CICS Transaction Server for z/OS
バージョン 4 リリース 1

アプリケーション・プログラミング・ガイド

SC88-5850-00
(英文原典：SC34-7022-00)

CICS Transaction Server for z/OS
バージョン 4 リリース 1

アプリケーション・プログラミング・ガイド

SC88-5850-00
(英文原典：SC34-7022-00)

お願い

本書および本書で紹介する製品をご使用になる前に、 893 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、CICS Transaction Server for z/OS バージョン 4 リリース 1 (製品番号 5697-E93)、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。

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： SC34-7022-00
CICS Transaction Server for z/OS
Version 4 Release 1
Application Programming Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2009.7

© Copyright International Business Machines Corporation 1989, 2009.

目次

前書き xiii
本書について xiii
本書の対象読者 xiii

CICS Transaction Server for z/OS バージョン 4 リリース 1 の変更点 xv

第 1 部 CICS アプリケーションの作成 1

第 1 章 概要: CICS アプリケーションの作成 3
CICS アプリケーションとは 3
CICS プログラム、トランザクション、およびタスク 3
CICS プログラミング 4
CICS プログラミング・コマンド. 5
EXEC インターフェース・ブロック (EIB) 6
変換 6
CICS のテスト. 7
CICS プログラミング・ロードマップ 7

第 2 部 プログラミング言語と言語環境 9

第 2 章 言語環境プログラム 11
言語環境プログラムの呼び出し可能サービス . . . 12
言語環境プログラムの異常終了および条件処理 . . 14
言語環境プログラムのストレージ 15
言語環境プログラムにおける言語の混合 16
ダイナミック・リンク・ライブラリー (DLL) . . . 18
言語環境プログラムのランタイム・オプションの定義 19
CEEBXITA および CEECSTX ユーザー出口 . . 21
CICSVAR、CICS 環境変数 22
言語環境プログラムの CEEBINT 出口 23

第 3 章 COBOL でのプログラミング . . 25
COBOL プログラミングの制約事項および要件. . . 26
言語環境プログラム CBLPSHPOP オプション . . 29
DL/I CALL インターフェースの使用 30
VS COBOL II プログラム 31
COBOL での基底付きアドレッシングの使用 . . . 32
COBOL プログラムからのサブプログラムの呼び出し 33
プログラムとサブプログラム間の制御のフロー . 34
サブプログラムの呼び出し規則 36
COBOL2 および COBOL3 変換プログラム・オプション 39
COBOL プログラムの CICS 変換プログラム・アクション 41
COBOL プログラムのバッチ・コンパイル 43
ネストされた COBOL プログラム. 45
OS/VS COBOL プログラムのアップグレード . . . 50

第 4 章 C および C++ でのプログラミング 53
C および C++ プログラミングの制約事項および要件 54
C および C++ での引数の渡し 58
C および C++ から EIB へのアクセス 60
C および C++ の地域サポート 60
XPLink と C および C++ プログラミング 61
XPLink による X8 および X9 モード TCB の使用 62
XPLink オブジェクトと非 XPLink オブジェクトの間での制御の引き渡し 62
グローバル・ユーザー出口と XPLink 62

第 5 章 PL/I でのプログラミング. . . . 65
PL/I プログラミングの制約事項と要件 65
PL/I アプリケーションにおける言語環境プログラムのコーディング要件 67
取り出した PL/I ルーチン 68

第 6 章 アセンブラー言語でのプログラミング 71
アセンブラー言語プログラミングの制約事項および要件. 71
アセンブラー言語アプリケーションのための言語環境プログラムのコーディング要件 73
アセンブラー言語プログラムの呼び出し 75

第 3 部 アプリケーション・プログラムの変換、コンパイル、インストール、およびテスト. 79

第 7 章 変換およびコンパイル 81
統合 CICS 変換プログラム 81
統合 CICS 変換プログラムの使用 82
CICS 変換プログラムのオプションの指定 . . . 83
変換のプロセス 84
CICS 提供の変換プログラム. 87
個別の変換プログラムの動的な起動 87
CICS 変換プログラムの使用. 88
変換プログラムのオプションの定義 90
変換プログラム・オプション 90
変換プログラムのオプション・テーブル . . . 104
COPY ステートメントの使用 105

© Copyright IBM Corp. 1989, 2009

CICS 提供のインターフェース・モジュール . . . 106
EXEC インターフェース・モジュール 106
CPI コミュニケーション・インターフェース・モジュール 106
SAA リソース・リカバリー・インターフェース・モジュール. 106
EXEC インターフェース・モジュールの使用 . . . 107
COBOL 108
PL/I 108
C および C++ 109
アセンブラー言語 109
LEASM を使用したアセンブラー言語プログラムの例 110

