のいくつかにおいて整数関数の要件を満たさないような関数。

#### | 数値項目 (\* **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。符号付きの場合は、  
| 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。

## O

#### オブジェクト (**object**)

状態 (そのデータ値) および演算 (そのメソッド) を持つエンティティ。オブジェクトは状態と動作をカプセル化する手段である。クラス内の各オブジェクトは、そのクラスの 1 つのインスタンスであると言われる。

#### オブジェクト・コード (**object code**)

コンパイラまたはアセンブラからの出力。それ自体が実行可能なマシン・コードか、またはその種のコードの作成を目的としての処理に適する。

#### \* **OBJECT-COMPUTER**

ENVIRONMENT DIVISION にある段落の名前であり、ここではオブジェクト・プログラムが実行されるコンピューター環境が記述される。

#### 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 (**object deck**)

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・インスタンス (**object instance**)

| 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。

### 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 (object program)

問題を解決するためにデータと相互に作用することを目的とする実行可能なマシン言語命令とその他の要素の集合またはグループ。このコンテキストでは、オブジェクト・プログラムとは一般に、COBOL コンパイラーがソース・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 (object reference)

クラスのインスタンスを識別する値。クラスが指定されなかった場合、オブジェクト参照は一般的なものとなり、任意のクラスのインスタンスに適用できる。

### オブジェクト時 (\*object time)

オブジェクト・プログラムが実行されるとき。実行時 (run time) と同義。

### 廃止される言語エレメント (\* obsolete element)

2002 COBOL 標準 から削除された 85 COBOL 標準 の COBOL 言語エレメント。

### ODO オブジェクト (ODO object)

次の例では、X が OCCURS DEPENDING ON 節のオブジェクト (ODO オブジェクト) である。

```
WORKING-STORAGE 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 対象 (ODO subject)

上記の例では、Y が OCCURS DEPENDING ON 節のサブジェクト (ODO サブジェクト) である。テーブル内の ODO サブジェクトの数である Y の値は、X の値によって決まる。

### オープン・モード (\* open mode)

OPEN ステートメントが実行されてから、REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。個々のオープン・モードは、OPEN ステートメントの中で、INPUT、OUTPUT、I-O、または EXTEND のいずれかとして指定する。

### オペランド(\* operand)

- (1) オペランドの一般的な定義は、「操作の対象となるコンポーネント」である。
- (2) 本書の目的に沿った言い方をすれば、ステートメントや記入項目の形式中に現れる小文字または日本語で書かれた語 (または語群) はオペランドと見なされ、そのオペランドによって指示されたデータに対して暗黙の参照を行う。

### 演算、操作 (operation)

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 (optional file)

オブジェクト・プログラムが実行されるたびに必ずしも使用可能でなくてもよいものとして宣言されているファイル。

### オプションナル・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 (\* output file)

出力モードまたは拡張モードのいずれかでオープンされるファイル。

## 出力モード (\* output mode)

OUTPUT または EXTEND 句の指定のある OPEN ステートメントが実行されてから、REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。

## 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。

## オーバフロー条件 (overflow condition)

ある演算結果の一部が意図した記憶単位の容量を超えたときに起こる条件。

## 多重定義 (overload)

同じクラスで使用可能な別のメソッドと同一の名前を使い (ただし、異なるシグニチャーを使用して)、メソッドを定義すること。シグニチャー (signature) も参照。

## 指定変更 (override)

サブクラスのインスタンス・メソッド (親クラスから継承された) を再定義すること。

## P

### パッケージ (package)

関連する Java クラスの集まり。個々に、または全体としてインポートすることができる。

### パック 10 進数データ項目 (packed-decimal data item)

内部 10 進数データ項目 (internal decimal data item) を参照。

### 埋め込み文字 (padding character)

物理レコード内の未使用文字位置を埋めるのに使用される英数字または国別文字。

### ページ (page)

データの物理的分離を表す、出力データの

垂直分割。分離は、内部論理要件または出力メディアの外部特性、あるいはその両方に基づいて行われる。

## ページ本体 (\* page body)

行を記述できる、または行送りすることができる (またはその両方ができる) 論理ページの部分。

## 段落 (\* paragraph)

PROCEDURE DIVISION では、段落名の後に分離文字ピリオドが続き、その後に 0 個以上の文が続く。IDENTIFICATION DIVISION および ENVIRONMENT DIVISION では、段落ヘッダーの後に 0 個以上の記入項目が続く。

## 段落ヘッダー (\* paragraph header)

予約語の後に分離文字ピリオドが付いたもので、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION において段落の始まりを示すもの。IDENTIFICATION DIVISION で許可されている段落ヘッダーは次のとおり。

PROGRAM-ID. (Program IDENTIFICATION DIVISION)  
CLASS-ID. (Class IDENTIFICATION DIVISION)  
METHOD-ID. (Method IDENTIFICATION DIVISION)  
AUTHOR.  
INSTALLATION.  
DATE-WRITTEN.  
DATE-COMPILED.  
SECURITY.

ENVIRONMENT DIVISION で許可されている段落ヘッダーは次のとおり。

SOURCE-COMPUTER.  
OBJECT-COMPUTER.  
SPECIAL-NAMES.  
REPOSITORY. (Program or Class CONFIGURATION SECTION)  
FILE-CONTROL.  
I-O-CONTROL.

## 段落名 (\* paragraph-name)

PROCEDURE DIVISION の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 (parameter)

(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。



## Persistent Reusable 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 (pointer data item)

アドレス値を保管できるデータ項目。これらのデータ項目は、USAGE IS POINTER 節を使用してポインターとして明示的に定義される。ADDRESS OF 特殊レジスタは、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。

## 移植する、ポート (port)

(1) 異なるプラットフォームで実行できるようにコンピューター・プログラムを変更すること。(2) インターネット・プロトコルでは、Transmission Control Protocol (TCP) プロトコルまたは User Datagram Protocol (UDP) プロトコルと高水準のプロトコルまたはアプリケーションの間の特定の論理結合子。ポートはポート番号によって識別される。

## 可搬性 (portability)

あるアプリケーション・プラットフォームから別のアプリケーション・プラットフォームに、ソース・プログラムに比較的わずかな変更を加えるだけでアプリケーション・プログラムを移行できる能力。

## 合成済み文字 (precomposed character)

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 (preinitialization)

プログラム (特に非 COBOL プログラム) からの複数の呼び出しの準備としての COBOL ランタイム環境の初期設定。この環境は、明示的に終了されるまで終了されない。

## 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。セグメント番号には 0 から 9 までの文字だけしか使用できない。セグメント番号は 1 桁または 2 桁として表すことができる。

## 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 内にある 1 つの段落または論理的に連続する段落のグループ、あるいは 1 つのセクションまたは論理的に連続するセクションのグループ。

## プロシージャ・ブランチ・ステートメント (\* procedure branching statement)

ソース・コードの中にステートメントが書かれている順番どおりに次の実行可能ステートメントに制御の移動をせず、別のステートメントに明示的に制御の移動を引き起こすステートメント。プロシージャ分岐ステートメントは次のとおり。ALTER、CALL、EXIT、EXIT PROGRAM、GO TO、MERGE (OUTPUT PROCEDURE 句付き)、PERFORM および SORT (INPUT PROCEDURE または OUTPUT PROCEDURE 句付き)、XML PARSE。

## PROCEDURE DIVISION

COBOL の部の 1 つで、問題を解決するための命令を記述する。

### プロシージャ統合 (procedure integration)

COBOL 最適化プログラムの機能の 1 つであり、実行されるプロシージャまたは含まれているプログラムへの呼び出しを単純化する。

PERFORM プロシージャ統合とは、PERFORM ステートメントが、実行されるプロシージャによって置き換えられるプロセスのこと。含まれているプログラムのプロシージャ統合とは、含まれているプログラムへの呼び出しがプログラム・コードによって置き換えられるプロセスのこと。

### プロシージャ名 (\* procedure-name)

PROCEDURE DIVISION 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 (procedure pointer)

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 (procedure-pointer data item)

入り口点を指すポインターを保管できるデータ項目。USAGE IS PROCEDURE-POINTER 節で定義されるデータ項目には、プロシージャ入り口点のアドレスが入っている。一般的に、COBOL および言語環境プログラムのプログラムと通信するために使用される。

### プロセス (process)

プログラムの全部または一部の実行中に発生する一連のイベント。複数のプロセスを並行して実行することができ、1 つのプロセス内で実行されるプログラムはリソースを共用することができる。

### プログラム (program)

(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せん。

### プログラム名 (program-name)

IDENTIFICATION DIVISION とプログラム終了マークにおいて、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 (program-name)

IDENTIFICATION DIVISION およびプログラム終了マークにおいて、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。

### プロジェクト (project)

ダイナミック・リンク・ライブラリー (DLL) や他の実行可能ファイル (EXE) などのターゲットを作成するのに必要な、データおよびアクションの完全セット。

### 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

文字	意味
.	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。

#### 修飾されたデータ名 (\* **qualified data-name**)

データ名と、その後に連結語の OF または IN とデータ名修飾子を続けたものが 1 つ以上のセットで続いて構成される ID。

#### 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。

#### レコード (\* **record**)

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、レコード域の現行の

文字位置の数は、明示または暗黙の RECORD 節によって決められる。

#### レコード記述 (\* **record description**)

レコード記述項目 (*record description entry*) を参照。

#### レコード記述項目 (\* **record description entry**)

特定のレコードに関連したデータ記述項目全体。「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 (**record key**)

索引付きファイル内のレコードを識別する内容を持つキー。

#### レコード名 (\* **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。

#### レコード・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 (**recursion**)

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。

#### 再起可能 (**recursively capable**)

PROGRAM-ID ステートメントで RECURSIVE 属性が指定されていれば、プログラムは再帰可能である (再帰的に呼び出すことができる)。

#### リール (**reel**)

ストレージ・メディアの個別部分。その大きさはインプリメントする人によって決定され、1 つのファイルの一部、1 つのファイルの全部、または任意の個数のファイル

が収容される。「ユニット (unit)」および「ボリューム (volume)」と同義。

#### 再入可能 (reentrant)

プログラムまたはルーチンの属性。この属性によって、プログラム・オブジェクトの 1 つのコピーを複数のユーザーが共用できる。

#### 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 (reference modification)

新規のカテゴリ英数字、カテゴリ DBCS、またはカテゴリ国別のデータ項目を定義する方法であり、USAGE DISPLAY、DISPLAY-1、または NATIONAL データ項目の左端文字および左端文字位置を基準にした長さを指定して定義する方法です。

#### 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。

#### 比較演算子 (\* 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 (\* resource)

オペレーティング・システムの制御下に置

かれており、実行中のプログラムによって使用できる機能またはサービス。

#### 結果の 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)

事前初期設定用の古い COBOL インターフェース (RTREUS ランタイム・オプション)、または言語環境プログラム・インターフェース CEEPIPI のいずれかを使用して、アセンブラー・プログラムをメインプログラムとして設定するときに、再使用可能環境が作成されます。

#### ルーチン (routine)

コンピューターに操作または一連の関連操作を実行させる、COBOL プログラム内の一連のステートメント。言語環境プログラムでは、プロシージャー、関数、またはサブルーチンのいずれかを指す。

#### 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。

#### 実行時 (\* run time)

オブジェクト・プログラムが実行されるとき。「オブジェクト時 (object time)」と同義。

#### ランタイム環境 (runtime environment)

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 の CALL または INVOKE ステートメントによって相互作用し、実行時に 1 つのエンティティーとして機能する複数のオブジェクト・プログラム。

## 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 (scope terminator)

PROCEDURE DIVISION の特定のステートメントの終わりを示す COBOL 予約語。これは明示的なもの (例えば、END-ADD など) であることもあれば、暗黙のもの (分離文字ピリオド) であることもある。

#### セクション (\* section)

ゼロ、1 つ、または複数の段落またはエンティティー (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 (segmentation)

85 COBOL 標準 分割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ー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 (selection structure)

条件が真であるか偽であるかに応じて、あるいは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たプログラム (\* 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。

#### 区切り文字のコンマ (\* separator comma)

文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのコンマ (,)。

#### 分離文字ピリオド (\* 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 (.)。

#### 区切り文字のセミコロン (\* 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 (;)。

#### 順序構造 (sequence structure)

一連のステートメントが、順序どおりに実行されるプログラムの処理ロジック。

#### 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 (serial search)

最初のメンバーから始めて最後のメンバーで終わるように、ある集合のメンバーが連続的に検査される探査方法。

#### Session 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 レベル記述記入項目 (77-level-description-entry)

レベル番号 77 を持つ不連続データ項目を記述するデータ記述記入項目。

#### 符号条件 (\* sign condition)

データ項目や算術式の代数值が、0 より小さいか、大きいか、または等しいかという命題で、それに関して真理値が判別できる。

#### シグニチャー (signature)

- (1) ある操作とそのパラメーターの名前。
- (2) あるメソッドの名前とその仮パラメーターの数と型。

#### 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。

#### | 遊びバイト (レコード内) (slack bytes (within records))

| 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、コンパイラに遊びバイトを挿入させる。

#### 遊びバイト (レコード間) (slack bytes (between records))

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まり。ソート・ファイルは、ソート機能によってのみ作成され使用される。

#### ソート・マージ・ファイル記述項目 (\* sort-merge file description entry)

DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 SD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

#### \* 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 (source program)

ソース・プログラムは、他の形式や記号を使用して表現することができるが、本書では、構文的に正しい COBOL ステートメントの集合を常に指している。COBOL ソース・プログラムは、IDENTIFICATION

DIVISION または COPY ステートメントで開始され、指定された場合はプログラム終了マーカーで終了するか、または追加のソース・プログラム行なしで終了する。

#### ソース単位 (source unit)

COBOL ソース・コードの 1 単位で、個別にコンパイルできる。プログラムまたはクラス定義。コンパイル単位 とも呼ばれる。

#### 特殊文字 (special character)

以下のセットに属する文字。

文字	意味
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
\$	通貨符号
,	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
:	コロンの
_	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徴を記述するために使用される概念。この概念のもとでは、データの特徴は、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と幅を持つデータ外観に適した形式で表現される。

ステートメント (\* **statement**)

COBOL ソース・プログラムに書かれる、動詞を冒頭に置いた、ワード、リテラル、および区切り記号の構文的に正しい組み合わせ。

構造化プログラミング (**structured programming**)

コンピューター・プログラムを編成してコーディングするための技法であり、この技法では、プログラムはセグメントの階層で構成され、それぞれのセグメントには 1 つの入り口点と 1 つの出口点がある。制御は、構造の下方へと渡され、階層内のより上位レベルへの無条件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。

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、(オプションで演算子 + または - 付きの整数が後ろにある) データ名、あるいは (オプションで演算子 + または - 付きの整数が後ろにある) 索引名のいずれかによって表されるオカレンス番号。これによりテーブル内の特定のエレメントを識別

する。可変数の引数を認める関数では、添え字付き ID を関数引数として使用する場合は、添え字に ALL を使用することができる。

添え字付きデータ名 (\* **subscripted data-name**)

データ名とその後の括弧で囲まれた 1 つ以上の添え字から構成される ID。

置換文字 (**substitution character**)

ソース・コード・ページからターゲット・コード・ページへの変換の際に、ターゲット・コード・ページで定義されていない文字を表すのに使用される文字。

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 (**surrogate pair**)

UTF-16 形式のユニコードで、共に 1 つのユニコード図形文字を表すエンコード方式ペアの単位。ペアの最初の単位は上位サロゲートと呼ばれ、第 2 の単位は下位サロゲートと呼ばれる。上位サロゲートのコード値の範囲は、X'D800' から X'DBFF' である。下位サロゲートのコード値の範囲は、X'DC00' から X'DFFF' である。サロゲート・ペアは、Unicode 16 ビット・コード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 (**switch-status condition**)

オンまたはオフに設定可能な UPSI スイッチが、特定の状況に設定されているという命題で、これに関して真理値を判別することができる。

シンボリック文字 (\* **symbolic-character**)

ユーザー定義の形象定数を指定するユーザー定義語。

構文 (**syntax**)

(1) 意味や解釈および使用の方法に依存しない、文字同士または文字のグループ同士の間の関係。(2) 言語における表現の構造。(3) 言語構造を支配する規則。(4) 記号相互の関係。(5) ステートメントの構築にかかわる規則。

システム名 (\* **system-name**)