第 8 章 アプリケーション・プログラムのインストール 119
プログラムのインストール・ステップ 120
動的プログラム LIBRARY リソースの使用 . . . 120
| 動的 LIBRARY リソースの使用例 122
MVS 常駐モードおよびアドレッシング・モードの定義 131
プログラムのアドレッシング・モードの設定 . . 131
CICS アドレス・スペースに関する考慮事項 . . 132
永続的に常駐するプログラムの作成 133
リンク・パック域にあるアプリケーションの実行 133
RDSA にあるアプリケーション・プログラムの実行 134
アセンブラー 134
C および C++ 135
COBOL 136
PL/I 137
アプリケーション・プログラムにおける BMS マップ・セットの使用 137
アプリケーション・プログラムをインストールするための CICS 提供プロシージャーの使用 138
ロード・ライブラリーの 2 次エクステントへのプログラムのインストール 140
CICS 提供インターフェース・モジュールの組み込み 140
アセンブラー言語アプリケーション・プログラムのインストール 141
COBOL アプリケーション・プログラムのインストール 143
COBOL アプリケーション・プログラムをインストールする JCL の例. 144
PL/I アプリケーション・プログラムのインストール 147
PL/I アプリケーション・プログラムをインストールする JCL の例 147
PL/I プロシージャーと統合変換プログラム . . 149
C アプリケーション・プログラムのインストール 149
C アプリケーション・プログラムをインストールする JCL の例 151
ユーザー独自のジョブ・ストリームの使用. . . . 153
変換プログラムの要件 154
EXEC CICS または EXEC DLI コマンドを使用するオンライン・プログラム 154
CALL DLI インターフェースを使用するオンライン・プログラム 156
EXEC DLI コマンドを使用するバッチまたは BMP プログラム 156
DL/I CALL コマンドを使用するバッチまたは BMP プログラム 156

第 9 章 マップ・セットおよび区分セットのインストール 157
マップ・セットのインストール 159
マップ・セットのタイプ. 159
物理マップ・セットのインストール 161
シンボリック記述マップ・セットのインストール 163
物理マップおよびシンボリック記述マップの同時インストール 165
区分セットのインストール 168
CICS へのプログラム、マップ・セット、および区分セットの定義. 169

第 10 章 アプリケーションのテスト 171
テストに対するアプリケーションの準備 172
テストに対するシステムの準備 173

第 11 章 実行診断機能 (EDF). 175
EDF を使用する場合の制約事項 176
OPEN TCB および EDF. 178
パラメーター・リストのスタッキング 178
セキュリティーに関する考慮事項. 179
EDF 表示画面での実行内容. 179
ヘッダー 180
本体 180
EDF の使用 186
プログラム実行の中断 187
単一画面モードにおける EDF の使用 188
二重画面モードでの EDF の使用. 190
EDF とリモート・トランザクション. 191
EDF および端末を使用しないトランザクション 191
EDF および DTP プログラム 191
EDF の停止 192
重ね書きによる変更 193
EDF の応答 194
EDF メニュー機能の使用 195

第 12 章 一時記憶域のブラウズ (CEBR) 203
CEBR トランザクションの使用 203
CEBR トランザクションの表示内容 205
ヘッダー 205
コマンド域 206
本体 206
メッセージ行 206
CEBR ファンクション・キーの使用 206
CEBR コマンドの使用 207
一時データでの CEBR トランザクションの使用 210

第 13 章 コマンド・レベル・インタープリター (CECI) 211
CECI による表示内容 211
コマンド行 211
状況表示行 212
本体 215
メッセージ行 216
ファンクション・キーにおける CECI オプション 216
CECI の使用 217
変更方法 218
CECI ファンクション・キーの使用 219
拡張域. 219
変数 219
EXEC インターフェース・ブロック (EIB). . . 221
エラー・メッセージの表示画面 221
コマンドの保管. 222
CECI の実行方法 223
CECI セッション 223
異常終了 224
例外条件 224
プログラム制御コマンド. 224
端末の共用 224
共用ストレージ: LENGTH オプションを指定しない ENQ コマンド 225

第 14 章 CICS アプリケーションでのデバッガーの使用準備 227
デバッグ・プロファイル. 228
デバッグ・プロファイルを使用してデバッグ対象のプログラムを選択する 230
デバッグ・プロファイルでの汎用パラメーターの使用 232

第 15 章 ワークステーションから CICS アプリケーションをデバッグする . . . 235
ワークステーションからアプリケーションのデバッグを準備する 235

第 16 章 CICS アプリケーションでのデバッグ・ツールの使用 237
デバッグ・ツールについて 237
デバッグ・ツールによるデバッグ・アプリケーションの準備 238