オペレーティング環境と連絡し合うために使用される COBOL ワード。



## T

### テーブル (\* table)

DATA DIVISION の中で OCCURS 節によって定義される、論理的に連続するデータ項目の集合。

### テーブル・エレメント (\* table element)

テーブルを構成する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 (text deck)

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 (thread)

プロセスの制御下にあるコンピューター命令のストリーム (プロセス内のアプリケーションによって開始される)。

### トークン (token)

COBOL エディターでは、プログラムにおける意味の単位。トークンには、データ、言語キーワード、ID、またはその他の言語構文の一部を含めることができる。

### 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 (trailer-label)

(1) 記録メディア・ユニットのデータ・レコードの後にある、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。

### 真の値 (\* 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。

### 型式化オブジェクト・リファレンス (typed object reference)

指定されたクラスまたはそのサブクラスのオブジェクトだけを参照できるデータ名。

## 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 整数-2 を指定するのではなく、OCCURS 整数-1 to UNBOUNDED によるテーブル。

### 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 では、国別データ・タイプの表記としてビッグ・エンディアン・フォーマットの UTF-16 を使用して Unicode をサポートしている。

### URI (Uniform Resource Identifier (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 構文は、文書「*Uniform Resource Identifier (URI): Generic Syntax*」で定義されています。

#### ユニット (**unit**)

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。

#### 汎用オブジェクト参照 (**universal object reference**)

どのクラスのオブジェクトでも参照できるデータ名。

#### 非制限ストレージ (**unrestricted storage**)

2 GB バイナリ下ストレージ。16 MB 境界より上または下がある。16 MB 境界より上では、31 ビット・モードでのみ、アドレス可能。

#### 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 (**UPSI switch**)

ハードウェア・スイッチの機能を実行するプログラム・スイッチ。UPSI-0 から UPSI-7 の 8 つのスイッチがある。

#### URI *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がある COBOL ワード。

## V

#### 変数 (\* **variable**)

オブジェクト・プログラムの実行によって変更を受ける可能性のある値を持つデータ項目。算術式で使われる変数は、数字基本項目でなければならない。

#### 可変長項目 (**variable-length item**)

OCCURS 節の DEPENDING 句で記述された表を含んだグループ項目。

#### 可変長レコード (\* **variable-length record**)

ファイル記述項目またはソート・マージ・ファイル記述項目が、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。

#### 可変オカレンス・データ項目 (\* **variable-occurrence data item**)

可変オカレンス・データ項目とは、反復される回数が可変であるテーブル・エレメントを言う。そのような項目は、そのデータ記述記入項目内に OCCURS DEPENDING ON 節を持っているか、またはそのような項目に従属していなければならない。

#### 可変位置グループ (\* **variably located group**)

同じレコード内の可変長テーブルに続くグループ項目 (可変長テーブルに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 (\* **variably located item**)

同じレコード内の可変長テーブルに続くデータ項目 (可変長テーブルに従属するわけではない)。

#### 動詞 (\* **verb**)

COBOL コンパイラ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 (**volume**)

外部ストレージのモジュール。テープ装置の場合はリール、直接アクセス装置の場合はユニット。

#### 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 (**VSAM file system**)

COBOL の順次編成、相対編成、および索引編成をサポートするファイル・システム。

## W

#### Web サービス (**web service**)

特定のタスクを実行し、HTTP や SOAP といったオープン・プロトコルを介してアクセス可能なモジュラー・アプリケーション。

## 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

独立項目または WORKING-STORAGE レコード、あるいはその両方から構成される、WORKING-STORAGE データ項目を記述する DATA DIVISION のセクション。

## ワークステーション (workstation)

コンピューターの総称 (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 (wrapper)

オブジェクト指向コードとプロシーチャー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。

XML Extensible Markup Language。マークアップ言語を定義するための標準メタ言語。SGML から派生した、SGML のサブセットである。XML では、SGML の複雑で使用頻度の低い部分が省略され、文書タイプを扱うアプリケーションの作成、構造化情報の作成および管理、異種コンピュータ

ー・システム間での構造化情報の伝送および共有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 data)

XML エLEMENT を持つ階層構造に編成されたデータ。データ定義は XML エLEMENT・タイプ宣言で定義される。

## XML 宣言 (XML declaration)

使用している XML のバージョンや文書のエンコードなど、XML 文書の特性を指定する XML テキスト。

## XML 文書 (XML document)

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 (XML namespace)

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 (XML schema)

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。

## 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 UNIX を使用してアクセスできる。

## ゾーン 10 進数データ項目 (zoned decimal data item)

暗黙的または明示的に USAGE DISPLAY として記述されており、PICTURE の記号 9、S、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。ゾーン 10 進数データ項目の内容は、文字 0 から 9 で表され、必要に応じて符号が付きます。PICTURE ストリングが符号を指

定しており、SIGN IS SEPARATE 節が指定されている場合、符号は文字 + または - として表されます。SIGN IS SEPARATE が指定されていない場合、符号は、符号位置の最初の 4 ビットをオーバーレイする 1 つの 16 進数字です (先行または末尾)。

#

## 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

#### COBOL for z/OS の資料

以下の資料が「Enterprise COBOL for z/OS library」にあります。

- カスタマイズ・ガイド (SC43-3366-01)
- 言語解説書 (SC43-3367-01)
- プログラミング・ガイド (SC43-3368-01)
- 移行ガイド (SC43-3369-01)
- パフォーマンス・チューニング・ガイド (SC43-4104-00)
- メッセージおよびコード (SC43-4107-00)
- *Program Directory* (GI13-4526-01)
- *Licensed Program Specifications* (GI13-4532-01)

#### 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。これらは <http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss> 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/home/product/B984385H82239E03/Enterprise\\_COBOL\\_for\\_z/OS](https://www.ibm.com/support/home/product/B984385H82239E03/Enterprise_COBOL_for_z/OS) 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 ライブラリー」にあります。

ランタイム・ライブラリー拡張機能

- 共通デバッグ・アーキテクチャー ライブラリー・リファレンス
- 共通デバッグ・アーキテクチャー ユーザーズ・ガイド
- DWARF/ELF エクステンション ライブラリー・リファレンス

#### z/Architecture

- *z/Architecture* 解説書

#### z/OSDFSMS

- 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。

- アプリケーション・プログラミング・ガイド
- アプリケーション・プログラミング・リファレンス
- カスタマイズ・ガイド
- 外部インターフェース・ガイド

## COBOL 報告書作成プログラム・プリコンパイラー

- *Programmer's Manual*, SC26-4301

## Db2 for z/OS

以下の資料が「Db2 ライブラリー」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 Debug for z Systems

IBM Debug for z Systems については、IBM Debug for z Systems ライブラリーを参照してください。

注: IBM Debug for z Systems は、IBM Debug Tool for z/OS に置き換わるものです。COBOL 文書ライブラリーで、IBM Debug Tool for z/OS への参照がすべて変更されているわけではありません。多くの COBOL アプリケーションのデバッグに、引き続き IBM Debug Tool for z/OS V13.1 を使用することができます。ただし、Enterprise COBOL V6 に用意されている新しいデバッグ機能を使用する場合は、最新バージョンの IBM Debug for z Systems が必要です。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/support/knowledgecenter/SSQ2R2\\_14.0.0/com.ibm.debugtool.doc/common/dcompo.html](https://www.ibm.com/support/knowledgecenter/SSQ2R2_14.0.0/com.ibm.debugtool.doc/common/dcompo.html) を参照してください。

## IBM Developer for z Systems

IBM Developer for z Systems に関する情報は、IBM Developer for z Systems ライブラリーにあります。

注: IBM Developer for z Systems は、Rational® Developer for z Systems を置き換えるものです。

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 SDK for Java - Tools Documentation*,  
[publib.boulder.ibm.com/infocenter/javasdk/  
tools/index.jsp](http://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](http://download.oracle.com/javaee/1.2.1/devguide/html/DevGuideTOC.html)
- *Java 2 on z/OS*, [www.ibm.com/servers/  
eserver/zseries/software/java/](http://www.ibm.com/servers/eserver/zseries/software/java/)
- *The Java EE 5 Tutorial*, [download.oracle.com/  
javaee/5/tutorial/doc/](http://download.oracle.com/javaee/5/tutorial/doc/)
- *The Java Language Specification, Third Edition*  
(Gosling 他著), [java.sun.com/docs/books/jls/](http://java.sun.com/docs/books/jls/)
- *The Java Native Interface*,  
[download.oracle.com/javase/1.5.0/docs/  
guide/jni/](http://download.oracle.com/javase/1.5.0/docs/guide/jni/)
- *JDK 5.0 Documentation*,  
[download.oracle.com/javase/1.5.0/docs/](http://download.oracle.com/javase/1.5.0/docs/)

## | JSON

- | • JavaScript Object Notation  
| (JSON), [www.json.org](http://www.json.org)

## Unicode および文字表現

- *Unicode*, [www.unicode.org/](http://www.unicode.org/)
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

## XML

- *Extensible Markup Language (XML)*,  
[www.w3.org/XML/](http://www.w3.org/XML/)
- *Namespaces in XML 1.0*,  
[www.w3.org/TR/xml-names/](http://www.w3.org/TR/xml-names/)
- *Namespaces in XML 1.1*,  
[www.w3.org/TR/xml-names11/](http://www.w3.org/TR/xml-names11/)
- *XML specification*, [www.w3.org/TR/xml/](http://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

本書の 962

Enterprise COBOL for z/OS の 961

z/OSの使用 961

アクセス方式サービス・プログラム

代替索引の事前作成 237

VSAM データ・セットのロード 222

z/OS に対する VSAM データ・セットの定義 230

アセンブラー

プログラム

からのコンパイル 298

からの呼び出し (CICS での) 504

マルチスレッド化 610

リスト 387, 786

PROCEDURE DIVISION の拡張 474

値の割り当て 29

アドレス

入り口点アドレスを渡す 568

増分 580

プログラム間での受け渡し 580

NULL 値 580

アドレスを増分する 580

アドレッシング・モード、定義 43

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 22

移行の考慮事項

Java および COBOL 344

異常終了、コンパイル時 372

一括表示表現 780

移動、制御権の

ネストされたプログラム 564

メインプログラムとサブプログラム 551

呼び出し側プログラム 552

呼び出し先プログラム 552

COBOL プログラム相互間の 554, 563

COBOL プログラムと非 COBOL プログラムの間での 551

入り口点

入り口アドレスの受け渡し 568

代替 570

入り口点 (続き)

代替、ENTRY ステートメントの 568  
プロシージャ・ポインター・データ

項目 568

ENTRY ラベル 570

印刷されるページのサイズ、制御 194

インスタンス

削除 723

作成 721

定義 693

インスタンス・データ

初期化 721

それをアクセス可能にする 708

定義 693, 701, 726

private 701

インスタンス・メソッド

オーバーライド 706

多重定義 707

定義 693, 702, 727

呼び出し、オーバーライドした 720

インスタンス・メソッドの多重定義 707

インライン PERFORM

概要 113

例 113

インライン・コメント 985

受け渡し、プログラム間でのデータの

アドレス 580

オプションの考慮事項 46

呼び出し側プログラムの引数 575

呼び出し先プログラムのパラメーター 576

BY CONTENT 573

BY REFERENCE 573

BY VALUE

概要 573

制約事項 576

EXTERNAL データ 584

Java との 748

JNI サービス 744

OMITTED 引数 577

RETURN-CODE 特殊レジスター 583

埋められていないトラック 191

英字データ

国別との比較 167

これを伴う MOVE ステートメント 35

英数字グループ項目

定義 26

GROUP-USAGE NATIONAL なしのグループ 27

英数字データ

これを伴う MOVE ステートメント 35

比較する

国別と 167

ZWB の影響 443

変換

IGZCA2D による DBCS への 818

MOVE による国別への 156

NATIONAL-OF による国別への 157

2 バイト文字を含む 817

英数字比較 108

英数字編集データ

これを伴う MOVE ステートメント 35

初期化

例 31

INITIALIZE の使用 83

英数字リテラル

混合 DBCS/EBCDIC の変換 817

説明 28

2 バイト文字を含む 817

DBCS の内容での 169

エラー

コンパイラー・オプションの競合 349

算術演算 273

処理 271

行順次ファイル 246

QSAM ファイル 196

VSAM ファイル 227

XML GENERATE 682

XML PARSE 656

処理のルーチン 283

入出力の処理 177

メッセージ・テーブル

指標付けを使用した例 86

添え字付けを使用した例 85

リスト 310

エラー・メッセージ

コンパイラー

カスタマイズ 852

作成する重大度レベルの判別 378

重大度レベル 318, 853

ソースの訂正 316

ソース・リストへの組み込み 460

端末への送信 305

出口モジュールからの 861

フォーマット 317

フラグを立てる重大度の選択 460

リストにおける位置 317

エラー・メッセージ (続き)	オブジェクト指向 COBOL (続き)	カタログ式プロシージャ
コンパイラ (続き)	制約事項 (続き)	コンパイル (IGYWC) 289
リストの生成 317	SQL コンパイラ・オプション	コンパイル、リンク・エディット、実行 (IGYWCLG) 292
コンパイラの指示 317	693	コンパイルおよびリンク・エディット (IGYWCL) 291
エンクレーブ 551	SQL ステートメント 517	コンパイル用 JCL 288
エンコード	バインディング	各国語サポート (NLS)
英数字 XML 文書の場合の指定 651	概要 340	データ処理 141
言語文字 146	例 341	DBCS 168
構文解析対象の XML 文書 631	プログラムは再入可能でなければなら	LANGUAGE コンパイラ・オプション 385
生成された XML 出力での制御 681	ない 571	
説明 155	リンク	可変位置グループ 91
CODEPAGE オプションでの指定 359	概要 334	可変位置データ項目 91
XML 文書での矛盾 657, 658	例 336	可変長テーブル
XML 文書の 648, 649	DLL 601	値の割り当て 90
エンコード宣言	IMS	オーバーレイを防ぐ 94
指定 652	データベースへのアクセス 537	作成 87
省略を推奨 651	COBOL からの Java メソッドの呼び出し 536	例 89
オーバーフロー条件	Java からの COBOL メソッドの呼び出し 535	ロードの例 89
ストリングの結合および分割 272	Java との通信 749	可変長レコード
CALL 283	OO プログラムとの間の呼び出し 567	ソート 255
UNSTRING 120	オブジェクト・インスタンス、定義 693	OCCURS DEPENDING ON (ODO) 節 784
オーバーライド	オブジェクト・インスタンスの解放 723	QSAM
インスタンス・メソッド 706	オブジェクト・コード	要求 182
ファクトリー・メソッド 732	コンパイルおよびリスト 310	レイアウト 184
大文字への変換 130	作成 305	VSAM
オブジェクト	生成 362	定義 217
削除 723	80 桁レコードでの作成 367	RRDS 211
作成 721	オプション・ファイル	環境変数
定義 693	QSAM 193	行順次ファイルの割り振り 243
オブジェクト参照	VSAM 221	コピーブック 445
型式化 714		コンパイラ 321
設定 716		設定およびアクセス 543
汎用 714		設定およびアクセスの例 544
比較する 715		テキスト名 322, 445
引き渡しの例 718		ファイルの定義、例 9
ローカルからグローバルへの変換 721		ファイルを割り振るための使用 176
オブジェクト指向 COBOL		ライブラリー名 322, 445
アプリケーションの準備		ランタイム 543
JCL または TSO/E の使用 340		CLASSPATH
z/OS UNIX のもとの 334		設定の例 340
オブジェクト指向プログラムの書き込み 693		説明 544
互換 344		Java クラスの場所の指定 336
コンパイル		COBJVMINITOPTIONS
JCL または TSO/E の使用 339		説明 544
z/OS UNIX のもとの 333		JVM オプションの指定 337
実行		COBOL_INSTALL_DIR 322
JCL または TSO/E の使用 340		COBOPT 322
XPLINK リンケージ 343		LIBPATH
z/OS UNIX のもとの 336		設定の例 340
制約事項		説明 544
ソートとマージ 247		COBOL クラスの場所の指定 336
CICS 693		PATH
CICS のもとでは実行できない 502		設定の例 340
EXEC CICS ステートメント 693		
EXEC SQL ステートメント 693		

## [力行]

- 環境変数 (続き)
  - PATH (続き)
    - 説明 544
  - QSAM ファイルの定義 197
  - STEPLIB
    - 説明 544
    - 例 324
  - SYSLIB
    - 説明 322
    - JNL.cpy の場所の指定 334
  - \_BPX\_SHAREAS 546
  - \_CEE\_ENVFILE
    - 説明 544
  - Java 設定値を示す 341
  - \_CEE\_RUNOPTS
    - 説明 544
    - ランタイム・オプションの指定 542
  - XPLINK の設定 343
  - \_IGZ\_SYSOUT 544
- 環境名 6
- 漢字データのテスト 169
- 漢字比較 108
- 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 570
  - 定義 568
  - 呼び出し DLL プログラム
    - 例 599
  - CALL ステートメント 570
  - COBOL の呼び出し 570
  - DLL での 597
  - JNI サービスのアドレッシング 865
  - SET 関数ポインター 568
- 簡略名
  - SPECIAL-NAMES 段落 6
- 完了コード
  - ソート 260
  - マージ 260
- 関連データ・ファイルの作成 306
- キー
  - 基本、KSDS ファイルの 213
  - 許容データ型
    - MERGE ステートメントでの 257
    - OCCURS 節の 76
    - SORT ステートメントでの 257
  - ソート用の
    - 概要 248
    - 定義 257
  - 相対レコード 214
  - 代替、KSDS ファイルの 214
  - テーブル・エレメントの順序を指定するための 76
  - 二分探索用の 97
  - マージ用の
    - 概要 248
    - 定義 257
- キーボード・ナビゲーション 961
- キーワード 988
- 記述、コンピューター 6
- 基本クラスター名 232
- 基本ライブラリー 304
- 逆順、テープ・ファイルの 192
- 行、テーブルの 77
- 行順次ファイル
  - オープン 244
  - 書き込み 243
  - 国別データはサポートされない 245
  - クローズ 246
  - 再オープン防止のためのクローズ 244
  - 処理 241
  - 制御文字 242
  - 入出力エラー処理 246
  - 入出力ステートメント 243
  - ブロック化 13
  - 編成 241
  - 読み取り 243
  - レコードの追加 245
  - レコードの読み取り 244
  - DATA DIVISION 記入項目 242
  - ENVIRONMENT DIVISION 記入項目 241
  - z/OS のもとでの
    - 作成 243
  - ジョブ制御言語 (JCL) 243
  - 割り振り 243
- 行番号 469, 497
- 共用
  - データ
    - 概要 573
    - 再帰的またはマルチスレッド化されたプログラムの 18
    - パラメーター受け渡しの仕組み 573
    - 別のプログラムからの 17
    - 別々にコンパイルされたプログラム間での 584
    - 別々にコンパイルされたプログラムの 18
    - メソッドから値を戻す 720
    - メソッドへの引数の引き渡し 718
    - 有効範囲、名前の 566
    - Java との 748
    - LINKAGE SECTION のコーディング 578
    - PROCEDURE DIVISION ヘッダー 578
    - RETURN-CODE 特殊レジスター 583
  - ファイル
    - 有効範囲、名前の 566
    - EXTERNAL 節の使用 13, 585
    - GLOBAL 節の使用 13
- 句、定義 20
- 国別 10 進数データ (USAGE NATIONAL)
  - 初期化の例 32
  - 定義 151
  - フォーマット 54
  - 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 151
  - 英数字リテラルを伴う VALUE 節の例 85
  - 概要 151
  - 基本項目として扱われる
    - ほとんどの場合 27, 151
    - MOVE の例 36
  - 国別データのみを含むことができる 27, 153
  - グループ項目として扱われる
    - 要約 154
    - INITIALIZE で 34
    - INITIALIZE の例 154
    - MOVE CORRESPONDING で 36
  - これを伴う MOVE ステートメント 36
  - 使用
    - 概要 152
    - 基本項目として 153
  - 初期化
    - INITIALIZE の使用 34, 83
    - VALUE 節の使用 85
  - 生成された XML 文書の 676
  - テーブルの定義 76
  - 定義 152
  - 引数として渡す 579
  - 例 27
  - BYTE-LENGTH 組み込み関数および 138
  - Java との通信 749
  - LENGTH 組み込み関数および 138
  - USAGE NATIONAL グループとの対比 27
- 国別データ
  - 英数字リテラルを伴う VALUE 節の例 137
  - 外部 10 進数 54
  - 外部浮動小数点 54
  - キーの
    - MERGE ステートメントでの 257
    - OCCURS 節の 76
    - SORT ステートメントでの 257
  - 組み込み関数を使用した評価 134
  - 形象定数 150
  - 検査 (INSPECT) 128
  - これを伴う MOVE ステートメント 35, 156
  - 最小項目または最大項目の検出 135

## 国別データ (続き)

- 指定 147
- 出力での表示 39
- 条件式の 165, 166
- 初期化の例 31
- 生成された XML 文書の 676
- その参照変更 125
- 定義 148
- 比較する
  - 英字、英数字、または DBCS と 167
  - 英数字グループと 167
  - 概要 165
  - 数値との 166
  - 2 つのオペランド 166
- 分割 (UNSTRING) 120
- 変換
  - 大文字または小文字への 130
  - 概要 156
  - ギリシャ語の英数字との間、例 159
  - 中国語 GB 18030 との間の 164
  - 例外 158
  - DISPLAY-OF による英数字への 157
  - INSPECT による 128
  - MOVE による英数字、DBCS、または整数からの 156
  - NATIONAL-OF による英数字または DBCS からの 157
  - NUMVAL、NUMVAL-C による数値への 131
  - UTF-8 との間の 159
- 文字の逆順 131
- リテラル
  - 使用 149
- 連結 (STRING) 118
- ACCEPT による入力 38
- BYTE-LENGTH 組み込み関数および 138
- Java との通信 749
- LENGTH OF 特殊レジスター 138
- LENGTH 組み込み関数および 138
- USAGE 節がない場合の NSYMBOL コンパイラー・オプション 148
- XML 文書のエンコード 649

## 国別比較 108

## 国別浮動小数点データ (USAGE NATIONAL)

- 定義 54, 151

## 国別編集データ

- これを伴う MOVE ステートメント 35
- 初期化
  - 例 32
- INITIALIZE の使用 83
- 定義 148

## 国別編集データ (続き)

- 編集記号 148
- PICTURE 節 148

## 国別リテラル

- 使用 149
- 説明 28

## 組み込みエラー・メッセージ 460

## 組み込み関数

- 英数字データ項目の変換 130
- 国別データ項目の変換 130
- コンパイルの日付の検索 139
- 最大または最小項目の検出 135
- 参照修飾子としての 127
- 紹介 41
- 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 66
  - 整数、浮動小数点、混合 64
  - 同等の言語環境プログラム呼び出し可能サービス 66
  - ネストされた 65
  - 引数としてのテーブル・エレメント 65
  - 引数としての特殊レジスター 65
  - 用途 64
  - 例 64
- 中間結果 811, 814
- データ項目の長さの検出 138
- データ項目の評価 134
- テーブル・エレメントの処理 99
- ネスト 42
- 例
  - ANNUITY 69
  - BIT-OF 133
  - BIT-TO-CHAR 134
  - CHAR 135
  - CURRENT-DATE 68
  - DISPLAY-OF 159
  - HEX-OF 133
  - HEX-TO-CHAR 134
  - INTEGER 127
  - INTEGER-OF-DATE 68
  - LENGTH 68, 136, 138
  - LOG 69
  - LOWER-CASE 130
  - MAX 68, 99, 135, 136
  - MEAN 70
  - MEDIAN 70, 99
  - MIN 127
  - NATIONAL-OF 159
  - NUMVAL 131
  - NUMVAL-C 68, 131
  - ORD 135
  - ORD-MAX 99, 136
  - PRESENT-VALUE 69
  - RANGE 70, 99

## 組み込み関数 (続き)

- 例 (続き)
  - REM 69
  - REVERSE 131
  - SQRT 69
  - SUM 99
  - UPPER-CASE 130
  - WHEN-COMPILED 139
- Unicode 関数の例 162
- UTF-8 160
- 組み込み相互参照
  - 説明 464
  - 例 494
- 組み込みの CICS 変換プログラム
  - 概要 508
  - コンパイラー・オプション 506
  - 利点 508
- 組み込みマップ要約 462, 471
- クライアント
  - 定義 711
- クラス
  - インスタンス化
    - COBOL 722
    - Java 722
  - インスタンス・データ 701
  - オブジェクト、JNI による参照の取得 744
  - 定義 693, 697
  - 名前
    - プログラムにおいて 698
  - ファクトリー・データ 730
  - ユーザー定義 9
  - name
    - 外部 699, 713
- クラス条件
  - データの妥当性検査 454
- テスト
  - 概要 108
  - 漢字の 169
  - 数値の 61
  - DBCS の 169
- クラスター、VSAM 230
- グループ移動と基本移動の対比 36, 153
- グループ項目
  - 可変位置 91
  - 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 153
  - 国別データと比較 167
  - グループ移動と基本移動の対比 36, 153
  - グループ項目として扱われる
    - INITIALIZE で 34
    - INITIALIZE の例 82
  - これを伴う MOVE ステートメント 36

グループ項目 (続き)

初期化

INITIALIZE の使用 33, 82

VALUE 節の使用 84

テーブルの定義 76

定義 26

引数として渡す 579

グローバル名 566

ケース構造、EVALUATE ステートメント 105

計算

算術データ項目 781

指標の 80

添え字の 785

形式 V VSAM ファイルの READ INTO 217

形式、レコードの

可変長

QSAM のレイアウト 184

QSAM 用の要求 182

VSAM 用定義 217

形式 D 206

要求 182

レイアウト 184

形式 F 206

要求 181

レイアウト 182

形式 S

概要 186

要求 185

レイアウト 186

形式 U 206

要求 187

レイアウト 188

形式 V 206

要求 182

レイアウト 184

固定長

QSAM のレイアウト 182

QSAM 用の要求 181

VSAM 用定義 217

スパン

概要 186

要求 185

レイアウト 186

不定形式

要求 187

レイアウト 188

QSAM ASCII テープの場合 206

形象定数

国別文字 150

定義 29

HIGH-VALUE の制約事項 150

継承の階層の定義 695

継続

記入項目 266

継続 (続き)

構文検査 362

プログラム 273

言語環境プログラム呼び出し可能サービス (callable services)

概要 797

各国語サポート 798

組み込み関数との違い 66

サンプル・リスト 799

種類 797

条件処理 797

省略されたフィードバック・コード 800

使用例 800

数学 798

数学用 66

対応する数学組み込み関数 66

動的ストレージ・サービス 798

日時の計算 798

日時用 67

フィードバック・コード 800

メッセージ処理 798

戻りコード 800

CALL での呼び出し 800

RETURN-CODE 特殊レジスター 800

言語間通信

サブプログラム 551

マルチスレッド化 610

CICS のもとで 505

COBOL および Java 間の 743

IMS アプリケーション 537

PL/I タスク 610

検査、有効データの

条件式 108

検証、XML 文書の

概要 641

例 671

コーディング

インスタンス・メソッド 702, 727

エラー、回避 779

オブジェクト指向プログラム

概要 693

再入可能でなければならない 571

技法 12, 779

クライアント 711

クラス定義 697

決定 103

効率的 779

コンストラクター・メソッド 730

サブクラス

概要 724

例 727

条件テスト 109

単純化 795

データ部 12

テーブル 75

コーディング (続き)

テスト条件 109

手続き部 19

入出力ステートメント

行順次ファイル用 243

QSAM ファイル用 191

VSAM ファイル用 218

入出力の概要 174

ファイル入出力の概要 171

ファクトリー定義 729

ファクトリー・メソッド 730

ループ 112

CICS のもとで実行されるプログラム

概要 501

再入可能でなければならない 571

システム日付の取得 503

制約事項 502

呼び出し 504

DISPLAY ステートメント 503

I/O 502

SORT ステートメント 511

Db2 のもとで実行されるプログラム

概要 515

ストアード・プロシーチャーは再入可能でなければならない 571

ストリング・データの CCSID 522

ENVIRONMENT DIVISION 6

EVALUATE ステートメント 105

IDENTIFICATION DIVISION 3

IF ステートメント 103

IMS のもとで実行されるプログラム

概要 529

再入可能でなければならない 571

制約事項 529

Java との相互運用が可能なデータ型 749

SQL ステートメント

概要 517

制約事項 517

SQLIMS ステートメント

概要 530

コード

コピー 796

最適化 786

コード化文字セット

定義 146

XML 文書における 648

コード・ページ

英数字 XML 文書の場合の指定 651

オーバーライド 158

指定 359

定義 146

特殊文字の 16 進値 652

ユーロ通貨サポート 73

Db2 ストリング・データ 522

DBCS 360

コード・ページ (続き)

XML 文書での矛盾 657, 658

コード・ポイント、定義 146

更新、VSAM レコードの 225

構造、INITIALIZE による初期化 33

構造化プログラミング 780

構文エラー

NOCOMPILE コンパイラー・オプションによる検出 457

構文解析データ項目、定義 631

構文図の読み方 xvii

効率、コーディングの 779

互換

オブジェクト指向構文 344

Java および COBOL 344

互換モード 50, 805

固定小数点演算

指数 808

比較 71

評価 70

例の評価 72

固定小数点データ

外部 10 進数 54

固定小数点と浮動小数点との間の変換 58

使用計画 781

中間結果 807

パック 10 進数 56

変換と精度 59

2 進数 55

固定長レコード

QSAM

要求 181

レイアウト 182

VSAM

定義 217

RRDS 210

コピーブック

使用 796

説明 445

相互参照 493

探索 328, 447

ユーザー提供モジュールからの入手 375

コピーブック相互参照、記述 462

コピー・ライブラリー

指定 304

探索順序 447

データ・セット 300

例 796

COPY ステートメント 445

SYSLIB 304

z/OS UNIX 探索順序 322, 328

コプロセッサ、Db2

概要 515

コプロセッサ、Db2 (続き)

推奨コンパイラー・オプション

SQLCCSID 524

ストリング・データの CCSID の決定 522

プリコンパイラーとの相違 524

SQL INCLUDE の使用 517

SQL コンパイラー・オプションで使用可能にする 520

コプロセッサ、IMS

概要 529

SQLIMS コンパイラー・オプションで使用可能にする 532

コメント 973

コメント行 973

小文字への変換 130

混合 DBCS/EBCDIC リテラル

英数字から DBCS への変換 818

DBCS から英数字への変換 820

コンパイラー

エラー・メッセージのリストの生成 317

制限

データ部 12

中間結果の計算 806

メッセージ

カスタマイズ 852

作成する重大度レベルの判別 378

重大度レベル 318, 853

ソース・リストへの組み込み 460

端末への送信 305

出口モジュールからの 861

フラグを立てる重大度の選択 460

戻りコード

概要 318

メッセージ・カスタマイズの影響 855

最も高い重大度に依存 319

z/OS UNIX シェルでの呼び出し

概要 324

例 326

z/OS UNIX のもとでの環境変数 321

コンパイラー指示ステートメント

概要 22

説明 445

コンパイラー・オプション

組み込みの CICS 変換プログラム用 506

コンパイラー呼び出しでの 468

シグニチャー情報バイト 476

指定 307

PROCESS (CBL) の使用 308

状況 468

省略形 345

その表 345

著作権 364

コンパイラー・オプション (続き)

デバッグ用

概要 456

TEST に関する制約事項 455

THREAD に関する制約事項 455

パフォーマンスの考慮事項 788

分離型の CICS 変換プログラム用 506, 510

矛盾する 349

有効な 476

優先順位

バッチで 313

例 314

SYSOPTF データ・セットで 304, 401

z/OS UNIX のもとでの 323

z/OS のもとでの 307

リスト例 485

85 COBOL 標準 準拠 348

ADATA 350

ADV 351

AFP 351

パフォーマンスの考慮事項 788

APOST 352

ARCH 352

パフォーマンスの考慮事項 788

ARITH

説明 354

パフォーマンスの考慮事項 789

AWO

説明 355

パフォーマンスの考慮事項 789

BLOCK0

説明 356

パフォーマンスの考慮事項 789

BUFSIZE 357

CICS 358

CODEPAGE 359

COMPILE 362

COPYLOC 363

CURRENCY 365

DATA 366

DBCS 367

DECK 367

DEFINE 368

DIAGTRUNC 369

DISPSIGN 370

DLL 371

DUMP 372

DYNAM 373, 789

EXIT 374

EXPORTALL 377

FASTSRT 261, 377

パフォーマンスの考慮事項 789

FLAG 378, 460

FLAGSTD 379

コンパイラー・オプション (続き)

HGPR 381  
 パフォーマンスの考慮事項 789  
 IMS および CICS のもとでの 502  
 IMS で推奨される 534  
 INITCHECK 382, 458  
 パフォーマンスの考慮事項 790  
 INITIAL 383  
 INLINE 384  
 パフォーマンスの考慮事項 789  
 INTDATE 385  
 LANGUAGE  
 説明 385  
 バッチ・コンパイルでの例 315  
 LINECOUNT 386  
 LIST 387, 464  
 MAP 388, 462, 464  
 MAXPCF 389  
 MAXPCF(nnn)  
 パフォーマンスの考慮事項 790  
 MDECK 391  
 NAME 392  
 NOCOMPILE 457  
 NOFASTSRT 264  
 NSYMBOL 393  
 NUMBER 394, 465  
 NUMCHECK 395, 458  
 パフォーマンスの考慮事項 790  
 NUMPROC 398  
 NUMPROC(PFD)  
 パフォーマンスの考慮事項 790  
 NUMPROC(PFD|NOPFD) 60  
 OBJECT 399  
 OFFSET 399  
 OPTFILE 400  
 OPTIMIZE  
 説明 402  
 パフォーマンスの考慮事項 786,  
 790  
 OUTDD 403  
 PARMCHECK 404, 458  
 パフォーマンスの考慮事項 791  
 PGMNAME 405  
 QUALIFY 408  
 QUOTE 352  
 RENT  
 説明 409  
 パフォーマンスの考慮事項 791  
 RMODE  
 説明 410  
 パフォーマンスの考慮事項 791  
 RULES 411  
 SEQUENCE 414  
 SERVICE 415  
 SOURCE 415, 464  
 SPACE 416

コンパイラー・オプション (続き)

SQL  
 説明 416  
 Db2 での使用 520  
 SQLCCSID  
 スtring・データの CCSID への影響 522  
 説明 417  
 パフォーマンスの考慮事項 524  
 Db2 コプロセッサで推奨する 524  
 SQLIMS 418  
 SSRANGE 420, 459  
 パフォーマンスの考慮事項 791  
 STGOPT 421  
 SUPPRESS 422  
 SYSOPTF データ・セットでの指定 303  
 TERMINAL 423  
 TEST  
 説明 423  
 デバッグに使用 463  
 パフォーマンスの考慮事項 791  
 THREAD  
 説明 428  
 デバッグに関する制約事項 455  
 パフォーマンスの考慮事項 791  
 TRUNC  
 説明 430  
 パフォーマンスの考慮事項 791  
 TSO での指定 310  
 VBREF 433, 464  
 VLR  
 説明 433  
 VSAMOPENFS 434  
 WORD 435  
 XMLPARSE 436  
 XREF 437, 462  
 ZONECHECK 438  
 ZONEDATA 440  
 ZWB 443  
 z/OS UNIX のもとでの指定 323  
 z/OS のもとでの指定 309  
 コンパイラー・オプションの階層  
 バッチで 313  
 SYSOPTF データ・セットで 401  
 z/OS UNIX のもとでの 323  
 z/OS のもとでの 307  
 コンパイラー・データ・セット  
 コンパイルに必要な 300  
 入出力 300  
 cob2 を使用した 329  
 SYSADATA (ADATA レコード) 306  
 SYSDEBUG (デバッグ・レコード) 306  
 SYSIN 303  
 SYSJAVA 306

コンパイラー・データ・セット (続き)

SYSLIB (ライブラリー) 304  
 SYSLIN (オブジェクト・コード) 305  
 SYSMDECK (ライブラリー処理) 307  
 SYSOPTF 303  
 SYSOUT (リスト) 304  
 SYSPUNCH (オブジェクト・コード) 305  
 SYSTEM (メッセージ) 305  
 z/OS UNIX ファイル・システムでの 289, 294  
 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 328  
 コンパイラー・リスト  
 入手 464  
 コンパイル  
 アセンブラー・プログラムからの 298  
 オブジェクト指向アプリケーション  
 例 336, 341  
 cob2 コマンド 333  
 JCL または TSO/E の使用 339  
 z/OS UNIX のもとでの 333  
 カタログ式プロシーチャーによる 288  
 コンパイル 289  
 コンパイル、リンク・エディット、実行 292  
 コンパイルおよびリンク・エディット 291  
 結果 310  
 シェル・スクリプトの使用 330  
 制御 307  
 データ・セット 300  
 バッチ 311  
 85 COBOL 標準 への準拠 348  
 cob2 コマンドの使用  
 概要 324  
 例 326  
 DLL 325  
 JCL (ジョブ制御言語) による 288  
 TSO のもとでの  
 概要 296  
 CLIST の例 297  
 z/OS UNIX のもとでの 321  
 z/OS UNIX ファイル 290  
 z/OS のもとでの 287  
 コンパイルおよびリンク、z/OS UNIX  
 シェルでの  
 オブジェクト指向アプリケーション  
 例 336  
 cob2 コマンド 335  
 概要 324  
 例 326  
 DLL 325  
 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 378

コンパイル時についての考慮事項 (続き)  
エラー・メッセージ (続き)  
重大度レベル 318  
コンパイラーの指示エラー 317  
コンパイルおよびリンク・ステップの表示 329  
ダンプ、生成 372  
表示後のコンパイルおよびリンク・ステップの実行 329  
コンパイルに使用されるデータ・セット 300  
コンパイルの統計 468  
コンパイル用作業データ・セット 300

## [サ行]

再帰呼び出し  
コーディング 567  
識別 4  
LINKAGE SECTION 18  
最後に使われた状態  
EXIT PROGRAM または GOBACK でのサブプログラム 553  
INITIAL 属性なしのサブプログラム 555, 556  
再実行依頼、ジョブの 774  
再始動  
概要 767  
自動 772  
自動または据え置き 767  
据え置き 772  
再始動、プログラムの 771  
最大または最小項目、検出 135  
最適化  
一括表示表現 780  
一貫性のあるデータ 782  
構造化プログラミング 780  
コンパイラー・オプションの影響 787  
指標計算 785  
指標付け 783  
添え字計算 785  
添え字付け 783  
テーブル・エレメント 783  
定数データ項目 781  
到達不能コード 786  
トップダウン・プログラミング 780  
ネストされたプログラムの統合 787  
バック 10 進数データ項目 782  
パフォーマンスにおける意味 784  
パフォーマンスへの影響 780  
パラメーター引き渡しの影響 577  
含まれているプログラムの統合 787  
プロシージャ統合 787  
未使用データ項目 402, 470  
ライン外の PERFORM 780  
ALTER ステートメントの回避 780

最適化 (続き)  
BINARY データ項目 781  
OCCURS DEPENDING ON 784  
最適化プログラム  
概要 786  
再入可能プログラム 571  
サイン表記 60  
索引付きファイル編成  
指定 213  
説明 172  
削除、VSAM ファイルからのレコードの 226  
作成  
オブジェクト 721  
オブジェクト・コード 305  
可変長テーブル 87  
関連データ・ファイル 306  
ライブラリー処理出力ファイル 307  
SYSJAVA ファイル 306  
z/OS での QSAM ファイル 197, 199  
z/OS での行順次ファイル 243  
サフィックス、cob2 での 329  
サブクラス  
インスタンス・データ 726  
コーディング  
概要 724  
例 727  
サブストリング  
その参照変更 124  
テーブル・エレメントの 125  
サブプログラム  
終了  
影響 552  
説明 551  
定義 573  
メインプログラム 551  
リンケージ 551  
共通データ項目 577  
PROCEDURE DIVISION 578  
サポート xix, 1007  
参考文献 1007  
算術演算  
エラー処理 273  
組み込み関数を使用した 64  