第 4 部 CICS アプリケーションのプログラミング手法 239

第 17 章 アプリケーション設計 241
疑似会話型設計と会話型設計 241
端末の割り込み機能 243
タスクの開始方法 244
どのトランザクションか? 245
ビジネス・ロジックと表示ロジックの分離. . . . 248
マルチスレッド化: 再入可能なプログラム、準再入可能なプログラム、およびスレッド・セーフ・プログラム. 249
準再入可能なアプリケーション・プログラム . . 250
スレッド・セーフ・プログラム 251
OPENAPI プログラム. 258
FORCEQR システム初期設定パラメーターの使用 259
再入不能プログラム 260
トランザクション内のデータの格納 260
トランザクション作業域 (TWA) 261
ユーザー・ストレージ 261
LINK コマンドおよび XCTL コマンドにおける COMMAREA 262
LINK および XCTL コマンドのチャネル . . . 263
プログラム・ストレージ. 263
一時記憶域 263
区画内一時データ 265
GETMAIN SHARED コマンド. 266
ユーザー独自のデータ・セット 266
CICS コマンドに渡される区域の長さ 266
LENGTH オプション. 267
ジャーナル・レコード 267
データ・セット定義 267
推奨 267
エラーの最小化. 268
アプリケーション・エラーからの CICS の保護 268
アプリケーションのテスト 268
端末を持たないトランザクションのセキュリティー 269

第 18 章 パフォーマンスの設計 271
プログラム・サイズ 271
仮想記憶域 272
ページングの影響の軽減. 273
リソースの排他制御 276
操作のコントロール 277
オペレーティング・システム待機. 278
NOSUSPEND オプション 278
効果的な順次データ・セットのアクセス 279
効率的なロギング 280

第 19 章 トランザクション間のデータの共用 281
共通作業域 (CWA) の使用 281
CWA の保護 282
TCTTE ユーザー域 (TCTUA) の使用 285
RETURN コマンドでの COMMAREA の使用. . . 286
RETURN コマンドにおけるチャネルの使用 . . . 286
データを共用する表示画面の使用. 287

第 20 章 拡張プログラム間データ転送: 新しい COMMAREA としてのチャネル . 289
チャネル: クイック・スタート 289
コンテナーおよびチャネル 289
基本的な例 290
チャネルの使用: いくつかの典型的なシナリオ . . 292

1 つのチャネルに 1 つのプログラム 293
1 つのチャネルに複数のプログラム (1 つのコンポーネント) 293
複数のチャネルに 1 つのコンポーネント . . . 294
複数の対話式コンポーネント 294
チャネルの作成. 295
現行チャネル 296
LINK コマンドを使用した現行チャネルの例 . . 296
XCTL コマンドを使用した現行チャネルの例 299
現行チャネル: START および RETURN コマンド 300
チャネルの有効範囲 301
LINK コマンドを使用した有効範囲の例 . . . 301
LINK コマンドおよび XCTL コマンドを使用した有効範囲の例. 303
プログラムに渡されたコンテナーの検出 305
リンクから返されたコンテナーの検出 305
CICS 読み取り専用コンテナー. 306
チャネルの設計: 最良事例 306
チャネルの構成および使用: 例 308
チャネルと BTS アクティビティー 309
コンテキスト 310
JCICS からのチャネルの使用 311
チャネルを使用した動的ルーティング 312
データ変換 313
データ変換が必要な理由. 313
チャネルを使用したコード・ページ変換の準備 313
チャネルを使用したデータ変換 315
チャネルの利点. 319
COMMAREA からチャネルへのマイグレーション 321
COMMAREA を渡す LINK コマンドのマイグレーション 321
COMMAREA を渡す XCTL コマンドのマイグレーション 322
RETURN コマンドでの疑似会話型 COMMAREA のマイグレーション 322
START データのマイグレーション 323
データを渡すために一時ストレージを使用するプログラムのマイグレーション 323
動的にルーティングされたアプリケーションのマイグレーション. 323

第 21 章 プログラム制御. 325

プログラムのリンク 326
アプリケーション・プログラムの論理レベル . . 326
戻り操作が見込まれる別のプログラムへのリンク 326
他のプログラムへのデータの受け渡し 327
COMMAREA 327
チャネル 330
INPUTMSG 330
混合アドレッシング・モードの使用 332
LINK を使用したデータの受け渡し 333
RETURN を使用したデータの受け渡し. 335

第 22 章 類縁性 339

類縁性のタイプ. 340
トランザクション間の類縁性 340
トランザクションとシステム間の類縁性 . . . 340
プログラミング手法と類縁性 341
安全な手法 341
安全ではない手法 342
疑いのある手法. 342
推奨 343
類縁性を防止するプログラミング手法 343
COMMAREA 343
TCTUA 345
ENQMODEL リソース定義による ENQ および DEQ コマンドの使用. 346
BTS コンテナー 348
類縁性を生じさせるプログラミング手法 349
共通作業域の使用 349
GETMAIN SHARED ストレージの使用. . . . 350
LOAD PROGRAM HOLD コマンドの使用. . . 351
タスク存続期間ストレージの共用. 353
WAIT EVENT コマンドの使用 354
ENQMODEL リソース定義のない ENQ および DEQ コマンドの使用. 355
類縁性を生じさせる可能性のあるプログラミング手法 356
一時記憶域の使用 357
一時データの使用 360
RETRIEVE WAIT および START コマンドの使用 361
START および CANCEL REQID コマンドの使用 363
DELAY および CANCEL REQID コマンドの使用 365
POST および CANCEL REQID コマンドの使用 367
トランザクション間の類縁性の検出 368
アプリケーション生成プログラムによって発生するトランザクション間の類縁性 369
トランザクション間の類縁性の期間と有効範囲 . . 369
類縁性トランザクション・グループ 370
関係および存続期間 370

第 23 章 リカバリーの設計. 377

ジャーナル処理. 377
ジャーナル・レコード 377
ジャーナル出力の同期 378
同期点処理 380

第 24 章 例外条件の取り扱い. 383

デフォルト CICS 例外処理. 383
インライン・コードによる例外条件の処理. . . . 384
RESP オプションおよび RESP2 オプションの使用方法. 385
C における例外処理の例 385
COBOL での例外処理の例 387
デフォルト CICS 例外処理の変更 387
HANDLE CONDITION コマンドの使用. 389
RESP および NOHANDLE オプション. . . . 390
CICS が実行する内容を見失わない方法. . . . 391

HANDLE CONDITION ERROR コマンドの使用 392
IGNORE CONDITION コマンドの使用 392
HANDLE ABEND コマンドの使用 394
PUSH HANDLE および POP HANDLE コマンドの使用 394

第 25 章 異常終了のリカバリー 397
プログラム・レベルの異常終了出口の作成. . . . 398
操作の再試行 400
トレース 401
トレースの入り口点 402
アプリケーション・パフォーマンスのモニター . . 403
ダンプ. 403

第 26 章 QUERY SECURITY コマンド 405
QUERY SECURITY コマンドの使用. 405
レコード・レベルまたはフィールド・レベルでのセキュリティー保護 405
CICS 定義のリソース ID 406
SEC システム初期設定パラメーター. 406
プログラミングのヒント. 406

第 27 章 CICS の相互通信 407
設計上の考慮事項 408
プログラミング言語 408
トランザクション・ルーティング. 408
機能シップ 409
分散プログラム・リンク (DPL) 409
分散プログラム・リンク機能の使用 410
分散プログラム・リンクの例 412
分散プログラム・リンクのプログラミングに関する考慮事項 417
非同期処理 423
分散トランザクション処理 (DTP). 423
共通プログラミング・インターフェース・コミュニケーション (CPI コミュニケーション) 423
外部 CICS インターフェース (EXCI) 424

第 5 部 データ・マッピング 427

第 28 章 アプリケーション・データおよび XML のマッピングと変換 429
CICS XML 支援機能 430
DFHLS2SC: 高水準言語から XML スキーマへの変換 431
DFHSC2LS: XML スキーマから高水準言語への変換 437
CICS 支援機能用のマッピング・レベル. . . . 445
高水準言語および XML スキーマのマッピング 450
エレメントの可変配列 479
XML 属性のサポート. 484
<xsd:any> と xsd:anyType のサポート 486
<xsd:choice>のサポート. 489
代替グループのサポート. 491
抽象エレメントと抽象データ型のサポート. . . 492
COBOL での可変反復内容の処理方法 494
可変長の値と空白のサポート 497
言語構造からのマッピングの生成. 500
XML スキーマからのマッピングの生成. 502
アプリケーション・データの XML への変換. . . 504
アプリケーション・データへの XML の変換. . . 506
アプリケーションからの XML の照会 507
データ型による XML の取り扱い 508
<xsd:any> データ型の処理 509
XML 変換の妥当性検査 510

第 6 部 ビジネス・サービスおよびバンドル 513

第 29 章 CICS アプリケーションからのビジネス・サービスの作成 515
Service Component Architecture (SCA) 515
SCA コンポジットおよびワイヤリング. 517
コンポジットの作成およびデプロイのベスト・プラクティス 519
チャネル・ベース・サービスの作成 520
XML ベース・サービスの作成. 522
CICS によるサービスの処理 524
SCA の問題のトラブルシューティング 525