コーディングが容易な COMPUTE ステートメント 63  
算術式  
括弧で囲まれた 63  
参照修飾子としての 127  
説明 63  
非算術ステートメントでの 814  
算術比較 71  
算術評価  
固定小数点と浮動小数点の対比 70  
精度 805  
中間結果 805

算術評価 (続き)  
データ形式の変換 58  
パフォーマンスに関するヒント 781  
変換と精度 59  
優先順位 64, 807  
例 70, 72  
参照修飾子  
組み込み関数、例 127  
としての算術式 127  
としての変数 125  
参照変更  
国別データ 125  
組み込み関数 124  
生成された XML 文書 676  
テーブル 79, 125  
範囲外の値 125  
例 126  
SSRANGE による検査式 420  
UTF-8 文書 160  
サンプル・プログラム 941  
支援テクノロジー 962  
シングニチャー  
固有でなければならない 703  
定義 703  
シングニチャー情報バイト  
データ部 476  
手続き部 476  
有効なコンパイラー・オプション 476  
ENVIRONMENT DIVISION 476  
指数  
固定小数点演算で評価される 808  
パフォーマンスに関するヒント 782  
浮動小数点演算で評価される 813  
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 302  
QSAM ファイル 189, 356  
システム日付  
CICS のもとで 503  
システム名 6  
システム・ダンプ 272  
実行時  
パフォーマンス 779  
ファイル名の変更 10  
マルチスレッド化制限 611  
z/OS UNIX での引数へのアクセス  
概要 547  
例 547  
z/OS での引数へのアクセス  
概要 588  
例 589  
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 340  
XPLINK リンケージ 343  
z/OS UNIX のもとの  
概要 336



実行する、オブジェクト指向アプリケーションを (続き)  
z/OS UNIX のもとでの (続き)  
XPLINK リンケージ 343  
実行単位  
説明 551  
マルチスレッド化での役割 604  
自動再始動 772  
指標  
値を割り当てる 80  
エレメントの変位の計算例 78  
キーの誤りの検出 281  
初期化 81  
増分または減分 81  
他のテーブルの参照 80  
定義 79  
範囲検査 459  
OCCURS INDEXED BY 節による作成 80  
指標付け  
エレメントの変位の計算例 78  
添え字付けより望ましい 783  
テーブル 80  
定義 79  
例 86  
指標データ項目  
添え字や指標として使用できない 81  
USAGE IS INDEX 節による作成 80  
終了 552  
主記憶域のバッファへの割り振り 357  
出力  
行順次ファイル用のコーディング 243  
データ・セット 304  
ファイルへ 171  
CICS 用のコーディング 502  
QSAM ファイル用のコーディング 191  
VSAM ファイル用のコーディング 218  
z/OS でのコンパイラからの 301  
出力ファイル、cob2 での 329  
出力プロシージャ  
コーディング 252  
制約事項 254  
有効でない FASTSORT オプション 262  
例 253, 258  
RETURN または RETURN INTO が必要 252  
順次記憶装置 173  
順次ファイル編成 172  
順序、評価の  
コンパイラ・オプション 349  
算術演算子 64, 807  
使用可能ファイル  
QSAM 193  
VSAM 229

状況コード、VSAM ファイル  
説明 279  
例 280  
条件式  
EVALUATE ステートメント 103  
IF ステートメント 103  
PERFORM ステートメント 114  
条件処理  
言語環境プログラムの使用 797  
入力または出力プロシージャでの 254  
QSAM ファイルのクローズ 195  
VSAM ファイルのクローズ 227  
条件ステートメント  
オブジェクト参照 715  
概要 21  
NOT 句を指定した 21  
条件付きコンパイル出力、例 497  
条件の検査 109  
照合シーケンス  
国別キーのバイナリー 257  
指定 7  
シンボリック文字 9  
代替  
選択 259  
例 8  
非数字比較 7  
文字の序数位置 135  
ASCII 8  
EBCDIC 8  
HIGH-VALUE 7  
ISO 7 ビット・コード 8  
LOW-VALUE 7  
MERGE 7, 259  
NATIVE 8  
SEARCH ALL 7  
SORT 7, 259  
STANDARD-1 8  
STANDARD-2 8  
常駐モード、定義 43  
商標 965  
初期化  
インスタンス・データ 721  
可変長グループ 90  
国別グループ項目  
INITIALIZE の使用 34, 83  
VALUE 節の使用 85  
グループ項目  
INITIALIZE の使用 33, 82  
VALUE 節の使用 84  
構造、INITIALIZE による 33  
テーブル  
エレメントのすべての出現 85  
グループ・レベルの 84  
それぞれの項目の個別の 84  
INITIALIZE の使用 82

初期化 (続き)  
テーブル (続き)  
PERFORM VARYING の使用 115  
例 30  
ジョブ再実行要求 774  
ジョブ・ストリーム 551  
処理  
チェーン・リスト  
概要 580  
例 581  
テーブル  
指標付けを使用した例 86  
添え字付けを使用した例 85  
資料 1007  
身体障がい 961  
診断、プログラムの 468  
シンボリック定数 781  
スイッチおよびフラグ  
スイッチをオフに設定する例 112  
スイッチをオンに設定する例 111  
説明 109  
単一値のテストの例 110  
定義 110  
複数値のテストの例 110  
リセット 111  
スイッチ状況条件 108  
数学  
組み込み関数 64, 69  
言語環境プログラム呼び出し可能サービス (callable services) 66, 798  
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 66  
整数、浮動小数点、混合 64  
同等の言語環境プログラム呼び出し可能サービス 66  
ネストされた 65  
引数としてのテーブル・エレメント 65  
引数としての特殊レジスター 65  
用途 64  
例  
ANNUITY 69  
CURRENT-DATE 68  
INTEGER 127  
INTEGER-OF-DATE 68  
LENGTH 68, 136  
LOG 69  
MAX 68, 99, 135, 136  
MEAN 70  
MEDIAN 70, 99  
MIN 127  
NUMVAL 131  
NUMVAL-C 68, 131  
ORD 135  
ORD-MAX 99  
PRESENT-VALUE 69

## 数字組み込み関数 (続き)

### 例 (続き)

RANGE 70, 99

REM 69

SQRT 69

SUM 99

## 数字編集データ

### 初期化

例 32

INITIALIZE の使用 83

定義 148

編集記号 51

BLANK WHEN ZERO 節

コーディング、数値データでの 148

例 51

PICTURE 節 51

USAGE DISPLAY

初期化の例 32

表示 51

USAGE NATIONAL

初期化の例 33

表示 51

## 数値データ

### 外部 10 進数

USAGE DISPLAY 54

USAGE NATIONAL 54

### 外部浮動小数点

USAGE DISPLAY 54

USAGE NATIONAL 54

国別 10 進数 (USAGE  
NATIONAL) 54

国別との比較 166

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 54

サイン表記 60

定義 49

### 内部浮動小数点

USAGE COMPUTATIONAL-1  
(COMP-1) 56

USAGE COMPUTATIONAL-2  
(COMP-2) 56

### パック 10 進数

サイン表記 60

USAGE COMPUTATIONAL-3  
(COMP-3) 56

USAGE PACKED-DECIMAL 56

表示浮動小数点 (USAGE  
DISPLAY) 54

### 変換

固定小数点と浮動小数点との間の  
58

精度 59

MOVE による国別への 156

## 数値データ (続き)

### 編集記号 51

#### 2 進数

USAGE BINARY 55

USAGE COMPUTATIONAL  
(COMP) 55

USAGE COMPUTATIONAL-4  
(COMP-4) 55

USAGE COMPUTATIONAL-5  
(COMP-5) 55

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 167

### 数値のクラス・テスト

検査、有効データの 61

NUMPROC、NUMCLS の影響 61

数値比較 108

数値リテラルの説明 28

据え置き再始動 772

スキップ、レコードのブロックの 189

### ステートメント

暗黙の範囲終了符号 22

コンパイラー指示 22

条件付き 21

定義 20

ネスト・レベル 469, 497

範囲区切り 21

明示範囲終了符号 22

命令ステートメント 20

ステートメント、プログラムで使用される  
464

ステートメント相互参照リスト  
説明 464

ストライド、テーブル 785

ストライプ拡張形式 QSAM ファイル  
203

### ストリング

処理 117

ヌル終了 579

### Java

宣言する 750

取り扱い 753

### 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 798

ソート時の使用 266

### 装置

順次 173

直接アクセス 173

引数のための 575

マッピング 464

文字データ 155

スパン・ファイル 186

## スパン・レコード形式

説明 185

要求 185

レイアウト 186

### スレッド化

および事前初期設定 606

制御転送 606

プログラムの終了 606

z/OS UNIXの考慮事項 541

### 制御

移動 552

ネストされたプログラム内の 563

プログラムのフロー 103

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 238, 793

制限事項、コンパイラーの

データ部 12

ユーザー・データ 12

静的データ、定義 693

静的データ域、ストレージの割り振り 46

静的マップ 488

### 静的メソッド

定義 693

呼び出し 732

### 静的呼び出し

行う 555

動的呼び出しでの 561

パフォーマンス 560

例 561

製品サポート xix, 1007

### 制約事項

オブジェクト指向プログラム 693

添え字付け 79

入出力プロシージャ 254

### CICS

コーディング 502

ソート 268

ファイル 6

分離型の変換プログラム 508

呼び出し 504

16 MB 境界 502

FILE を使用した検証を伴う構文解  
析 642

OUTDD コンパイラー・オプション  
404

Db2 コプロセッサ 521

### IMS

コーディング 6, 529

16 MB 境界 502

IMS SQL コプロセッサ 532

SQL コンパイラー・オプション 521

SQL ステートメント 517

SQLIMS コンパイラー・オプション  
532

### セクション

グループ化 115

## セクション (続き)

宣言 24

定義 20

## セグメント化 788

### 設定

指標 81

指標データ項目 80

スイッチおよびフラグ 111

### 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 276

マルチスレッド化 277

USE FOR DEBUGGING 455

### ソース・コード

行番号 469, 470, 474, 497

コンパイラ・データ・セット 303

プログラム・リスト 310

リストの説明 464

### ソート

オリジナル・シーケンスの保持 259

可変長レコード 255

完了コード 260

#### キー

概要 248

定義 257

基準 257

作業ファイル

説明 249

終了 260

出力プロシージャ

コーディング 252

例 253, 258

使用ストレージ 266

診断メッセージ 260

制御ステートメントの受け渡し 266

正常終了の判別 260

制約事項 247

説明 247

代替照合シーケンス 259

チェックポイント・リスタート 267

#### テーブル

概要 98

動作の制御 264

特殊レジスター 264

入出力プロシージャに関する制約事項 254

入力プロシージャ

コーディング 251

例 258

### パフォーマンス

可変長ファイル 255

FASTSRT 261

ファイルの説明 249

プロセス 248

## ソート (続き)

ワークスペース 267

CICS のもとで 268

FASTSRT コンパイラ・オプション

同じ QSAM ファイルを入出力両

用で使用 262

パフォーマンスの向上 261

要件 261

NOFASTSRT コンパイラ・オプション 264

z/OS データ・セット定義用の DD ステートメント 255

z/OS のもとで必要なデータ・セット 255

ゾーン 10 進数データ (USAGE

DISPLAY) 395, 438, 440

英数字との比較に対する ZWB の影響 443

サイン表記 60

フォーマット 54

例 49

### 相互参照

組み込み 464

コピーブック 464

ステートメント 464

ステートメント・リスト 433

データおよびプロシージャ名 462

テキスト名およびデータ・セット 462

特殊定義記号 495

プログラム名 493

リスト 437

COPY/BASIS 493

COPY/BASIS ステートメント 464

相対ファイル編成 172

### 装置

クラス 300

要件 300

### 添え字

定義 79

範囲検査 459

変数、例 78

リテラル、例 78

### 添え字付け

参照変更 79

制約事項 79

相対 79

データ名またはリテラルを使用 79

定義 79

変数、例 78

リテラル、例 78

例 85

属性メソッド 708

ソフトコピー情報 xix

## [タ行]

代替入り口点の呼び出し 570

### 代替索引

作成 231

使用 214

パス 231, 232

パスワード 229

パフォーマンスの考慮事項 238

例 232

### 代替照合シーケンス

選択 259

例 8

### 代替予約語テーブル

指定 435

CICS 511

### 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 334

オブジェクト指向プログラムでの使用 336

コンパイル 592

#### 作成

概要 591

z/OS UNIX シェルから 325

説明 591

動的呼び出しでの使用 596

必要なコンパイラ・オプション 325

リンク 593

CALL identifier の使用 595

C/C++ プログラムでの使用 600

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 571

DLL 用のバインダー・オプション 593

Java とのインターオペラビリティでの使用 336

Java とのインターオペラビリティのための 334

OO COBOL アプリケーションで 601

z/OS UNIX ファイル・システム における探索順序 596

### 対話式システム生産性向上機能

(ISPF) 946

対話式プログラムの例 946

多重継承、許可されていない 696, 725

### 探索

#### テーブル

概要 95

逐次探索 95

二分探索 97

パフォーマンス 95

名前の宣言の 567

### 探索順序

z/OS UNIX ファイル・システム における DLL 596

短縮リストの例 466

- ダンプ
  - 要求 271
  - DUMP コンパイラー・オプションを指定した 310
- 端末へのメッセージの送信 423
- 段落
  - グループ化 115
  - 定義 20
- チェーン・リスト処理
  - 概要 580
  - 例 581
- チェックポイント
  - 概要 767
  - 再始動用の JCL の例 774
  - 生成されるメッセージ 771
  - 設計 769
  - 設定 767
  - ソート時の制限 768
  - 単一 768
    - テープ 770
    - ディスク 770
  - テスト 769
  - 複数 768, 770
  - メソッド 768
  - レコード・データ・セット 769
  - 85 COBOL 標準 768
  - DFSORT 時の再始動 267
- 置換
  - データ項目 (INSPECT) 128
  - テキスト、Db2 の考慮事項 525
  - QSAM ファイル内のレコード 194
  - VSAM ファイル内のレコード 226
- 置換文字 150
- 逐次探索
  - 説明 95
  - 例 96
- チューニング考慮事項、パフォーマンス 787, 788
- 中間結果 805
- 中国語 GB 18030 データ
  - 処理 164
- 直接アクセス
  - 記憶装置 (DASD) 238
  - 直接指標付け 80
  - ファイル編成 173
- 追加、レコードの
  - 行順次ファイルへの 245
  - QSAM ファイルへの 194
  - VSAM ファイルへの 225
- 通貨記号
  - 使用 72
  - 複数の文字 73
  - ユーロ 73
  - 16 進数リテラル 73
- データ
  - 受け渡し 573
- データ (続き)
  - 英数字と DBCS との間の変換 817
  - グループ化 579
  - 形式、数値タイプ 52
  - 形式の変換 58
  - 効率的な実行 779
  - 数字 49
  - 妥当性検査 61
  - 非互換 61
  - 分割 (UNSTRING) 120
  - 命名 14
  - レコード・サイズ 14
  - 連結 (STRING) 117
- データ域、動的 373
- データおよびプロシージャ名相互参照の記述 462
- データ記述記入項目 12
- データ項目
  - 可変位置 91
  - 基本、定義 26
  - 共通、サブプログラム・リンケージの 577
  - 組み込み関数を使用した評価 134
  - 組み込み関数を使用した変換 130
  - グループ、定義 26
  - 最小項目または最大項目の検出 135
  - サブストリングの参照 124
  - 参照変更 124
  - 指標によるテーブル・エレメントの参照 79
  - 初期化の例 30
  - 数字 49
  - 分割 (UNSTRING) 120
  - 変換、大文字または小文字への 130
  - マップ 310
  - 未使用 402, 470
  - 文字から数値への変換 131
  - 文字のカウント (INSPECT) 128
  - 文字の逆順 131
  - 文字の置換 (INSPECT) 128
  - 文字の変換 (INSPECT) 128
  - 連結 (STRING) 117
  - 16 進数字または 2 進数字から変換 133
  - 16 進数字または 2 進数字に変換 133
  - 2 バイト文字を含む英数字 817
  - DBCS 817
  - Java の型のコーディング 748
- データ項目の比較 440
  - オブジェクト参照 715
- 国別
  - 英字、英数字、または DBCS と 167
  - 英数字グループと 167
  - 概要 165
  - 数値との 166
- データ項目の比較 (続き)
  - 国別 (続き)
    - 2 つのオペランド 166
  - ゾーン 10 進数および英数字、ZWB の影響 443
- データ項目の分割 (UNSTRING) 120
- データ項目の連結 (STRING) 117
- データ操作
  - 文字データ 117
  - DBCS データ 817
- データ定義 470
- データ定義属性コード 470
- データの検査 (INSPECT) 128
- データ部
  - インスタンス・データ 701, 726
  - インスタンス・メソッド 704
  - 行順次ファイルについての記入項目 242
  - クライアント 714
  - コーディング 12
  - 項目 476
  - 項目のマッピング 388, 464
  - シグニチャー情報バイト 476
  - 制限 12
  - 制約事項 12
  - 説明 12
  - ファクトリー・データ 730
  - ファクトリー・メソッド 731
  - FD 記入項目 12
  - FILE SECTION 12
  - GROUP-USAGE NATIONAL 節 76
  - LINKAGE SECTION 12, 18
  - LOCAL-STORAGE SECTION 12
  - OCCURS DEPENDING ON (ODO) 節 87
  - QSAM ファイルについての記入項目 180
  - REDEFINES 節 84
  - VSAM ファイルについての記入項目 216
  - WORKING-STORAGE SECTION 12
- データ名
  - 相互参照 491
  - 相互参照リスト 310
  - MAP リスト内の 470
  - OMITTED 15
  - VSAM ファイルのパスワード 228
- データ・セット
  - 環境変数を使用した定義 176
  - コンパイラー・オプション 303
  - 出力 304
  - ソース・コード 303
  - 代替データ・セット名 298
  - チェックポイント/再始動の例 774
  - チェックポイント・レコード 769
  - 名前、代替 298

## データ・セット (続き)

ファイル、同じ意味 6

JAVAERR 341

JAVAIN 341

JAVAOUT 341

SYSADATA 306

SYSDEBUG 306

SYSIN 303

SYSJAVA 306

SYSLIB 304

SYSLIN 305

SYSMDECK 307

SYSOPTF 303

SYSPRINT 304

SYSPUNCH 305

SYSTEM 305

## テーブル

値のロード 81

値の割り当て 83

エレメント 75

エレメントのサブストリングの参照  
125

エレメントの参照 79

可変長

オーバーレイを防ぐ 94

作成 87

初期化 90

ロードの例 89

行 77

組み込み関数による処理 99

効率のよいコーディング 783, 785

参照変更 79

指標、定義 79

指標による参照の例 78

初期化

エレメントのすべての出現 85

グループ・レベルの 84

それぞれの項目の個別の 84

INITIALIZE の使用 82

PERFORM VARYING の使用 115

ストライド計算 785

説明 42

ソート

概要 98

添え字、定義 79

添え字による参照の例 78

多次元 76

探索

概要 95

順次 95

逐次 95

パフォーマンス 95

2 進数 97

定義 75

同一エレメント仕様 783

動的ロード 82

## テーブル (続き)

配列との比較 42

深さ 77

ループ 115

レコードの再定義 84

列 75

1 次元 75

2 次元 77

3 次元 77

OCCURS 節による定義 75

テーブルの動的なロード 82

テーブル・ファイル

逆順 192

パフォーマンス 190

定義

デバッグ・データ・セット 306

ファイルの概要 9, 171

ライブラリー 304

QSAM ファイル

z/OS に対する 199

z/OS へ 197

VSAM ファイル

z/OS へ 230

z/OS のもとでのソートまたはマージ・ファイル 255

z/OS への行順次ファイル 243

定数

形象、定義 29

データ項目 781

定義 28

定数域 489

定様式ダンプ 271

ディレクトリー

パスの追加 328

適合要件

85 COBOL 標準 348

INVOKE でのオブジェクト参照引き  
渡しの例 718

INVOKE の RETURNING 句 720

INVOKE の USING 句 718

テキスト名相互参照、記述 462

出口モジュール

生成されるエラー・メッセージ 861

メッセージ重大度のカスタマイズ 850

ライブラリー名の代わりに使用される  
844

COBOL プログラムの呼び出し 842

SYSADATA データ・セットのために  
呼び出される 848

SYSLIB の代わりに使用される 844

SYSPRINT の代わりに使用される 847

テスト

条件 114

数値オペランド 108

データ 108

UPSI スイッチ 108

## 手続き部

インスタンス・メソッド 705

クライアント 712

サブプログラムでの 578

シグニチャー情報バイト 476

ステートメント

コンパイラー指示 22

条件付き 21

範囲区切り 21

命令ステートメント 20

説明 19

存在する 476

追加情報 476

用語 19

RETURNING

使用 583

パラメーターの戻し 19

USING

パラメーターの受け取り 19, 576

BY VALUE 578

鉄道線路構文図の読み方 xvii

デバッグ

概要 451

コンパイラー・オプション

概要 456

TEST に関する制約事項 455

THREAD に関する制約事項 455

データ・セットの定義 306

デバッガーの使用 463

パフォーマンスとの対比 426

ランタイム・オプション 455

COBOL 言語機能の使用 452

デバッグ、言語機能の

クラス・テスト 454

宣言部分 455

デバッグ行 455

デバッグ・ステートメント 455

範囲終了符号 453

ファイル状況キー 453

DISPLAY ステートメント 452

INITIALIZE ステートメント 454

SET ステートメント 454

WITH DEBUGGING MODE 節 455

デバッグ・ツール

コンパイラー・オプション 463

説明 451

統計組み込み関数 70

到達不能コード 786

動的データ域、ストレージの割り振り 46

動的ファイル割り振り

環境変数を使用

行順次ファイル 243

QSAM ファイル 197

VSAM ファイル 233

割り振りの順序 176

CBLQDA を使用 193

動的呼び出し

行う 556

使用時期 557

静的呼び出しでの 561

制約事項 556

パフォーマンス 560

例 561

DLL リンケージでの使用 596

特殊機能指定 6

特殊レジスター

組み込み関数の引数 65

ADDRESS OF

CALL ステートメントでの使用  
574

JNIEnvPtr

JNI 呼び出し可能サービスのため  
の使用 744

LENGTH OF 138, 574

RETURN-CODE 583

SORT-RETURN

ソートまたはマージの終了 260

ソートまたはマージの成功の判断  
260

WHEN-COMPILED 139

XML 構文解析での使用 632, 634

XML-CODE 633, 635

XML-EVENT 633, 634

XML-INFORMATION 633

XML-NAMESPACE 633, 639

XML-NAMESPACE-PREFIX 633, 639

XML-NNAMESPACE 633, 639

XML-NNAMESPACE-PREFIX 633,  
639

XML-NTEXT 633, 638

XML-TEXT 633, 638

特殊レジスター・テーブル 490

特記事項 963

トップダウン・プログラミング

回避するための構成 780

## [ナ行]

内部浮動小数点データ

(COMP-1、COMP-2) 56

長さを検出する、データ項目の 138

名前宣言

探索 567

日時操作

言語環境プログラム呼び出し可能サー  
ビス (callable services) 798

二分探索

説明 97

例 97

入出力

エラー後のロジック・フロー 273

エラーの検査 277

入出力 (続き)

概要 171

コーディングの概要 174

処理エラー

行順次ファイル 246

QSAM ファイル 196, 273

VSAM ファイル 227, 273

FASTSORT オプションによる制御 377

入出力エラー宣言でのデッドロック 277

入出力コーディング

誤った索引キーの検出 281

エラー処理技法 273

正常操作の検査 277

AT END (ファイルの終わり) 句 276

EXCEPTION/ERROR 宣言 276

VSAM 状況コードの検査 279

入力

行順次ファイル用のコーディング 243

ファイルから 171

CICS 用のコーディング 502

QSAM ファイル用のコーディング  
191

VSAM ファイル用のコーディング 218  
z/OS のもとでのコンパイラーへの  
300

入力プロシージャ

コーディング 251

制約事項 254

有効でない FASTSORT オプション 262  
例 258

RELEASE または RELEASE FROM  
が必要 251

ヌル終了ストリング

処理 579

取り扱い 123

例 124

ネストされた COPY ステートメント

796, 844

ネストされた IF ステートメント

コーディング 104

望ましい EVALUATE ステートメン  
ト 104

CONTINUE ステートメント 104

NULL ブランチを伴う 103

ネストされた区切り範囲ステートメント  
23

ネストされた組み込み関数 65

ネストされたプログラム

移動、制御の 563

指針 564

説明 564

マップ 464, 474

有効範囲、名前の 566

呼び出し 563

INITIAL 節の影響 5

ネストされたプログラムの統合 787

ネストされたプログラム・マップ

説明 464

例 474

ネスト・レベル

ステートメント 469, 497

プログラム 469, 474, 497

## [ハ行]

バイト・オーダー・マークが生成されない  
681

バイト・ストリーム・ファイル

QSAM による処理 205

配列

COBOL 42

Java

宣言する 750

取り扱い 751

バインダー

c89 コマンド 324

DLL の場合に推奨 593

DLL の場合に必要なオプション 593

バインドする、オブジェクト指向アプリケ  
ーションを

例 341

JCL または TSO/E の使用 340

パス名

コピーブックの検索に使用 328, 445

パスワード

代替索引 229

例 229

VSAM ファイル 228

パック 10 進数データ 395

パック 10 進数データ項目

効率的な使用 56, 782

サイン表記 60

説明 56

同義語 53

バッチ・コンパイル

オプションの優先順位

概要 313

例 314

説明 311

LANGUAGE オプション

例 315

バッファ

最適な使用 11

QSAM 用に取得 204

パフォーマンス

一貫性のあるデータ型 782

検証を伴う XML 文書の構文解析 642

コーディング 779

コンパイラー・オプション

AFP 788

ARCH 788

ARITH 789

パフォーマンス (続き)

コンパイラー・オプション (続き)

AWO 789  
 BLOCK0 789  
 DYNAM 789  
 FASTSRT 789  
 HGPR 789  
 INITCHECK 790  
 INLINE 789  
 MAXPCF 790  
 NUMCHECK 790  
 NUMPROC 60, 790  
 OPTIMIZE 786, 790  
 PARMCHECK 791  
 RENT 791  
 RMODE 791  
 SQLCCSID 524  
 SSRANGE 791  
 TEST 791  
 THREAD 429, 791  
 TRUNC 430, 791  
 コンパイラー・オプションの影響 787  
 最適化プログラム  
   概要 786  
 算術式 782  
 算術評価 781  
 指数 782  
 ストライプ拡張形式 QSAM データ・  
   セット 203  
 チューニング 779  
 データの使用 781  
 テープ、QSAM 190  
 テーブル探索  
   逐次探索の改善 96  
   二分と逐次の比較 95  
 テーブルのコーディング 783  
 テーブルの処理 785  
 デバッグとの対比 426  
 バッファ・サイズの影響 357  
 プログラミング・スタイル 780  
 ブロック化、QSAM ファイルの 188,  
   356  
 変数添え字のデータ形式 79  
 呼び出し 560  
 ライン外 PERFORM とインラインの  
   比較 113  
 ワークシート 792  
 AIXBLD ランタイム・オプション 793  
 APPLY WRITE-ONLY 節 11  
 CBLPSHPOP に関する考慮事項 512  
 CBLPSHPOP ランタイム・オプション  
   512  
 CICS  
   概要 779  
 CICS コーディング 793

パフォーマンス (続き)

EVALUATE 内の WHEN 句の順序

106  
 FASTSRT 261  
 FASTSRT によるソート 261  
 IMS 環境 534, 793  
 OCCURS DEPENDING ON 784  
 VSAM ファイル 237, 793  
 パラメーター  
   呼び出し先プログラムの中での記述  
     576  
   ADEXIT 849  
   INEXIT 843  
   LIBEXIT 845  
   MSGEXIT 851  
   PRTEXIT 847  
   z/OS UNIX でのメインプログラムか  
     らのアクセス  
       概要 547  
       例 547  
   z/OS でのメインプログラムからのア  
     クセス  
       概要 588  
       例 589  
 範囲区切りステートメント  
   説明 21  
   ネストされた 23  
 範囲終了符号  
   暗黙の 22  
   デバッグでの補助 453  
   明示的 21, 22  
 範囲終了符号としてのピリオド 22  
 汎用オブジェクト参照 714  
 比較条件 108  
 引数  
   メインプログラムからの  
     z/OS UNIX でのアクセス 547  
     z/OS でのアクセス 588  
   呼び出し側プログラムでの記述 575  
   BY VALUE で渡す 576  
   OMITTED の指定 577  
   OMITTED 引数に関するテスト 577  
 引数として渡すデータのグループ化 579  
 引数を省略するための OMITTED 句 577  
 ビッグ・エンディアン、リトル・エンディ  
   アンへの変換 147  
 日付操作  
   コンパイルの日付の検索 139  
 評価、データ項目の内容の  
   組み込み関数 134  
   クラス・テスト  
     概要 108  
     数値の 61  
   INSPECT ステートメント 128  
 表現  
   データ 61

表現 (続き)

符号 60

表示浮動小数点データ (USAGE  
 DISPLAY) 54  
 ファイル  
   オプション  
     QSAM 193  
     VSAM 221  
   オペレーティング・システムに対する  
     定義 9  
   外部 585  
   概要 172  
   行順次、割り振り 243  
   使用可能  
     QSAM 193  
     VSAM 229  
   使用法の説明 10  
   処理  
     行順次 241  
     マルチスレッド化 607  
   QSAM 179  
   VSAM 209  
 説明 12  
 属性 201  
 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  
   可変長ファイル 255  
   FASTSRT 261  
 ファイル定義レコードのストレージ  
   608  
 プログラム・ファイルを外部ファイル  
   に関連付ける 6  
 マルチスレッド化処理  
   シリアライゼーション 607  
   推奨使用パターン 608  
   推奨編成 608  
   例 609  
 利用不能  
   QSAM 193  
   VSAM 229  
 CICS での制限 6  
 COBOL コーディング  
   概要 174  
   入出力ステートメント 191, 218,  
     243  
   DATA DIVISION 記入項目 180,  
     216, 242  
   ENVIRONMENT DIVISION 記入  
     項目 180, 212, 241  
   z/OS に対する識別 199  
   z/OS への識別 197, 230  
 ファイル位置標識 (CRP) 220, 224  
 ファイル記述 (FD) 項目 14  
 ファイル状況キー  
   エラー処理 453

- ファイル状況キー (続き)
  - エラー処理の場合の設定 177
  - 正常 OPEN かどうかの検査 277, 279
  - 入出力エラーの検査 277
  - 05 220
  - 35 220
  - 39 220
  - VSAM 状況コードと一緒に使用 279
  - VSAM、の重要性 227
- ファイル状況コード
  - 02 224
  - 30 222
  - 37 191
  - 39 192, 201, 205
  - 49 226
  - 90 190, 196, 227
  - 92 226, 545
- ファイルのオープン
  - 行順次 244
  - マルチスレッド化シリアライゼーション 607
  - QSAM 192
  - VSAM
    - 概要 220
    - 空の 221
- ファイルの終わり (AT END 句) 276
- ファイルの可用性
  - z/OS のもとでの QSAM ファイル 193
  - z/OS のもとでの VSAM ファイル 229
- ファイルのクローズ
  - 行順次 246
  - マルチスレッド化シリアライゼーション 607
  - QSAM
    - 概要 195
    - マルチスレッド化 196
  - VSAM
    - 概要 227
    - マルチスレッド化 227
- ファイルのクローズ、自動的な
  - 行順次 246
  - QSAM 195
  - VSAM 227
- ファイルの割り振り 176
- ファイル編成
  - 概要 171
  - 行順次 241
  - 索引付き 172, 213
  - 順次 172, 212
  - 選択 173
  - 相対 172
  - 相対レコード 214
  - ESDS, KSDS, RRDS の比較 211
  - QSAM 179
- ファイル編成 (続き)
  - VSAM 210
- ファイル割り振り
  - 行順次 243
  - 説明 176
  - QSAM 197
  - TSO のもとでの 296
  - VSAM 233
- ファイル・アクセス・モード
  - 索引付きファイル (KSDS) の場合 215
  - 順次 215
  - 順次ファイル (ESDS) の場合 215
  - 選択 173
  - 相対ファイル (RRDS) の場合 215
  - 動的 215
  - パフォーマンスの考慮事項 238
  - 要約テーブル 212
  - ランダム 215
  - 例 216
- ファクトリー定義、コーディング 729
- ファクトリー・セクション、定義 729
- ファクトリー・データ
  - それをアクセス可能にする 730
  - 定義 693, 730
  - private 730
- ファクトリー・メソッド
  - 隠蔽 732
  - 定義 693, 730
  - プロシージャ・プログラムのラップに使用 738
  - 呼び出し 732
- ファクトリー・メソッドの隠蔽 732
- 深さ、テーブルの 77
- 複合 OCCURS DEPENDING ON
  - 可変位置グループ 91
  - 可変位置データ項目 91
  - 基本形式 91
  - 複合 ODO 項目 91
- 複数の通貨記号
  - 使用 73
  - 例 74
- 含まれているプログラムの統合 787
- 符号条件
  - 数値オペランドの符号のテスト 108
- 物理ブロック 171
- 物理レコード 15, 171
- 不定形式レコード形式
  - 要求 187
  - レイアウト 188
  - QSAM 206
- 浮動コメント標識 (\*>) 982
- 浮動小数点演算
  - 指数 813
  - 比較 71
  - 評価 70
  - 例の評価 72
- 浮動小数点データ
  - 外部 54
  - 固定小数点と浮動小数点との間の変換 59
  - 使用計画 781
  - 中間結果 813
  - 内部
    - 形式 56
    - パフォーマンスに関するヒント 782
  - 変換と精度 59
- フラグおよびスイッチ 109
- ブランチ、暗黙の 113
- プログラム
  - 決定
    - スイッチおよびフラグ 109
    - ループ 114
    - EVALUATE ステートメント 103
    - IF ステートメント 103
    - PERFORM ステートメント 114
  - 構造体 3
  - 再始動 771
  - 再入可能 571
  - サブプログラム 551
  - シグニチャー情報バイト 476
  - 初期化コード 483
  - 診断 468
  - 制約 779
  - 属性コード 474
  - 統計 468
  - ネスト・レベル 469, 497
  - メイン 551
  - cob2 によるコンパイルおよびリンク
    - 概要 324
    - 例 326
    - DLL 325
  - z/OS UNIX の開発 541
  - z/OS UNIX のもとでのコンパイル 321
  - z/OS のもとでのコンパイル 287
- プログラム終了
  - ステートメント 552
  - メインおよびサブプログラムで取られるアクション 552
- プログラム情報
  - リスト例 485
- プログラム処理テーブル 504
- プログラムのドキュメンテーション 6
- プログラム名
  - 指定 3
  - 相互参照 493
  - 大/小文字の処理 405
  - 特定の接頭部の使用を回避 3
- プログラム・プロローグ域
  - リスト例 486
- プロシージャおよびデータ名相互参照の記述 462



プロシージャー統合 787  
プロシージャー・ポインター・データ項目  
  入り口点の入り口アドレス 568  
  定義 568  
  呼び出し可能サービスへのパラメータ  
    の受け渡し 568  
  C/C++ の呼び出し 570  
  DLL での 597  
  JNI サービスの呼び出し 570  
  SET プロシージャー・ポインター 568  
プロセス  
  定義 604  
ブロック化、レコードの 188  
ブロック化、QSAM ファイルの  
  BLOCK CONTAINS 節の使用 188  
  BLOCK0 の使用 356  
ブロック化因数の定義 181  
ブロック・サイズ  
  コンパイラー・データ・セット 302  
  システム決定の  
    コンパイラー・データ・セット 302  
  QSAM ファイル 189, 356  
ASCII ファイル 206  
QSAM ファイル 188, 356  
  可変長 182  
  固定長 181  
  レコード・レイアウト 184  
  DCB の使用 198  
文、定義 20  
分離型の CICS 変換プログラム  
  コンパイラー・オプション 506, 510  
  使用 510  
  制約事項 508  
分離符号  
  移植性 50  
  印刷 50  
  行順次ファイル用 245  
  符号付き国別 10 進数に必要 50  
ページ  
  制御 194  
  深さ 15  
ベース・ロケーター 470, 471  
ベース・ロケーター・テーブル 489  
変換、データ項目の  
  英数字へ  
    DISPLAY による 39  
    DISPLAY-OF を使用した 157  
  大文字または小文字への  
    組み込み関数を使用した 130  
    INSPECT による 129  
  国別から UTF-8 への 159  
  国別から中国語 GB 18030 へ 164  
  国別データでの例外 158  
  国別と  
    中国語 GB 18030 から 164  
    ACCEPT による 38

変換、データ項目の (続き)  
  国別と (続き)  
    MOVE を使用した 156  
    NATIONAL-OF を使用した 157  
    UTF-8 から 159  
  組み込み関数を使用した 130  
  コード・ページ間の 132  
  精度 59  
  データ・フォーマット間の 58  
  文字の逆順 131  
  16 進数字から 133  
  16 進数字に 133  
  16 進数字または 2 進数字から  
    組み込み関数を使用した 133  
  16 進数字または 2 進数字に  
    組み込み関数を使用した 133  
  2 進数字から 133  
  2 進数字に 133  
  INSPECT による 128  
  INTEGER、INTEGER-PART による整  
    数への 127  
  NUMVAL、NUMVAL-C による数値  
    への 131  
変換、COBOL データから XML への  
  概要 676  
  例 683  
変更  
  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 131  
変数  
  参照修飾子としての 125  
  定義 25  
変数、環境  
  設定およびアクセスの例 544  
  ライブラリー名 445  
  ランタイム 543  
ポインター・データ項目  
  アドレスの受け渡し 580  
  アドレスの増分 580  
  説明 42  
  チェーン・リストの処理 580  
  チェーン・リストの処理に使用 581  
  NULL 値 580  
保護、VSAM ファイルの 228  
保持、ソートでのオリジナル・シーケンス  
  の 259  
本製品のアクセシビリティ機能 961

## [マ行]

マージ  
  完了コード 260  
  キー  
    概要 248  
    定義 257

マージ (続き)  
  基準 257  
  作業ファイル  
    説明 249  
  終了 260  
  使用ストレージ 266  
  診断メッセージ 260  
  制御ステートメントの受け渡し 266  
  正常終了の判別 260  
  制約事項 247  
  説明 247  
  代替照合シーケンス 259  
  ファイルの説明 249  
  プロセス 248  
  z/OS データ・セット定義用の DD ス  
    テートメント 255  
  z/OS のもとに必要なデータ・セット  
    255  
マッピング、DATA DIVISION 項目の  
  464  
マルチスレッド化  
  概要 603  
  言語間通信 610  
  再帰 606  
  再帰的要件 610  
  再入可能性 610  
  再入可能性要件 610  
  事前初期設定 606  
  制御転送 606  
  制約 610  
  ソートおよびマージの制限 247  
  データ・セクションの選択 603  
    OO クライアントにおいて 715  
  入出力エラー宣言 277  
  ネストされたプログラム 610  
  非同期シグナル 611  
  ファイル入出力のコーディング  
    シリアライゼーション 607  
    推奨使用パターン 608  
    推奨編成 608  
    例 609  
  古いコンパイラー 611  
  プログラムの終了 606  
用語 604  
ランタイムの制限 611  
リソースへのアクセスの同期化 610  
AMODE 設定 611  
COBOL プログラム 603  
COBOL プログラムの準備 603  
EXIT PROGRAM ステートメント 552  
GOBACK ステートメント 553  
IGZEOPT 611  
IGZETUN 611  
PL/I タスクとの 610  
QSAM ファイルのクローズ 196  
STOP RUN ステートメント 553

マルチスレッド化 (続き)

THREAD コンパイラー・オプション  
いつ選択するか 605  
制限 429

UPSI スイッチ 611

VSAM ファイルのクローズ 227

マルチスレッド化での UPSI スイッチ  
611

マルチスレッド化での非同期シグナル 611

マルチスレッド環境での実行 428

マルチスレッドでのファイルのシリアライ  
ゼーション 607

矛盾するコンパイラー・オプション 349

無制限グループ

処理 100

明示範囲終了符号 22

命名

ファイル 9

プログラム 3

命令ステートメントのリスト 20

メインプログラム

サブプログラム 551

動的呼び出し 556

z/OS UNIX でのパラメーター・リス  
トへのアクセス

概要 547

例 547

z/OS でのパラメーター・リストへの  
アクセス

概要 588

例 589

メソッド

インスタンス 702, 727

オーバーライド 706, 732

から値を戻す 706

コンストラクター 730

シグニチャー 703

スーパークラスの呼び出し 720

多重定義 707

ファクトリー 730

ファクトリーの隠蔽 732

呼び出し 716, 732

渡された引数の取得 706

Java アクセス制御 748

メッセージ

コンパイラー

カスタマイズ 852

作成する重大度レベルの判別 378

重大度レベル 318, 853

ソース・リストへの組み込み 460

端末への送信 305

フラグを立てる重大度の選択 460

リストの生成 317

コンパイラーの指示 317

出口モジュールからの 861

SYSTEM への送信 423

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 798

メモリー・マップ

DSA 474

文字セット、定義 146

文字の逆順 131

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 583

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 800

コンパイラー

概要 318

メッセージ・カスタマイズの影響  
855

最も高い重大度に依存 319

CICS ECI からの 506

Db2 SQL ステートメントからの 520

RETURN-CODE 特殊レジスター 583,  
800

VSAM ファイル

説明 279

例 280

RLS モード 236

XML 構文解析の 654, 823

## [ヤ行]

ユーザー定義の条件 108

ユーザー出口作業域 841

ユーロ通貨記号 73

有効範囲、名前の

グローバル 566

ローカル 566

優先順位

コピーブック探索順序 322

コンパイラー・オプション

バッチで 313

SYSOPTF データ・セットで 304,  
401

z/OS UNIX のもとでの 323

z/OS のもとでの 307

算術演算子 64, 807

CICS オプション 507

優先順位番号、セグメンテーションの 788

優先符号 60

用語

VSAM 209

用語集 967

呼び出し

インスタンス・メソッド 716

オーバーフロー条件 283

オブジェクト指向プログラムとの間の  
567

言語環境プログラム 呼び出し可能サー  
ビスへ 800

呼び出し (続き)

言語環境プログラム呼び出し可能サー  
ビス (callable services) 800

言語間の 551

再帰的 567

静的

行う 555

動的呼び出しでの 561

パフォーマンス 560

例 561

データの受け渡し 573

動的

行う 556

静的呼び出しでの 561

制約事項 556

パフォーマンス 560

例 561

パラメーターの受け取り 576

引数の受け渡し 575

ファクトリーまたは静的メソッド 732

例外条件 283

24 ビット・プログラムの AMODE 切  
り替え 558

31 ビット・アドレッシング・モード  
558

CICS の制約事項 504

COBOL プログラム相互間の 551, 554

COBOL プログラムと非 COBOL プ  
ログラムの間での 551

JNI サービスへの 743

LINKAGE SECTION 578

OMITTED 引数 577

z/OS UNIX プログラムのもとの

COBOL 542

z/OS プログラムのもとの

COBOL 588

読み取り、レコードの

行順次ファイルから 244

ブロック・サイズ 189

読み取り、VSAM ファイルからのレコー  
ドの

順次 224

動的 224

ランダム 224

予約語テーブル、CICS 代替

概要 511

WORD による指定 435

## [ラ行]

ラージ・ブロック・インターフェース  
(LBI) 190

ライブラリー

パスの指定 445

ライブラリー名

指定されなかった場合の代替 328

ライブラリー名 (続き)  
  使用されない場合 844  
  データ・セット名への相互参照 493  
ライン外の PERFORM 113  
ラッパー、定義 738  
ラッピング、プロシージャ指向プログラムの 738  