第 30 章 ユーザー・リソース・タイプをバンドルで作成する 527
バンドル登録インターフェース 527
登録プログラムの書き込み 528
登録インターフェースで使用されるコンテナー 529
コールバック・プログラムの書き込み 531
コールバック・インターフェースで使用されるコンテナー. 533
DFHRL-CONTROL コンテナー 533
DFHRL-DATA コンテナー 534
DFHRL-NAME コンテナー 534
DFHRL-ROOT コンテナー 535
DFHRL-TYPE コンテナー 535
バンドル・マニフェスト・スキーマ 535
SCA 複合スキーマ 537

第 7 部 ファイル制御 539

第 31 章 ファイル制御についての理解 541
VSAM データ・セット: KSDS、ESDS、RRDS . . 541
空のデータ・セット 543
VSAM 代替索引 543
RLS モードでのファイルのアクセス. 544
VSAM レコードの識別 545
キー 545
相対レコード番号 (RRN)、相対バイト・アドレス (RBA) または拡張相対バイト・アドレス (XRBA) 546
ESDS の拡張アドレッシングへのアップグレード 547

リカバリー可能ファイルにおける VSAM レコードのロック 549
更新ロックおよび削除ロック (非 RLS モードのみ) 549
RLS レコード・レベルのロック 550
排他ロックおよび共用ロック 551
ロック期間 551
ロックのアクティブ状態および保存状態 . . . 552
BDAM データ・セット 553
BDAM レコードの識別 555
CICS 共用データ・テーブル 557
カップリング・ファシリティー・データ・テーブル 558
データの共用手法 561
トランザクションのデッドロック. 563
VSAM が検出するデッドロック (RLS のみ) . . 565
デッドロックを回避するための規則 565

第 32 章 ファイル制御操作. 567
CICS コマンドを使用したレコードの読み取り . . 567
直接読み取り (READ コマンドの使用) 567
順次読み取り (ブラウズ) 571
BDAM データ・セットからのレコードのブラウズ 575
スキップ順次処理 576
CICS コマンドを使用したレコードの更新 576
TOKEN オプション 577
条件付き VSAM ファイル更新要求 578
BDAM データ・セットからのレコードの更新 579
CICS コマンドを使用したレコードの削除 579
ブラウズ中のレコードの更新および削除 (VSAM RLS のみ) 580
UPDATE のロック 580
CICS コマンドを使用したレコードの追加 581
ESDS に書き込む場合の CICS ロック 583
BDAM データ・セットへのレコードの追加 . . 583
効率的なデータ・セットの操作 584
VSAM データ・セット 584
BDAM データ・セット 586
効果的なブラウズ (非 RLS モード) 586

第 8 部 端末管理 587

第 33 章 端末アクセス方式のサポート 589

第 34 章 端末制御コマンド. 591
送信/受信モード 591
端末の競合 592
RETURN IMMEDIATE 592
順序のない会話. 593
割り込み 593
端末待ち 594

第 35 章 データ伝送コマンドの使用 595
RECEIVE における受信内容 595
入力のチェーニング 595
論理メッセージ. 596
NOTRUNCATE オプション. 596
印刷キー 596

第 36 章 装置制御コマンド. 597

第 37 章 端末装置のサポート. 599

第 38 章 ご使用の端末情報の検出 . . . 603
端末管理の操作における EIB フィードバック . . 604

第 39 章 VTAM の使用 607
入力データのチェーニング 607
出力データのチェーニング 607
論理レコードの取り扱い. 608
応答プロトコル. 608
機能管理ヘッダーの使用. 609
インバウンド FMH 609
アウトバウンド FMH. 609
割り込みの防止 (ブラケット・プロトコル) . . . 610

第 40 章 順次端末サポートの使用 . . . 611
順次端末に対するコーディングの考慮事項. . . . 612
印刷形式設定 612
GOODNIGHT 規則 612

第 41 章 TCAM の使用 615
TCAM/DCB インターフェースのコーディング . . 615

第 42 章 バッチ・データ交換の使用 617

第 43 章 端末管理: パフォーマンスの設計 621

第 44 章 3270 ファミリーの端末 . . . 623
3270 バッファー 623
出力データ・ストリーム. 624
3270 書き込みコマンド 624
3270 表示フィールド. 625
表示特性 626
3270 フィールド属性. 626
保護 627
変更 627
輝度 628
基本カラー 628
拡張属性 628
データ・ストリーム中のオーダー. 629
フィールド開始オーダー. 630
フィールド変更オーダー. 631
バッファー・アドレス設定オーダー 631
属性設定オーダー 632
アウトバウンド・データ・ストリームの例. . . . 633
3270 端末からの入力 635
データ・キー 635
キーボード制御キー 636
アテンション・キー 636
3270 端末からの読み取り 637