乱数の生成 66  
ランタイム・オプション  
  85 COBOL 標準 準拠 348  
  AIXBLD 793  
  ALL31 558  
  CBLOPTS 588  
  CBLPSHPOP 512  
  DEBUG 455  
  ENVAR 341  
  MSGFILE 404  
  POSIX  
    オブジェクト指向アプリケーションでの使用 341  
  DLL 探索順序 596  
TRAP  
  行順次ファイルのクローズ 246  
  ON SIZE ERROR 273  
  QSAM のファイルのクローズ 195  
  VSAM のファイルのクローズ 227  
XPLINK  
  設定 343  
  デフォルトとしては推奨されない 343  
z/OS UNIX のもとの指定 542  
z/OS のもとの指定 588  
リスト  
  組み込みエラー・メッセージ 460  
  短縮リストの生成 464  
  データおよびプロシージャ名相互参照 462  
  テキスト名相互参照 462  
  テキスト名のソート済み相互参照 493  
  プログラム名のソート済み相互参照 493  
  ユーザー提供の行番号 465  
  MAP 出力で使用される用語 472  
  PROCEDURE DIVISION のアセンブラー拡張 474  
リストのヘッダー 5  
リソース・リスト 1007  
リテラル  
  英数字  
    説明 28  
    DBCS の内容での 169  
  国別  
    使用 149  
    説明 28  
  使用 28  
  数字 28

リテラル (続き)  
  定義 28  
  16 進数  
    使用 149  
  DBCS  
    最大長 169  
    使用 168  
    説明 28  
  リトル・エンディアン、ビッグ・エンディアンへの変換 147  
理由コード、XML 構文解析の 654, 823  
利用不能ファイル  
  QSAM 193  
  VSAM 229  
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 334  
  JCL または TSO/E の使用  
    概要 340  
    例 341  
  z/OS UNIX のもとの  
    概要 334  
    例 336  
リンク・ステップ用 c89 コマンド 324  
リンク・リスト処理、例 581  
ループ  
  コーディング 112  
  条件付き 114  
  テーブル内 115  
  明示的に指定した回数だけ実行される 114  
  DO 114  
例外条件  
  CALL 283  
  XML GENERATE 682  
  XML PARSE 656  
例外処理  
  Java との 745  
  XML GENERATE での 682  
  XML PARSE での 654  
レコード  
  順序への編成の影響 171  
  説明 12  
  フォーマット  
    可変長 QSAM 182, 184  
    可変長 VSAM 217  
    形式 D 182, 184, 206  
    形式 F 181, 182, 206  
    形式 S 185, 186  
    形式 U 187, 188, 206  
    形式 V 182, 184, 206  
    固定長 QSAM 181, 182  
    固定長 VSAM 217  
    スパン 185, 186  
    不定形式 187, 188  
  QSAM ASCII テープ 206

レジスター、EXIT コンパイラー・オプションが使用する 842  
列、テーブルの 75  
レベル 88 項目  
  条件式 108  
  スイッチおよびフラグ 109  
  スイッチをオフに設定する例 112  
  スイッチをオンに設定する例 111  
  単一値のテストの例 110  
  複数値のテストの例 110  
レベル番号 470  
ローカル参照、グローバルへの変換 721  
ローカル名 566  
論理レコード  
  可変長形式  
    QSAM のレイアウト 184  
    QSAM 用の要求 182  
    VSAM 用定義 217  
  固定長形式  
    QSAM 用の要求 181  
    VSAM 用定義 217  
  説明 171  
  QSAM、定義 181

## [ワ行]

ワークスペース  
  ソート時の使用 267  
割り込み 767

## [数字]

16 MB 境界  
  パフォーマンス・オプション 789  
  CICS プログラム 502  
  IMS プログラム 502  
16 進数字から変換 133  
16 進数字に変換 133  
16 進数リテラル  
  国別  
    使用 149  
    説明 28  
  通貨符号として 73  
2 進数字から変換 133  
2 進数字に変換 133  
2 進数データ 395  
2 進数データ項目  
  一般的説明 55  
  効率的な使用 55, 781  
  中間結果 811  
  同義語 53  
24 ビット・アドレッシング・モード 43  
31 ビット・アドレッシング・モード 43  
動的呼び出し 558

64 ビットのアドレッシング  
サポートなし 43  
85 COBOL 標準  
チェックポイント 768  
必要なコンパイラー・オプション 348  
必要なランタイム・オプション 348

## A

ACCEPT ステートメント  
入力データの割り当て 38  
CICS のもとで 503  
stdin からの読み取り 38  
ADATA コンパイラー・オプション 350  
ADDRESS OF 特殊レジスター  
CALL ステートメントでの使用 574  
ADEXIT サブオプション、EXIT オプシ  
ョンの  
構文 376  
処理 848  
ADMODE 属性  
マルチスレッド化 611  
adt サフィックス、cob2 での 330  
ADV コンパイラー・オプション 351  
AFP コンパイラー・オプション 351  
パフォーマンスの考慮事項 788  
AIXBLD ランタイム・オプション  
パフォーマンスへの影響 793  
ALL 添え字  
関数引数としてのテーブル・エレメン  
ト 65  
テーブル・エレメントの反復処理 99  
例 99  
ALL31 ランタイム・オプション  
マルチオプションの相互作用 44  
AMODE 切り替えの OFF 558  
ALLOCATE コマンド (TSO)  
コンパイラー・データ・セット 296  
z/OS UNIX ファイル 296  
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 232  
KSDS ファイルの代替キーの識別 214  
AMODE  
および DLL 597  
切り替え  
概要 559  
例 559  
ALL31(OFF) 558  
説明 43  
EXIT モジュール 842  
AMP パラメーター 233  
ANNUITY 組み込み関数 69  
API、UNIX、および POSIX  
呼び出し 545

APOST コンパイラー・オプション 352  
APPLY WRITE-ONLY 節 11  
ARCH コンパイラー・オプション 352  
パフォーマンスの考慮事項 788  
ARITH コンパイラー・オプション  
説明 354  
パフォーマンスの考慮事項 789  
ASCII  
アルファベット、QSAM 206  
ジョブ制御言語 (JCL) 206  
テープ・ファイル、QSAM 206  
レコード形式、QSAM 206  
EBCDIC への変換 132  
XML 文書でサポートされるコード・  
ページ 648  
ASCII ファイル  
CODE-SET 節 15  
DCB 内の OPTCD= パラメーター 15  
ASSIGN 節  
DD 名に対応する 9  
QSAM ファイル 180  
AT END (ファイルの終わり) 句 276  
ATTACH マクロ 298  
AWO コンパイラー・オプション  
説明 355  
パフォーマンスの考慮事項 789  
APPLY-WRITE ONLY 節のパフォー  
マンス 11  
a.out ファイル、cob2 からの 330

## B

Base クラス  
java.lang.Object に相当 699  
java.lang.Object のために使用 698  
BASIS ステートメント 445  
BIT-OF 組み込み関数 133  
BIT-TO-CHAR 組み込み関数 133  
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  
148  
BLOCK CONTAINS 節  
FILE SECTION 記入項目 14  
QSAM ファイル 181, 188, 356  
VSAM ファイルでは無意味 216  
BLOCK0 コンパイラー・オプション  
説明 356  
パフォーマンスの考慮事項 789  
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 340  
z/OS UNIX プログラムの呼び出し  
542  
BUFSIZE コンパイラー・オプション 357  
BY CONTENT 573

BY REFERENCE 573  
BY VALUE  
制約事項 576  
説明 573  
有効なデータ型 576  
BYTE-LENGTH 組み込み関数  
国別データを伴う 138  
使用 134

## C

CALL ID  
常時動的 558  
動的呼び出し 556  
DLL から行う 595  
NODLL の場合 556  
NODYNAM の場合 561  
CALL literal  
静的呼び出し 555  
動的呼び出し 556  
DYNAM の場合 556  
NODLL の場合 555, 556  
NODYNAM の場合 555, 561  
CALL コマンド (TSO) 296  
CALL ステートメント  
エラー処理に関する 283  
オーバーフロー条件 283  
関数ポインター 570  
言語環境プログラム呼び出し可能サー  
ビス (callable services) 800  
その中でのプログラム名の処理 406  
代替入り口点への 570  
例外条件 283  
AMODE の処理 558  
BY CONTENT 573  
BY REFERENCE 573  
BY VALUE  
制約事項 576  
説明 573  
CANCEL を使用した 558  
CICS の制約事項 504  
DYNAM の場合 373  
EXIT オプションのレジスターへの影  
響 842  
ON EXCEPTION を指定した 283  
ON OVERFLOW を指定した 22, 283  
RETURNING 584  
USING 576  
CANCEL ステートメント  
サブプログラムの場合 557  
その中でのプログラム名の処理 406  
動的 CALL を使用した 557  
DLL リンケージでは使用できない 597  
cbl サフィックス、cob2 での 329  
CBL ステートメント  
概要 445

## CBL ステートメント (続き)

コンパイラー・オプションの指定 308

CBLPSHPOP ランタイム・オプション  
512

CBLQDA ランタイム・オプション 193

## CCSID

構文解析対象の XML 文書 631

定義 147

CODEPAGE オプションでの指定 359

Db2 スtring・データ 522

EBCDIC マルチバイト CCSID 360

PARSE ステートメント 631

XML 文書での矛盾 657, 658

XML 文書の 648

CHAR 組み込み関数の例 135

CHKPT キーワード 267

## CICS

組み込みの変換プログラム

概要 508

コンパイラー・オプション 506

ネストされたプログラムの呼び出し  
505

利点 508

このもとでの言語間通信 505

コマンドおよび PROCEDURE

DIVISION 501

コマンド・レベル・インターフェース  
501

システム日付の取得 503

実行するプログラムのコーディング

概要 501

制約事項 502

呼び出し 504

DISPLAY ステートメント 503

I/O 502

SORT ステートメント 511

## 制約事項

オブジェクト指向プログラム 502,  
693

ソート 268

ファイル 6

分離型の変換プログラム 508

16 MB 境界 502

FILE を使用した検証を伴う構文解  
析 642

OUTDD コンパイラー・オプショ  
ン 404

そのもとでのソート

概要 268

制約事項 268

予約語テーブルの変更 511

代替予約語テーブル 511

ネストされたプログラムの呼び出し  
505

パフォーマンス

概要 779

## CICS (続き)

パフォーマンスの考慮事項 512, 793

プログラムの開発 501

分離型の変換プログラム

コンパイラー・オプション 510

使用 510

制約事項 508

ネストされたプログラムの呼び出し  
505

マクロ・レベル・インターフェース  
501

マルチスレッド化環境において 610

CICS HANDLE 512

例 513

LABEL 値 512

CICS オプションを指定したコンパイ  
ル 506

DFHCOMMAREA パラメーター

ネストされたプログラムの呼び出し  
505

別々にコンパイルされたプログラ  
ムの呼び出し 504

DFHEIBLK パラメーター

ネストされたプログラムの呼び出し  
505

別々にコンパイルされたプログラ  
ムの呼び出し 504

ECI 呼び出しと RETURN-CODE 特  
殊レジスター 506

EXIT コンパイラー・オプション 862

NODYNAM コンパイラー・オプショ  
ン 504

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 508

サブオプションの指定 358, 508

使用 506

説明 358

マルチオプションの相互作用 349

CISZ (制御インターバル・サイズ)、パフ  
ォーマンスの考慮事項 238, 793

CKPT キーワード 267

CLASSPATH 環境変数

設定の例 340

説明 544

Java クラスの場所の指定 336

CLOSE ステートメント

行順次ファイル 244

QSAM 192

VSAM 218

## 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 333

オブジェクト指向アプリケーションを  
リンクする場合 334

オプションおよび構文 327

## cob2 コマンド (続き)

これによるコンパイル

概要 324

例 326

説明 327

入出力 329

リンク

概要 324

例 326

DLL 作成用 325

COBJVMINITOPTIONS 環境変数

説明 544

JVM オプションの指定 337

## COBOL

オブジェクト指向

実行 336

バインディング 340

リンク 334

IMS のもとでの 535

JCL または TSO/E を使用したコ  
ンパイル 339

z/OS UNIX のもとでのコンパ  
イル 333

## Java

間の通信 743

アプリケーションの構造化 739

互換 344

実行 336, 340

バインディング 340

リンク 334

IMS のもとでの 535

JCL または TSO/E を使用したコ  
ンパイル 339

z/OS UNIX のもとでのコンパ  
イル 333

COBOL DLL プログラムの呼び出し 598

COBOL 環境の事前初期設定

マルチスレッド化 606

COBOL クライアント

オブジェクト参照引き渡しの例 718

例 733

COBOL 用語 25

COBOL3 変換プログラム・オプション

510

COBOL\_INSTALL\_DIR 環境変数 322

COBOPT 環境変数 322

CODEPAGE コンパイラー・オプション

影響を受けない項目 360

オーバーライドする操作 360

国別リテラルの場合 155

説明 359

DBCS コード・ページ 360

CODE-SET 節 15

COLLATING SEQUENCE 句

国別キーに適用されない 257

COLLATING SEQUENCE 句 (続き)  
PROGRAM COLLATING  
SEQUENCE 節のオーバーライド 7,  
259  
SORT または MERGE での使用 259  
COMMON 属性 5, 563  
COMP (COMPUTATIONAL) 55  
COMPILE コンパイラー・オプション  
構文エラーを見つけるための  
NOCOMPILE の使用 457  
説明 362  
COMPUTATIONAL (COMP) 55  
COMPUTATIONAL-1 (COMP-1)  
パフォーマンスに関するヒント 782  
フォーマット 56  
COMPUTATIONAL-2 (COMP-2)  
パフォーマンスに関するヒント 782  
フォーマット 56  
COMPUTATIONAL-3 (COMP-3)  
説明 56  
COMPUTATIONAL-4 (COMP-4) 55  
COMPUTATIONAL-5 (COMP-5) 55  
COMPUTE ステートメント  
コーディングが容易 63  
算術結果の割り当て 37  
COMP-1 (COMPUTATIONAL-1)  
パフォーマンスに関するヒント 782  
フォーマット 56  
COMP-2 (COMPUTATIONAL-2)  
パフォーマンスに関するヒント 782  
フォーマット 56  
COMP-3 (COMPUTATIONAL-3) 56  
COMP-4 (COMPUTATIONAL-4) 55  
COMP-5 (COMPUTATIONAL-5) 55  
CONFIGURATION SECTION 6  
CONTENT-CHARACTERS XML イベント  
セグメントの構文解析時 645  
例 670  
CONTINUE ステートメント 104  
CONTROL ステートメント 445  
CONVERTING 句 (INSPECT) の例 129  
COPY ステートメント  
説明 445  
ネストされた 796, 844  
例 796  
z/OS UNIXの考慮事項 446  
z/OSの考慮事項 304  
COPYLOC コンパイラー・オプション  
363  
COPYRIGHT コンパイラー・オプション  
364  
COUNT IN 句  
UNSTRING 120  
XML GENERATE 682  
CRP (ファイル位置標識) 220, 224

CURRENCY コンパイラー・オプション  
365  
CURRENT-DATE 組み込み関数  
例 68  
CICS のもとで 504  
C/C++ プログラム  
マルチスレッド化 610  
COBOL DLL を使用した 600

## D

D フォーマット・レコード  
要求 182  
レイアウト 184  
DASD (直接アクセス・ストレージ・デバ  
イス) 238  
DATA DIVISION  
グループ・レベルの USAGE  
NATIONAL 節 152  
グループ・レベルの USAGE 節 27  
リスト 464  
OCCURS 節 75  
USAGE IS INDEX 節 80  
DATA RECORDS 節 15  
DATA コンパイラー・オプション  
説明 366  
データの受け渡し時 46  
データ・ロケーションへの影響 46  
パフォーマンスの考慮事項 789  
マルチオプションの相互作用 44  
DATE-COMPILED 段落 3  
DATE-OF-INTEGER 組み込み関数 69  
Db2  
コーディングに関する考慮事項 515  
コプロセッサ  
概要 515  
推奨コンパイラー・オプション  
SQLCCSID 524  
ストリング・データの CCSID の  
決定 522  
データベース要求モジュール  
(DBRM) 516, 521  
プリコンパイラーとの相違 524  
SQL INCLUDE の使用 517  
SQL コンパイラー・オプションで  
使用可能にする 520  
プリコンパイラー  
コプロセッサとの相違 524  
使用 516  
推奨コンパイラー・オプション  
NOSQLCCSID 524  
ホスト変数用のコード・ページの指  
定 518  
CICS または CAF の場合の  
NODYNAM コンパイラー・オプシ  
ョン 527

Db2 (続き)  
SQL コンパイラー・オプション 520  
SQL ステートメント  
概要 515  
国別 10 進数データの使用 519  
コーディング 517  
バイナリー・データの使用 520  
文字データの使用 518  
戻りコード 520  
CCSID の決定 522  
SQL DECLARE 518  
SQL INCLUDE 517  
SQLCCSID コンパイラー・オプション  
522  
TSO または IMS の場合の DYNAM  
コンパイラー・オプション 527  
Db2 プリコンパイラー  
使用 516  
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  
333, 339  
説明 367  
マルチオプションの相互作用 349  
Java とのインターオペラビリティの  
ための 333, 339  
DBCS データ  
エンコードおよびストレージ 155  
これを伴う MOVE ステートメント  
35  
宣言する 168  
テスト 169  
比較する  
国別と 167  
表記 817  
変換  
英数字との間の 817  
国別への、概要 170  
IGZCD2A による英数字への 820  
リテラル  
最大長 169  
使用 168  
説明 28  
DBCS 比較 108  
dbg サフィックス、cob2 での 330  
dbrm サフィックス、cob2 での 330  
DBRM データ・セット  
説明 516  
定義 521  
DBRM に使用されるロケーションを指定  
するための -dbrmlib cob2 オプション  
327  
DBRMLIB DD DD ステートメント 516,  
521  
DCB 191  
DD 制御ステートメント  
行順次ファイルの割り振り 243

DD 制御ステートメント (続き)  
ソート・データ・セットの定義 255  
ファイルの定義 9  
マージ・データ・セットの定義 255  
AMP パラメーター 233  
ASCII テープ・ファイル 206  
DBRMLIB 521  
DCB はデータ・セット・ラベルを指  
定変更する 198  
JAVAERR 341  
JAVAIN 341  
JAVAOUT 341  
QSAM ファイルの作成 197, 199  
RLS パラメーター 235  
SYSADATA 306  
SYSDEBUG 306  
SYSIN 303  
SYSJAVA 306  
SYSLIB 304  
SYSLIN 305  
SYSMDECK 307  
SYSOPTF 303  
SYSPRINT 304  
SYSPUNCH 305  
DD 名の定義 9  
DECK コンパイラー・オプション 367  
DEFINE コンパイラー・オプション 368  
dek サフィックス、cob2 での 330  
DELETE ステートメント  
コンパイラー指示 448  
マルチスレッド化シリアライゼーシ  
ョン 607  
VSAM、コーディング 218  
DEPENDING ON 節 183, 217  
DFHCOMMAREA パラメーター  
ネストされた CICS プログラムの呼び  
出し 505  
別個にコンパイルされた CICS プログ  
ラムの呼び出し 504  
DFHEIBLK パラメーター  
ネストされた CICS プログラムの呼び  
出し 505  
別個にコンパイルされた CICS プログ  
ラムの呼び出し 504  
DFSORT  
データ・セットの定義 255  
RETURN ステートメントのエラー・  
メッセージ 253  
DIAGTRUNC コンパイラー・オプション  
369  
DISPLAY (USAGE IS)  
エンコードおよびストレージ 155  
外部 10 進数 54  
浮動小数点 54  
DISPLAY ステートメント  
行送りの抑止 40

DISPLAY ステートメント (続き)  
システム論理出力装置での表示 40  
出力の送信 403  
データ値の表示 39  
デバッグでの使用 452  
CICS のもとで 503  
OUTDD との相互作用 40  
STDOUT または STDERR への書き込  
み 40  
DISPLAY-1 (USAGE IS)  
エンコードおよびストレージ 155  
DISPLAY-OF 組み込み関数  
ギリシャ語データでの例 159  
使用 157  
中国語データでの例 165  
UTF-8 データでの例 159  
XML 文書での 650  
DISPSIGN コンパイラー・オプション  
370  
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 340  
例 341  
リンク  
オブジェクト指向アプリケーション  
の準備 334  
例 336  
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 340  
例 341  
リンク  
オブジェクト指向アプリケーション  
の準備 334  
例 336  
EBCDIC サービスで 755  
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  
333, 339  
説明 371  
マルチオプションの相互作用 349  
Java とのインターオペラビリティの  
ための 333, 339  
DLL (ダイナミック・リンク・ライブラリ  
を参照) 591  
DO ループ 114  
do-until 114  
do-while 114  
DSA メモリー・マップ 474  
DUMP コンパイラー・オプション  
出力 310  
説明 372  
DYNAM コンパイラー・オプション  
説明 373

DYNAM コンパイラー・オプション (続  
き)  
動的呼び出しでの 556  
パフォーマンスの考慮事項 789  
マルチオプションの相互作用 349  
TSO または IMS、および Db2 の場  
合 527

## E

E レベルのエラー・メッセージ 318, 460  
EBCDIC  
ASCII への変換 132  
DBCS でサポートされているマルチバ  
イト CCSID 360  
JNI サービス 754  
XML 文書でサポートされるコード・  
ページ 648  
ECI 呼び出しと RETURN-CODE 特殊レ  
ジスター 506  
EGCS 980  
EJECT ステートメント 448  
END-OF-INPUT XML イベント  
セグメントの構文解析時 645  
例 670  
ENTER ステートメント 448  
ENTRY ステートメント  
その中でのプログラム名の処理 406  
代替入り口での 568  
ENVAR ランタイム・オプション 341  
ENVIRONMENT DIVISION  
インスタンス・メソッド 704  
行順次ファイルについての記入項目  
241  
クライアント 713  
クラス 699  
項目、プログラム初期設定コード 476  
サブクラス 726  
シグニチャー情報バイト 476  
照合シーケンスのコーディング 7  
説明 6  
CONFIGURATION SECTION 6  
INPUT-OUTPUT SECTION 6  
QSAM ファイルについての記入項目  
180  
VSAM ファイルについての記入項目  
212  
ERRMSG、エラー・メッセージのリスト  
の生成 317  
ESDS (入力順データ・セット)  
ファイル・アクセス・モード 215  
編成 212  
EVALUATE ステートメント  
いくつかの条件をテストする例 107  
ケース構造 105  
コーディング 105

EVALUATE ステートメント (続き)

構造化プログラミング 780

ネストされた IF と対比 107

パフォーマンス 106

複数値のテストの例 110, 111

複数条件のテストに使用 103

複数の WHEN 句の例 107

THRU 句の例 106

EXCEPTION XML イベント 656

EXCEPTION/ERROR 宣言

行順次エラー処理 246

説明 276

ファイル状況キー 278

QSAM エラー処理 196

VSAM エラー処理 228

EXEC 制御ステートメントの RD パラメーター 771

EXIT PROGRAM ステートメント

サブプログラムにおける 552

マルチスレッド化 552

EXIT コンパイラー・オプション

使用 374

説明 374

ユーザー出口作業域 841

レジスターの使用 842

MSGEXIT サブオプション 850

SQL および CICS ステートメントについての考慮事項 862

EXPORTALL コンパイラー・オプション

説明 377

マルチオプションの相互作用 349

DLL に関する考慮事項 592

EXTERNAL 節

データ項目の 584

ファイルの共用 13, 585

ファイルの場合の例 585

EXTERNAL データ

共用 584

## F

F フォーマット・レコード

要求 181

レイアウト 182

FACTORY 段落

ファクトリー・データ 730

ファクトリー・メソッド 730

FASTSORT コンパイラー・オプション

説明 377

ソート・パフォーマンスの向上 261, 789

通知メッセージ 261

要件

ソート入出力ファイル 261

JCL 261

QSAM 262

FASTSORT コンパイラー・オプション (続き)

要件 (続き)

VSAM 263

FD (ファイル記述) 項目 14

FILE SECTION

説明 12

レコードの説明 12

BLOCK CONTAINS 節 14

CODE-SET 節 15

DATA RECORDS 節 15

EXTERNAL 節 13

FD 記入項目 14

GLOBAL 節 13

LABEL RECORDS 節 14

LINAGE 節 15

OMITTED 15

RECORD CONTAINS 節 14

RECORD IS VARYING 14

RECORDING MODE 節 15

VALUE OF 15

FILE STATUS 節

行順次エラー処理 246

使用 277

説明 177

例 282

NOFASTSORT エラー処理 264

QSAM エラー処理 196

VSAM エラー処理 228

VSAM 状況コードを持つ 279

FILE-CONTROL 段落

項目の例 7

FD 項目との関係 9

FIPS メッセージ

カテゴリー 853

FLAGSTD コンパイラー・オプション 379

FLAG コンパイラー・オプション

コンパイラー出力 461

使用 460

説明 378

FLAGSTD コンパイラー・オプション 379

マルチオプションの相互作用 349

## G

GB 18030 データ

国別と間の変換 164

処理 164

get メソッドおよび set メソッド 708

GETMAIN のアドレスの保管 841

GLOBAL 節、ファイルに対する 13, 18

GOBACK ステートメント

サブプログラムにおける 553

マルチスレッド化 553

GOBACK ステートメント (続き)

メインプログラムにおける 553

GROUP-USAGE NATIONAL 節

国別グループの初期化 34

国別グループの宣言の例 27

国別グループの定義 152

テーブルの定義 76

Java との通信 749

## 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 46

マルチオプションの相互作用 44

HEX-OF 組み込み関数 133

HEX-TO-CHAR 組み込み関数 133

HGPR コンパイラー・オプション 381

パフォーマンスの考慮事項 789

## I

I レベルのメッセージ 318, 460

IDENTIFICATION DIVISION

エラー 3

クライアント 711

クラス 698

コーディング 3

サブクラス 725

必要な段落 3

メソッド 703

リストのヘッダーの例 5

CLASS-ID 段落 698, 725

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 103

ネストされた 104

複数条件の場合に、代わりに

EVALUATE を使用 104

NULL ブランチを伴う 103

IGZCA2D サービス・ルーチン 818

IGZCD2A サービス・ルーチン 820

igzcljava.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 340

例 341

リンク

オブジェクト指向アプリケーションの準備 334

例 336

IGZEOPT モジュール

マルチスレッド化 611



IGZETUN モジュール  
  マルチスレッド化 611  
IGZSRCD データ・セット 266  
IMS  
  コプロセッサー  
    概要 529  
  コンパイルおよびリンク 534  
  のもとでのプログラムのコーディング  
    概要 529  
    制約事項 6, 529  
  パフォーマンスの考慮事項 793  
  COBOL-Java インターオペラビリティ  
    ー  
      データベースへのアクセス 537  
      トランザクションの同期 537  
      メッセージ 537  
      AIB の使用 538  
      COBOL からの Java メソッドの呼  
      び出し 536  
      EXEC SQL の制約事項 537  
      Java からの COBOL メソッドの呼  
      び出し 535  
      STOP RUN 537  
  IMS のもとでの EXEC SQL の使用  
    537  
  SQLIMS コンパイラー・オプション  
    532  
  SQLIMS ステートメント 531  
    文字データの使用 531  
    戻りコード 532  
    SQLIMS INCLUDE 531  
IMS SQL  
  コプロセッサー 531  
INEXIT サブオプション、EXIT オプショ  
  ンの  
    構文 375  
    処理 842  
INITCHECK コンパイラー・オプション  
  説明 382  
  無効な COBOL データ 458  
INITIAL コンパイラー・オプション  
  説明 383  
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 554  
INITIAL 属性  
  代わりに動的呼び出し および  
    CANCEL を使用 558  
  サブプログラムへの影響 555, 556  
INITIALIZE ステートメント  
  国別グループ値のロード 34  
  グループ値のロード 33  
  テーブルの値のロード 82  
  デバッグ用の使用 454  
  例 30

INITIALIZE ステートメント (続き)  
  REPLACING 句 82  
INLINE コンパイラー・オプション  
  説明 384  
INPUT-OUTPUT SECTION 6  
INSERT ステートメント 448  
INSPECT ステートメント  
  使用 128  
  例 128  
  UTF-8 データでの使用を避ける 653  
INTDATE コンパイラー・オプション  
  カレンダー開始日への影響 67  
  説明 385  
INTEGER 組み込み関数の例 127  
INTEGER-OF-DATE 組み込み関数 68  
INTEGER-PART 組み込み関数 127  
INVALID KEY 句  
  説明 281  
  例 282  
INVOKE ステートメント  
  オブジェクトの作成に使用 721  
  メソッドの呼び出しに使用 716  
  ON EXCEPTION を指定した 717,  
    733  
  PROCEDURE DIVISION  
    RETURNING で 583  
  RETURNING 句 720  
  USING 句 718  
ISAM データ・セット、VSAM KSDS デ  
  ータ・セットと類似 209  
ISPF (対話式システム生産性向上機能)  
  946

## J

J2EE クライアント  
  実行 338  
  例 756  
Java  
  インターオペラビリティ 743  
  オブジェクト配列 750  
  クラス型 749  
  グローバル参照  
    受け渡し 746  
    オブジェクト 746  
    管理 746  
  JNI サービス 747  
  ストリング  
    宣言する 750  
    取り扱い 753  
  ストリング配列 750  
  相互運用可能データ型、コーディング  
    749  
  データ共用 748

Java (続き)  
  と COBOL  
    JCL または TSO/E を使用したコ  
      ンパイル 339  
  長い配列 750  
  二重配列 751  
  バイト配列 750  
  配列  
    宣言する 750  
    取り扱い 751  
    例 753  
  配列クラス 748  
  パッチ COBOL プログラムから呼び  
    出す 760  
  ブール配列 750  
  浮動配列 751  
  短い配列 750  
  メソッド  
    アクセス制御 748  
  文字配列 750  
  例  
    整数配列の処理 753  
    例外処理 746  
    J2EE クライアント 756  
  例外  
    処理 745  
    例 746  
    catch 745  
    throw 745  
  ローカル参照  
    受け渡し 746  
    オブジェクト 746  
    解放 747  
    管理 746  
    削除 747  
    保管 747  
    マルチスレッド化ごと 747  
  JNI サービス 747  
boolean 型 749  
byte 型 749  
char 型 749  
COBOL  
  間の通信 743  
  アプリケーションの構造化 739  
  互換 344  
  実行 336, 340  
  バインディング 340  
  リンク 334  
  z/OS UNIX のもとでのコンパ  
    イル 333  
COBOL での実行  
  JCL または TSO/E の使用 340  
  XPLINK リンケージ 343  
  z/OS UNIX のもとでの 336  
double 型 749  
float 型 749

Java (続き)

- int 型 749
- int 配列 750
- jstring クラス 748
- long 型 749
- short 型 749

Java 仮想マシン

- オブジェクト参照 746
- 初期化 337
- 例外 745

Java との相互運用が可能なデータ型 749

javac コマンド

- Java 6 以降用に再コンパイル 344
- Java クラス定義のコンパイル 333

JAVAERR データ・セット 341

JAVAIN データ・セット 341

JAVAOUT データ・セット 341

java.lang.Object

- Base として参照 698

JCL

- オブジェクト指向アプリケーションで使用する 339
  - 例 341
- カタログ式プロシージャ 288
- 行順次ファイル用 243
- コンパイル用 288
- ソート用 255
- チェックポイント/再始動の例 774
- マージ用 255
- ASCII テープ・ファイル 206
- FASTSORT の要件 261
- QSAM ファイル用 198
- VSAM データ・セット用 233
- z/OS UNIX ファイル・システムでのコンパイル用 290

JNI

- オブジェクト参照の比較 715
- クラス・オブジェクト参照の取得 744
- 構造環境 743
  - アドレス可能度的場合 744
- サービスへアクセス 743
- 使用した場合の制限 744
- 例外取り扱いサービス 745
- ローカル参照からグローバルへの変換 721
- EBCDIC サービス 754
- Java ストリング・サービス 753
- Java 配列サービス 751
- Unicode サービス 753
- UTF-8 サービス 756

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 744

JNINativeInterface

- 構造環境 743
- JNI.cpy 743

JNI.cpy

- コンパイル用 334
- リスト 865
- JNINativeInterface の場合 743

JOB 制御ステートメントの RD パラメーター 771

JSON 出力

- 生成
  - 概要 625
- JSON 出力の生成 625
- JSON 入力
  - 構文解析
    - 概要 617
    - 例 621
- JSON 入力の処理 617
- JSON の生成
  - 概要 625
- JSON パーサー
  - 概要 617
  - 例 621
- JSON 文書
  - 構文解析
    - 概要 617
    - 説明 617
    - 例 621
  - 生成
    - 概要 625
- JSON 文書の構文解析 617
- JSON 例外コード
  - 構文解析 835
  - 生成 835
- JSON-CODE 特殊レジスター
  - 構文解析の例外コード 835
  - 生成の例外コード 835
- JSON-STATUS 特殊レジスター
  - 構文解析の非例外理由コード 835
- jstring Java クラス 748

JZOS

- 例 760
- Java Batch Launcher and Toolkit for z/OS 763

## K

KSDS (キー順データ・セット)

- ファイル・アクセス・モード 215
- 編成 213

## 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 385

LIB (ラージ・ブロック・インターフェース) 190

LENGTH OF 特殊レジスター

- 受け渡し 574
- 使用 138

LENGTH 組み込み関数

- 可変長の結果 136
- 国別データを伴う 138
- 使用 134
- 例 68, 138
- LENGTH OF 特殊レジスターと比較 138

LIBEXIT サブオプション、EXIT オプションの

- 構文 375
- 処理 844

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 340
  - 例 341
- リンク
  - オブジェクト指向アプリケーションの準備 334
  - 例 336
- EBCDIC サービスで 755

LIBPATH 環境変数

- 設定の例 340
- 説明 544
- COBOL クラスの場所の指定 336

library

- 定義 304
- ディレクトリー記入項目 298
- BASIS 304
- COPY 304

library-name 環境変数 322

LINECOUNT コンパイラー・オプション 386

LINK マクロ 298

LINKAGE SECTION

- コーディング 578
- 再帰呼び出し 18
- パラメーターを記述するための 576
- THREAD オプションを指定した 18

LIST コンパイラー・オプション

- 外部シンボル・セクション 490
- コンパイラー出力 476, 483
- 出力で使用する記号 473
- 出力の取得 464
- 出力の読み取り 474
- 静的マップ・セクション 488
- 説明 387
- ソース・プログラムのアセンブラー・コード 474
- タイム・スタンプおよびバージョン情報の例 485

LIST コンパイラー・オプション (続き)  
定数域セクション 489  
特殊レジスター・テーブル 490  
ベース・ロケーター・テーブル 489  
マルチオプションの相互作用 349  
DSA メモリー・マップ 474, 491  
MD5 シグニチャーの例 484  
OFFSET オプションとの対立 464  
LOCAL-STORAGE SECTION  
クライアント 714, 715  
ロケーションの決定 46  
WORKING-STORAGE との比較  
概要 15  
例 16  
OO クライアント 715  
LOG 組み込み関数 69  
LOWER-CASE 組み込み関数 130  
lst サフィックス、cob2 での 330

## M

MAP コンパイラー・オプション  
組み込みマップ要約 464  
出力で使用する記号 473  
出力で使用する用語 472  
使用 462, 464  
説明 388  
データ項目と相対アドレス 310  
ネストされたプログラム・マップ 464  
例 474  
例 469, 474  
MAP 出力で使用する用語 472  
MAX 組み込み関数  
関数の例 68  
使用 135  
テーブル計算の例 99  
MAXPCF コンパイラー・オプション 389  
MDECK コンパイラー・オプション  
説明 391  
MEAN 組み込み関数  
テーブル計算の例 99  
統計計算の例 70  
MEDIAN 組み込み関数  
テーブル計算の例 99  
統計計算の例 70  
MERGE ステートメント  
概要 247  
制約事項 247  
説明 256  
ASCENDING|DESCENDING KEY  
句 257  
COLLATING SEQUENCE 句 7, 259  
GIVING 句 256  
USING 句 256  
METHOD-ID 段落 703

MIN 組み込み関数  
使用 135  
例 127  
MOVE ステートメント  
基本受信項目を伴う 35  
国別項目を伴う 35  
国別データへの変換 156  
グループ移動と基本移動の対比 36,  
153  
グループ受信項目を伴う 36  
算術結果の割り当て 37  
送信項目および受信項目の長さに対する ODO の影響 88  
CORRESPONDING 36  
MSGEXIT サブオプション、EXIT オプシ  
ョンの  
構文 376  
コンパイル戻りコードへの影響 855  
処理 850  
メッセージ重大度レベル 853  
ユーザー出口の例 855  
MSGFILE ランタイム・オプション 404

## N

N 区切り文字、国別または DBCS リテラ  
ル用の 28  
NAME コンパイラー・オプション  
使用 3  
説明 392  
NAMESPACE-DECLARATION XML イ  
ベント 639  
NATIONAL (USAGE IS)  
外部 10 進数 54  
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 159  
使用 157  
中国語データでの例 165  
UTF-8 データでの例 159  
XML 文書での 650  
NEXT SENTENCE ステートメント 104  
NOBLCARD 変換プログラム・オプショ  
ン 510  
NOCOMPILE コンパイラー・オプション  
構文エラーの検出に使用 457  
NODLL コンパイラー・オプション  
静的呼び出しでの 555  
動的呼び出しでの 556  
NODYNAM コンパイラー・オプション  
ストアド・プロシージャーの場合  
527  
静的および動的呼び出しでの 561  
静的呼び出しでの 555  
CICS のもとで 504

NODYNAM コンパイラー・オプション  
(続き)  
CICS または CAF、および Db2 の場  
合 527  
NOFASTSORT コンパイラー・オプション  
264, 267  
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 349  
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 524  
NSYMBOL コンパイラー・オプション  
国別データ項目の 148  
国別リテラルの場合 149  
説明 393  
マルチオプションの相互作用 349  
DBCS リテラル用の 149  
N リテラルへの影響 28  
NULL ブランチ 103  
NUMBER コンパイラー・オプション  
説明 394  
デバッグ用 465  
NUMCHECK コンパイラー・オプション  
395  
無効な COBOL データ 458  
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 61  
NUMPROC コンパイラー・オプション  
説明 398  
パフォーマンスの考慮事項 790  
符号処理への影響 60  
NUMCLS による影響 61  
NUMVAL 組み込み関数  
説明 131  
NUMVAL-C 組み込み関数  
説明 131  
例 68  
NX 区切り文字、国別リテラル用の 28

## O

o サフィックス、cob2 での 329, 330  
OBJECT コンパイラー・オプション  
説明 399  
マルチオプションの相互作用 349  
OBJECT 段落  
インスタンス・データ 701, 726  
インスタンス・メソッド 702  
OBJECT-COMPUTER 段落 6  
OCCURS DEPENDING ON (ODO) 節  
可変長テーブル作成用 87  
可変長レコード  
QSAM 183  
VSAM 217  
最適化 784  
単純 87

OCCURS DEPENDING ON (ODO) 節  
(続き)

複合体 91

ODO エLEMENTの初期化 90

ODO オブジェクト 87

ODO サブジェクト 87

OCCURS INDEXED BY 節による指標の  
作成 80

OCCURS 節

指標作成用の INDEXED BY 句 80

多次元テーブルを作成するためにネス  
トされた 76

テーブルの定義 75

テーブル・ELEMENTの定義 76

レベル 01 項目では使用できない 76

ASCENDING|DESCENDING KEY  
句

テーブル・ELEMENTの順序の指定  
76

二分探索に必要 97

例 97

ODO オブジェクト 87

ODO サブジェクト 87

OFFSET コンパイラー・オプション

出力 495

説明 399

マルチオプションの相互作用 349

OMITTED 節、FILE SECTION 15

OMITTED パラメーター 800

ON EXCEPTION 句

INVOKE ステートメント 717, 733

OO アプリケーションの XPLINK リンケ  
ージ規約 343

OO アプリケーションの構造化 739

OPEN ステートメント

行順次ファイル 243

ファイル状況キー 277

ファイルの可用性 192, 220, 244

マルチスレッド化シリアライゼーシ  
ョン 607

QSAM ファイル 191

VSAM ファイル 218

OPTFILE コンパイラー・オプション 400

OPTIMIZE コンパイラー・オプション

説明 402

パフォーマンスの考慮事項 786, 790

パラメーター引き渡しの影響 577

ORD 組み込み関数の例 135

ORD-MAX 組み込み関数

使用 136

テーブル計算の例 99

ORD-MIN 組み込み関数 136

OUTDD コンパイラー・オプション

説明 403

割り振られない DD 40

DISPLAY との対話 40

## P

PARMCHECK コンパイラー・オプショ  
ン 404

無効な COBOL プログラム 458

PASSWORD 節 228

PATH 環境変数

設定の例 340

説明 544

PERFORM ステートメント

インライン 113

行外 113

指標を変更するための 81

テーブル用の

指標付けを使用した例 86

添え字付けを使用した例 85

明示的に指定した回数だけ実行される  
114

ループのコーディング 112

TEST AFTER 114

TEST BEFORE 114

THRU 115

TIMES 114

UNTIL 114

VARYING 115

VARYING WITH TEST AFTER 115

WITH TEST AFTER . . . UNTIL 114

WITH TEST BEFORE . . .

UNTIL 114

PGMNAME コンパイラー・オプション

説明 405

COMPAT サブオプション 406

LONGMIXED サブオプション 407

LONGUPPER サブオプション 406

PICTURE 節

国別データを表す N 148

国別編集データ 148

使用される記号の判別 365

数字編集データ 148

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 61

PL/I タスク

COBOL との 610

POSIX ランタイム・オプション 610

POSIX

スレッド 610

API の呼び出し 545

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 341

DLL 探索順序への影響 596

PRESENT-VALUE 組み込み関数 69

PROCESS (CBL) ステートメント

概要 448

PROCESS (CBL) ステートメント (続き)

コンパイラー・オプションの指定 308

パッチ・コンパイル 313

矛盾するオプション 349

優先順位

パッチで 313

z/OS UNIX のもとでの 323

z/OS のもとでの 307

PROGRAM COLLATING SEQUENCE  
節

国別または DBCS オペランドに影響  
を与えない 8

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  
ライド 259

COLLATING SEQUENCE 句によるオ  