インバウンド・フィールド形式 638
入力データ・ストリームの例 638
不定形式モード. 639

第 9 部 インターバル制御およびタスク制御 641

第 45 章 インターバル制御. 643
満了時刻 644
要求 ID 645

第 46 章 タスク制御 647
リソースへのアクセス・シーケンスの制御. . . . 648

第 10 部 ストレージ保護およびトランザクション分離 651

第 47 章 CICS のストレージ保護およびトランザクション分離 653
ストレージ制御. 653
ストレージ保護. 654
ストレージのカテゴリー. 655
トランザクション分離 656
システム障害の縮小 656
アプリケーション・データの保護. 656
無効なアドレスを渡されることからの CICS の保護 657
アプリケーション開発の補助 657
アプリケーション用のストレージ・キーの定義 . . 657
システム全体のストレージ域 658
タスク存続期間ストレージ 658
出口および PLT プログラム専用のプログラム作業用ストレージ. 658
COMMAREA によるデータの引き渡し 659
GETMAIN コマンド 659
実行およびストレージ・キーの選択 661
ユーザー・キーのアプリケーション 662
CICS キーのアプリケーション. 663
トランザクション分離の使用 666
MVS サブスペース 668
トランザクションのサブスペースおよび基本スペース 668
共通サブスペースおよび共用ストレージ . . . 669

第 11 部 一時データと一時記憶域 671

第 48 章 一時データ管理. 673
区画内一時データ・キュー 673
区画外キュー 674
間接キュー 675
自動トランザクション開始 (ATI). 675

第 49 章 一時記憶域制御. 677
一時記憶域キュー 677
一時記憶管理の標準的な使用法 678

第 12 部 CICS 文書 681

第 50 章 文書および文書テンプレートの概要 683
記号および記号リスト 684
文書テンプレートのキャッシングおよびリフレッシュ 687
文書のコード・ページ変換 689

第 51 章 文書テンプレートの設定 . . . 691
区分データ・セット内のテンプレート 691
z/OS UNIX システム・サービス・ファイル内のテンプレート 692
CICS ファイル、一時記憶、または一時データ内のテンプレート 692
CICS プログラムのテンプレート 693
DFHDHTL - プログラム・テンプレート prolog マクロおよび epilog マクロ 694
出口プログラムのテンプレート 695
出口プログラムでのテンプレートの連絡域. . . 696
文書テンプレートの記号の使用 698
テンプレート・コマンドの埋め込み 699

第 52 章 文書および文書テンプレートを使用したプログラミング 701
文書の作成 701
記号値の定義 702
記号および記号リストの指定に関する規則. . . . 704
文書へのデータの追加 707
文書でのデータの置換 709
文書の検索、保管、および再利用. 710
文書の削除 712

第 13 部 名前付きカウンター・サーバー 715

第 53 章 概要: 名前付きカウンター・サーバー 717
名前付きカウンター・フィールド. 718
名前付きカウンター・プール 718
名前付きカウンター・オプション・テーブル . . 719

第 54 章 名前付きカウンター EXEC インターフェースの使用 721

第 55 章 名前付きカウンター呼び出しインターフェースの使用 723
アプリケーション・プログラミングに関する考慮事項 723
構文規則 725
結果オーバーフローの検査 732

ヌル・パラメーターを持つ DFHNCTR 呼び出しの例 732
戻りコード 733

第 56 章 名前付きカウンターのリカバリー 737

第 14 部 印刷とスプール・ファイル 739

第 57 章 印刷の CICS サポート . . . 741
CICS プリンターの書式設定 741
印刷出力の要求. 742
CICS 3270 プリンター 743
CICS 3270 プリンターのオプション. 744
PRINT オプションおよび印刷制御ビット . . . 744
ERASE オプション 745
線幅オプション: L40、L64、L80、および HONEOM. 745
NLEOM オプション 746
FORMFEED 747
PRINTERCOMP オプション 747
非 3270 CICS プリンター 748
SCS 入力. 749

第 58 章 CICS におけるプリンターの使用 751
CICS プリンターの特性の判別. 751
BMS ページ・サイズ、3270 プリンター . . . 751
複数のプリンター・タイプのサポート 752
CICS プリンターの使用 752
START コマンドによる印刷 753
一時データでの印刷 753
BMS ルーティングによる印刷. 755
非 CICS プリンターの使用. 755
非 CICS プリンターの書式設定 755
非 CICS プリンター: データの送達 755
非 CICS プリンターに関するプログラミング 756
印刷アプリケーションの通知 758
表示画面の印刷. 758
CICS 印刷キー 758
ISSUE PRINT および ISSUE COPY 759
ハードウェア印刷キー 760
BMS 画面コピー 760

第 59 章 JES に対する CICS インターフェース. 761
JES に対する CICS インターフェースの使用. . . 763
スプール・インターフェースに関する制約事項 763
出力スプール・ファイルの作成 764
MVS 内部読み取りプログラムの使用 765
入力スプール・ファイルの読み取り 765
JES 出口 766
スプール・ファイルの識別 767
SPOOL コマンドの例. 769
COBOL 769
PL/I 770
C 771
アセンブラー 771

第 15 部 基本マッピング・サポート (BMS) 773

第 60 章 基本マッピング・サポート 775
BMS サポート・レベル 775
最小機能 BMS 776
標準機能 BMS 776
全機能 BMS. 777
BMS の出力例 777

第 61 章 マップの作成 781
マップ・フィールドの定義: DFHMDF 781
マップの定義: DFHMDI 784
マップ・セットの定義: DFHMSD. 784
BMS マクロの作成 785
マップのアセンブル 788
物理的およびシンボリックのマップ・セット . . 788
SDF II による代替 789
マップ・セットへのマップのグループ化 . . . 789
アプリケーション・データ構造 (ADS) 790
複合フィールドの使用 791
複合フィールド: GRPNAME オプション . . . 791
反復フィールド: OCCURS オプション 792
ブロック・データ 793
3270 以外の端末のサポート 794
3270 以外の装置の出力に関する考慮事項 . . . 795
入力における差異 795
3270 以外の端末の特殊オプション 796
装置依存マップ. 797
装置依存サポート: DDS 798
ご使用の端末情報の検出. 800

第 62 章 BMS マップ出力の送信 . . . 801
マップ用ストレージの獲得および定義 801
BASE および STORAGE オプション 802
出力マップの初期化 803
変数データのマップへの移動 803
表示特性の設定. 804
属性の変更 805
属性値の定義: DFHBMSCA. 806

第 63 章 SEND MAP コマンドの使用 807
SEND MAP の制御オプション 807
その他の BMS SEND オプション: WAIT および LAST 808
シンボリック・マップと物理マップの組み合わせ 808
MAPONLY オプション 809
DATAONLY オプション. 809
SEND CONTROL コマンド. 809
出力画面の作成. 810
開始時の実行内容 810

送信される内容. 811
値の取得元 811
マップの外側 812
GDDM および BMS の使用 813
カーソルの位置決め 813
無効データおよびその他のエラーの送信 814
出力後処理オプション: TERMINAL、SET、および PAGING 814
SET の使用 815

第 64 章 マップ・データの受信 817
入出力の例 817
記号入力マップ. 819
マップ入力のプログラミング 820
RECEIVE MAP コマンドの使用 821
マップ入力用ストレージの取得 821
形式設定画面の入力 822
データの変更 823
大文字変換 823
アテンション ID の使用. 824
HANDLE AID コマンドの使用 824
カーソルの検出. 825
マップ入力の処理 826
入力エラーの取り扱い 826
フラグのエラー. 827
正しい入力の保管 827
再検査. 828
マップ入力後のマップ出力の送信. 828
MAPFAIL およびその他の例外状態 829
EOC 状態 830
その他の入力の形式設定. 830

第 65 章 BMS の論理メッセージ . . . 833
論理メッセージの作成 833
SEND PAGE コマンド 834
RETAIN および RELEASE 835
AUTOPAGE オプション. 836
端末オペレーターのページング: CSPG トランザクション. 836
論理メッセージのリカバリー 837

第 66 章 累積出力 — ACCUM オプション 839
浮動マップ: ACCUM を使用した BMS におけるマップの配置 839
改ページ: BMS のオーバーフロー処理 840
マップ配置の規則 841
累積処理用の ASSIGN オプション 843
複合画面からの入力 843
パフォーマンスの考慮事項 844
パス長の最小化. 844
メッセージ長の削減 845

第 67 章 テキスト出力 847
SEND TEXT コマンド 847
テキスト論理メッセージ. 847
テキスト・ページ 848
テキスト行 849
ヘッダーおよびトレーラーの形式. 850
SEND TEXT MAPPED および SEND TEXT NOEDIT 851

第 68 章 メッセージ・ルーティング 853
メッセージの宛先 853
適格である端末. 854
OPCLASS のみで指定された宛先. 855
OPCLASS および LIST の省略 855
提供される経路リスト 855
経路リストの形式 856
メッセージの送達 858
送達不能なメッセージ 859
リカバリー可能メッセージ 860
メッセージの識別 860
ルーティングに関するプログラミングの考慮事項 861
ルーティングとページ・オーバーフロー . . . 861
SET でのルーティング 862
メッセージ・ルーティングによる会話のインターリービング 862

第 69 章 MAPPINGDEV 装置. 863
MAPPINGDEV オプションを指定した SEND MAP 863
MAPPINGDEV オプションを指定した RECEIVE MAP 864
MAPPINGDEV アセンブラー・アプリケーションの例 865