ーバライド 7

PROGRAM-ID 段落

コーディング 3

COMMON 属性 5

INITIAL 節 5

PRTEXIT サブオプション、EXIT オプシ  
ョンの

構文 375

処理 847

## Q

QSAM ファイル

オープン 192

クローズ 195

検索 200

再オープン防止のためのクローズ 192  
処理

概要 179

既存ファイル 201

逆順の 192

新規ファイル 202

z/OS UNIX ファイル 205

ストライプ拡張形式 203

属性 201

テープのパフォーマンス 190

入出力エラー処理 196, 273

入出力ステートメント 191

バッファの入手 204

パフォーマンスを向上させるブロック  
化 188, 356

ファイルの更新 194

プリンターへの書き込み 194

ブロック化、レコードの 188, 204

ブロック・サイズ 188, 356

レコードの置換 194

レコードの追加 194

ASCII テープ・ファイル 206

ASSIGN 節 180

BLOCK CONTAINS 節 188, 356

QSAM ファイル (続き)  
DATA DIVISION 記入項目 180  
ENVIRONMENT DIVISION 記入項目 180  
FASTSORT のもとで同じ入出力ファイルを使用 262  
FASTSORT の要件 262  
z/OS のもとでの  
環境変数 197  
ジョブ制御言語 (JCL) 198  
定義 197, 199  
ファイルの可用性 193  
ファイルの作成 197, 199  
DD ステートメント 197, 199  
QUALIFY コンパイラー・オプション 408  
QUOTE コンパイラー・オプション 352

## R

RANGE 組み込み関数  
テーブル計算の例 99  
統計計算の例 70  
RD パラメーター、JOB または EXEC ステートメント 771  
READ NEXT ステートメント 218  
READ ステートメント  
行順次ファイル 243  
マルチスレッド化シリアルライゼーション 607  
AT END 句 276  
QSAM 191  
VSAM 218  
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 213  
RECORDING MODE 節  
可変長レコード、QSAM 182, 184  
固定長レコード、QSAM 181  
レコード形式の指定 180  
QSAM ファイル 15  
REDEFINES 節を使用して、レコードをテーブルに作成 84  
RELEASE FROM ステートメント  
例 251  
RELEASE との比較 251  
RELEASE ステートメント  
RELEASE FROM との比較 251  
SORT での 251  
REM 組み込み関数 69  
RENT コンパイラー・オプション  
アドレス可能性への影響 45  
オブジェクト指向 COBOL のための 333, 339  
説明 409  
RENT コンパイラー・オプション (続き)  
データの受け渡し時 46  
パフォーマンスの考慮事項 791  
マルチオプションの相互作用 44, 349  
DLL 用 592  
IMS 用 534  
Java とのインターオペラビリティのための 333, 339  
REPLACE ステートメント  
説明 448  
Db2 に関する考慮事項 525  
REPLACING 句 (INSPECT) の例 128  
REPOSITORY 段落  
クライアント 713  
クラス 699  
コーディング 6  
サブクラス 726  
RERUN 節  
チェックポイント・リスタート 267  
RETURN ステートメント  
出力プロシージャで必要 252  
INTO 句を指定した 253  
RETURNING 句  
CALL ステートメント 584  
INVOKE ステートメント 720  
PROCEDURE DIVISION ヘッダー 583, 706  
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 583  
オペレーティング・システムに制御権が戻される時 583  
言語環境プログラム・サービスへの呼び出し 800  
プログラム間での戻りコードの共用 583  
CICS ECI 呼び出し 506  
Db2 についての考慮事項 520  
INVOKE で設定しない 717  
REVERSE 組み込み関数 131  
REWRITE ステートメント  
マルチスレッド化シリアルライゼーション 607  
QSAM 192  
VSAM 218  
RLS パラメーター 235  
RMODE  
説明 43  
EXIT モジュール 842  
RMODE コンパイラー・オプション  
アドレス可能性への影響 44  
説明 410  
データの受け渡し時 46  
パフォーマンスの考慮事項 791  
マルチオプションの相互作用 44  
ROUNDED 句 806

RRDS (相対レコード・データ・セット)  
可変長レコード 211, 215  
可変長レコードのシミュレート 215  
固定長レコード 210, 215  
パフォーマンスの考慮事項 238  
ファイル・アクセス・モード 215  
編成 214  
RULES コンパイラー・オプション  
説明 411

## S

S 形式レコード  
概要 186  
要求 185  
レイアウト 186  
S レベルのエラー・メッセージ 318, 460  
SD (ソート記述) 項目の例 250  
SEARCH ALL ステートメント  
指標を変更するための 81  
テーブルは順序付けが必要 97  
二分探索 97  
例 97  
SEARCH ステートメント  
指標を変更するための 81  
逐次探索 95  
テーブルの複数のレベルを検索するためのネスト 95  
例 96  
SELECT OPTIONAL  
QSAM 193  
VSAM 221  
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 715  
SEQUENCE コンパイラー・オプション 414  
SERVICE LABEL ステートメント 448  
SERVICE コンパイラー・オプション 415  
SET 条件名 TO TRUE ステートメント  
スイッチおよびフラグ 111  
例 113, 115  
SET ステートメント  
オブジェクト参照用 716  
関数ポインター・データ項目用 568  
指標データ項目を変更するための 80  
指標を変更するための 81  
条件設定用の、例 111  
その中でのプログラム名の処理 406  
デバッグ用の使用 454  
プロシージャ・ポインター・データ項目用 568  
SIGN IS SEPARATE 節  
移植性 50

SIGN IS SEPARATE 節 (続き)	SQL ステートメント (続き)	STRING ステートメント
印刷 50	SQL INCLUDE 517	オーバーフロー条件 272
行順次ファイル用 245	SQLCA	使用 117
符号付き国別 10 進数データに必要 50	Db2 からの戻りコード 520	例 118
SORT ステートメント	SQL ステートメントを使用するプログラ ムについて宣言 517	DBCS データを使用する 817
概要 247	SQLIMS ステートメントを使用するプ ログラムについて宣言 530	SUM 組み込み関数、テーブル計算の例 99
制約事項 247	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 522	SUPER 721
説明 256	説明 417	SUPPRESS コンパイラー・オプション 422
ASCENDING DESCENDING KEY 句 257	パフォーマンスの考慮事項 524	SYMBOLIC CHARACTERS 節 9
CICS アプリケーションの場合の制約 事項 268	Db2 コプロセッサで推奨する 524	SYSABEND ファイル 説明 301
CICS のもとで 268	SQLIMS コンパイラー・オプション 418	SYSADATA
予約語テーブルの変更 511	使用 532	出力 350
COLLATING SEQUENCE 句 7, 259	制約事項	ファイルの作成 306
GIVING 句 256	バッチでのコンパイル 532	レコード、出口モジュール 848
USING 句 256	SQLIMS ステートメント	SYSADATA ファイル
SORTCKPT DD ステートメント 267	コーディング	説明 301
SORT-CONTROL 特殊レジスター 265	概要 530	ファイル内容 871
SORT-CORE-SIZE 特殊レジスター 265	EXIT コンパイラー・オプション 862	例 873
SORT-FILE-SIZE 特殊レジスター 265	SQLIMS INCLUDE 531	レコード記述 874
SORT-MESSAGE 特殊レジスター 265	SQRT 組み込み関数 69	レコード・タイプ 872
SORT-MODE-SIZE 特殊レジスター 265	SSRANGE コンパイラー・オプション	SYSDEBUG データ・セット
SORT-RETURN 特殊レジスター 265	参照変更 125	使用 426
ソートまたはマージの終了 260	使用 459	定義 306
ソートまたはマージの成功の判断 260	説明 420	SYSDEBUG ファイル
SOURCE および NUMBER 出力の例 468	パフォーマンスの考慮事項 791	説明 301
SOURCE コンパイラー・オプション	STACK ランタイム・オプション	SYSIN データ・セット
出力の取得 464	データ・ロケーションへの影響 46	説明 300
説明 415	マルチオプションの相互作用 44	定義 303
SOURCE-COMPUTER 段落 6	STANDARD 節、FD 記入項目 15	SYSJAVA ファイル
SPACE コンパイラー・オプション 416	START ステートメント	説明 301
SPECIAL-NAMES 段落	マルチスレッド化シリアライゼーショ ン 607	定義 306
コーディング 6	VSAM 218	SYSLIB 環境変数
QSAM ファイル 206	stderr	説明 322
SQL コンパイラー・オプション	行送りの制御 40	JNI.cpy の場所の指定 334
使用 520	DISPLAY による送信 40	SYSLIB データ・セット
制約事項	DISPLAY の設定 544	使用されない場合 844
オブジェクト指向プログラム 693	stdin	説明 300
バッチでのコンパイル 521	ACCEPT による読み取り 38	定義 304
説明 416	stdout	SYSLIN データ・セット 305
SQL ステートメント	行送りの制御 40	説明 301
概要 515	DISPLAY による送信 40	SYSMDECK ファイル
国別 10 進数データの使用 519	DISPLAY の設定 544	説明 302
コーディング	STEPLIB 環境変数	定義 307
概要 517	コンパイラーの指定の例 324	SYSMDUMP ファイル
制約事項 517	説明 544	説明 301
制約事項 517	STGOPT コンパイラー・オプション 421	SYSOPTF データ・セット
バイナリー・データの使用 520	STOP RUN ステートメント	説明 300
文字データの使用 518	サブプログラムにおける 553	定義 303
戻りコード 520	マルチスレッド化 553	SYSPRINT データ・セット
CCSID の決定 522	メインプログラムにおける 553	使用されない場合 847
Db2 サービスのための使用 515		説明 301
EXIT コンパイラー・オプション 862		定義 304
SQL DECLARE 518		

SYSPUNCH データ・セット  
説明 301, 305  
DECK コンパイラー・オプションの要件 367  
SYSTEM データ・セット  
説明 301  
定義 305  
メッセージの送信 423  
SYSUDUMP ファイル  
説明 301  
SYSUT データ・セット 301

## T

TALLYING 句 (INSPECT)、例 128  
TERMINAL コンパイラー・オプション 423  
TEST AFTER 114  
TEST BEFORE 114  
TEST コンパイラー・オプション  
説明 423  
デバッグに使用 463  
パフォーマンスの考慮事項 791  
マルチオプションの相互作用 349  
text-name 環境変数 322  
THREAD コンパイラー・オプション  
オブジェクト指向 COBOL のための 333, 339  
説明 428  
ネストされたプログラムと一緒にには使用できない 564  
パフォーマンスの考慮事項 791  
マルチオプションの相互作用 349  
Java とのインターオペラビリティのための 333, 339  
LINKAGE SECTION 18  
TITLE ステートメント 448  
リストのヘッダーの制御 5  
TRACK OVERFLOW オプション 191  
TRAP ランタイム・オプション  
行順次ファイルのクローズ 246  
ON SIZE ERROR 273  
QSAM ファイルのクローズ 195  
VSAM ファイルのクローズ 227  
TRUNC コンパイラー・オプション  
説明 430  
パフォーマンスの考慮事項 791  
分離型の CICS 変換プログラムのサブオプション 510  
TSO  
コンパイラー・メッセージ用の  
SYSTEM 305  
ALLOCATE コマンド 296  
CALL コマンド 296  
UNIX のもとでのコンパイル  
概要 296  
TSO (続き)  
UNIX のもとでのコンパイル (続き)  
CLIST の例 297  
TSO のもとでコンパイルするための  
CLIST 297

## U

U フォーマット・レコード  
要求 187  
レイアウト 188  
U レベルのエラー・メッセージ 318, 460  
Unicode  
エンコードおよびストレージ 155  
説明 146  
データ処理 141  
Db2 での使用 518  
JNI サービス 753  
UNIX  
API の呼び出し 545  
UNSTRING ステートメント  
オーバーフロー条件 272  
使用 120  
例 121  
DBCS データを使用する 817  
UPPER-CASE 組み込み関数 130  
USAGE 節  
グループ・レベルの 27  
グループ・レベルの NATIONAL 句 152  
非互換データ 61  
INDEX 句による指標データ項目の作成 80  
OBJECT REFERENCE 714  
USE FOR DEBUGGING 宣言  
概要 455  
USE ステートメント 448  
USING 句  
INVOKE ステートメント 718  
PROCEDURE DIVISION ヘッダー 578, 706  
UTF-16  
エンコード方式、国別データの 147  
定義 147  
UTF-8  
エンコードおよびストレージ 155  
切り捨てる移動を避ける 653  
国別との間の変換 159  
組み込み関数の使用 160  
データ項目の処理 159  
定義 147  
ASCII インバリエント文字のエンコード方式 147  
INSPECT の使用を避ける 653  
JNI サービス 756  
XML 文書で参照変更を回避 160

UTF-8 (続き)  
XML 文書の構文解析 653  
XML 文書の生成例 677  
UTF-8 データ  
Unicode 組み込み関数の使用 162

## V

V フォーマット・レコード  
要求 182  
レイアウト 184  
VALUE IS NULL 580  
VALUE OF 節 15  
VALUE 節  
大きな、TRUNC(BIN) での 431  
外部浮動小数点に使用できない 55  
可変長グループへの割り当て 90  
国別グループでの英数字リテラル、例 85  
国別データを持つ英数字リテラルの例 137  
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 85  
グループ・レベルの 84  
それぞれの項目に個別に 84  
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 56  
VBREF コンパイラー・オプション  
出力例 496  
使用 464  
説明 433  
VLR コンパイラー・オプション  
説明 433  
VSAM ファイル  
エラー処理 273  
オープン  
概要 220  
空の 221  
拡張アドレッシング機能 239  
環境変数を使用した割り振り 233  
クローズ 227  
状況コード  
説明 279  
例 280  
代替索引の作成 231  
入出力エラー処理 227  
入出力ステートメントのコーディング 218  
パスワードによる保護 228  
パフォーマンスの考慮事項 237  
ファイル位置標識 (CRP) 220, 224  
ファイル状況キー 227  
ファイルの処理 209  
ファイル編成の比較 211

## VSAM ファイル (続き)

- レコードの更新 225
- レコードの削除 226
- レコードの置換 226
- レコードの追加 225
- レコードの読み取り 224
- レコード・レベル共用 (RLS)

- エラー処理 236
- 概要 235
- 更新問題の帽子 235
- 制約事項 236

## ロード

- アクセス方式サービス・プログラムによる 222
- 拡張フォーマット 222
- 順次 221
- 動的またはランダム 222

DATA DIVISION 記入項目 216

ENVIRONMENT DIVISION 記入項目 212

## z/OS のもとでの

- データ・セットの定義 230
- ファイルの可用性 229
- JCL 233
- RLS モード 235

## VSAM 用語

- 非 VSAM 用語との比較 209
- BDAM データ・セット 209
- BDAM に対応する RRDS 209
- ISAM に対応する KSDS 209
- QSAM に対応する ESDS 209

VSAMOPENFS コンパイラー・オプション 434

## W

W レベルのメッセージ 318, 460

## WHEN 句

- EVALUATE ステートメント 105
- SEARCH ALL ステートメント 97
- SEARCH ステートメント 95

WHEN-COMPILED 組み込み関数 139

WHEN-COMPILED 特殊レジスター 139

## WITH DEBUGGING MODE 節

- デバッグ行 455
- デバッグ・ステートメント 455

## WITH POINTER 句

- STRING 117
- UNSTRING 120

## WORD コンパイラー・オプション

- 説明 435
- 分離型の CICS 変換プログラムの場合に推奨 510
- マルチオプションの相互作用 349
- CICS 統合変換プログラムの場合に推奨 507

## WORKING-STORAGE SECTION

- インスタンス・データ 701, 726
- インスタンス・メソッド 704
- クライアント 714, 715
- データの保管場所 366
- ファクトリー・データ 730
- マルチスレッド化の考慮事項 715

## LOCAL-STORAGE との比較

- 概要 15
- 例 16
- OO クライアント 715

WRITE ADVANCING ステートメント 195

## WRITE ステートメント

- 行順次ファイル 243
- マルチスレッド化シリアルライゼーション 607

QSAM 191

VSAM 218

## X

x サフィックス、cob2 での 329, 330

## XML GENERATE ステートメント

- COUNT IN 682
- NAME 679
- NAMESPACE 678
- NAMESPACE-PREFIX 678
- NOT ON EXCEPTION 680
- ON EXCEPTION 682
- SUPPRESS 679
- TYPE 680
- WITH ATTRIBUTES 677
- WITH ENCODING 681
- XML-DECLARATION 677

## XML PARSE ステートメント

- 概要 628
- 使用 630
- NOT ON EXCEPTION 631
- ON EXCEPTION 631

## XML イベント

- エンコードの競合 657, 658

概要 634

- 処理 628, 632
- 処理プロシーチャー 630
- 致命的エラー 657

## CONTENT-CHARACTERS

- セグメントの構文解析時 645
- 例 670

## END-OF-INPUT

- セグメントの構文解析時 645
- 例 670

EXCEPTION 656

NAMESPACE-DECLARATION 639

## XML 構文解析

- エンコード競合の処理 657, 658

## XML 構文解析 (続き)

概要 627

検証を伴う

- 概要 641

- 制約事項 642

- パフォーマンスの考慮事項 642

- 例 671

終了 659

処理プロシーチャーでの制御フロー 635

説明 630

致命的エラー 657

特殊レジスター 632, 634

戻りコード 654, 823

理由コード 654, 823

例外の処理 654

1 セグメントずつ

- 概要 644

- 例 669

XML 構文解析の終了 659

## XML 出力

- エンコードの制御 681

拡張

- 基本的原理と技法 687
- データ定義の変更例 688

生成

- 概要 675

- 例 683

## XML 出力の拡張

- 基本的原理と技法 687
- データ定義の変更例 688

## XML 出力の生成 675

- 概要 676

- 例 683

## XML 処理プロシーチャー

- エンコード競合の処理 658

- 書き込み 632

- 構文解析例外の処理 654

- 指定 630

- 特殊レジスターの使用 632, 634

- パーサーでの制御フロー 635

- 複数セグメント 645

- 例

- 検証を伴う構文解析 671

- 1 セグメントずつ 669

- XML の処理用プログラム 661

EXIT PROGRAM または GOBACK  
でのエラー 634

XML PARSE の制約事項 633

XML-CODE の設定 658

XML スキーマ 643

## XML 生成

- エラーの処理 682

- エレメントの生成 676

- 概要 675



1043

## Z

ZONECHECK コンパイラー・オプション  
438

ZONEDATA コンパイラー・オプション  
440

ZWB コンパイラー・オプション 443

z/OS

下でのプログラムの実行 588

メイン・パラメーターへのアクセス

概要 588

例 589

UNIX のもとでのコンパイル 287

z/OS UNIX

オブジェクト指向アプリケーションの  
コンパイル

概要 333

例 336

オブジェクト指向アプリケーションの  
準備

概要 334

例 336

環境変数の設定

概要 543

例 544

環境変数へのアクセス

概要 543

例 544

コピーブック 446

コピーブック探索順序 322, 328, 447

コンパイラー環境変数 321

コンパイラー・オプションの指定 323

実行環境 542

実行する、オブジェクト指向アプリ  
ケーションを

概要 336

XPLINK リンケージ 343

制約事項 541

ソートおよびマージの制限 247

添え字からのコンパイル 330

プログラムの開発 541

プログラムの実行 542

プログラムは再入可能でなければなら  
ない 571

メイン・パラメーターへのアクセス

概要 547

例 547

リンクする、オブジェクト指向アプリ  
ケーションを

概要 334

例 336

UNIX のもとでのコンパイル 321

z/OS UNIX シェルでのリンク

情報を cob2 へ引き渡す 327

c89 コマンド 324

z/OS UNIX シェルでのリンク (続き)

cob2 コマンドの使用

概要 324

例 326

DLL 325

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 176

コンパイラー・データ・セット 290

ACCEPT によるファイルの読み取り  
38

DISPLAY によるファイルの書き込み  
40

DLL の探索順序 596

QSAM によるファイルの処理 205

## [特殊文字]

! 文字、16 進値 652

# 文字、16 進値 652

\*CBL ステートメント 445

\*CONTROL ステートメント 445

-b cob2 オプション

情報をリンカーに引き渡すための 327

DLL 作成用 325

-c cob2 オプション、リンクではなくコン  
パイルのための 327

-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 327

-e cob2 オプション、入り口点を指定する  
ための 328

-g cob2 オプション、TEST を指定するの  
と同等 328

-I cob2 オプション、コピーブックを検索  
するための 328

-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 328

-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 328

-o cob2 オプション、出力ファイルを指定  
するための 328

-q cob2 オプション、コンパイラー・オブ  
ションを指定するための 328

-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 329

# 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 329

-? 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 329

.a サフィックス、cob2 での 329

.adt サフィックス、cob2 での 330

.adt ファイル 350

.cbl サフィックス、cob2 での 329

.dbg サフィックス、cob2 での 330

.dbrm サフィックス、cob2 での 330

.dek サフィックス、cob2 での 330

.lst サフィックス、cob2 での 330

.o サフィックス、cob2 での 329, 330

.x サフィックス、cob2 での 329, 330

[ 文字、16 進値 652

| 文字、16 進値 652

] 文字、16 進値 652

\_BPX\_SHAREAS 環境変数 546

\_CEE\_ENVFILE 環境変数

説明 544

Java 設定値を示す 341

\_CEE\_RUNOPTS 環境変数

説明 544

ランタイム・オプションの指定 542

XPLINK の設定 343

\_IGZ\_SYSOOUT 環境変数

設定 544

STDOUT または STDERR への書き込  
み 40





プログラム番号: 5655-EC6

Printed in Japan