第 70 章 区分画面サポート. 867
区分化画面の使用法 868
スクロール 868
データ入力 868
ルックアサイド. 869
データの比較 869
エラー・メッセージ 869
区画定義 869
3290 の文字サイズ 870
区分化の設定 871
BMS SEND コマンドの区分画面オプション . . . 872
アクティブ区分画面の判別 872
BMS RECEIVE コマンドの区分画面オプション . . 872
区分画面の ASSIGN オプション 873
区分画面と論理メッセージ 873
区分画面とルーティング. 874
アテンション ID および例外条件. 874
端末の共用 875

第 71 章 特殊ハードウェアのサポート 877
論理デバイス・コンポーネント 877
論理デバイス・コンポーネントの定義 877
論理デバイス・コンポーネントへのデータの送信 878
LDC および論理メッセージ 878
LDC とルーティング. 879
10/63 磁気スロット読み取り装置 879

フィールド選択の機能 880
トリガー・フィールドのサポート. 880
カーソルおよびペンで検出可能なフィールド . . . 881
選択フィールド. 881
アテンション・フィールド 882
検出可能フィールドからの BMS 入力 882
外部形式制御 883

第 72 章 BMS: パフォーマンスの設計 885
ページの作成およびルーティングの操作 888

第 16 部 付録 891

特記事項. 893
商標 894

参考文献. 895
CICS Transaction Server for z/OS の CICS ブック 895
CICS Transaction Server for z/OS の CICSPlex SM ブック. 896
他の CICS 資料 896
その他の IBM 資料 897

アクセシビリティー 899

索引 901

前書き

本書について

この資料には、プログラムを作成するユーザーが バージョン 4 リリース 1 のサービスを使用するためのプログラム・インターフェースが記述されています。

本書は、CICS® EXEC アプリケーション・プログラムを使用して CICS のサービスやリソースにアクセスする、プロシージャー型アプリケーション・プログラムの開発についての**ガイダンス** です。「*CICS アプリケーション・プログラミング・リファレンス*」の**参照** 情報を補完する内容になっています。CICS アプリケーションなどのデバッグについては、「*CICS Problem Determination Guide*」を参照してください。Java™ 言語を使用したアプリケーション・プログラムの開発については「*Java Applications in CICS*」を、CICS OO クラスの使用については「*CICS C++ OO Class Libraries*」を参照してください。

本書の対象読者

本書は、経験豊富なアプリケーション・プログラマーを対象にしています。CICS に対して比較的初心者の方でも理解することができます。システム・プログラマーやシステム・アナリストに対しても有用です。

本書を理解する上での前提事項

「*Designing and Programming CICS Applications*」レベルの CICS アプリケーション・プログラミングの基本的な知識があれば、COBOL、C、C++、PL/I、またはアセンブラー言語でプログラムを作成できます。

本書の使用方法

知りたいことが書かれている部分をお読みください。(各部には、検索に役立つ完全な目次があります。) 本書は、ガイドブックであり、解説書ではありません。最初にお読みになる際には、各部の最初から終わりまで簡単に一読しておくことをお勧めします。

用語に関する注意

API 特に明記されていない場合は、CICS のコマンド・レベル・アプリケーション・プログラミング・インターフェースのこと。

ASM アセンブラー言語の省略形として使用されることがある。

MVS™ オペレーティング・システムのこと。z/OS®、OS/390®、または MVS/Enterprise System Architecture System Product (MVS/ESA™ SP™) などがある。

VTAM®
ACF/VTAM のこと。

本書で記述されているサンプル・プログラムでは、ドル記号 ($) は国の通貨記号であり、EBCDIC コード・ポイント X'5B' が割り当てられていると見なされます。一

© Copyright IBM Corp. 1989, 2009

部の国では、別の通貨記号 (例えば、ポンド記号 (£) や円記号 (¥)) も同じ EBCDIC コード・ポイントと見なされます。上記の国では、適切な通貨記号がドル記号の代わりに使用されます。

本書で取り上げていない内容

CICS フロントエンド・プログラミング・インターフェースの使用については、本書で説明しません。FEPI 設計の考慮事項に関するバックグラウンド情報およびその API に関するプログラミング情報については、「*CICS Front End Programming Interface User's Guide*」を参照してください。

EXEC CICS WEB コマンドの使用については、本書では説明しません。この情報については、「*CICS インターネット・ガイド*」を参照してください。

オブジェクト指向プログラミング言語および技法については、本書では説明しません。Java 言語を使用したアプリケーション・プログラムの開発については「*Java Applications in CICS*」を、CICS OO クラスの使用については「*CICS C++ OO Class Libraries*」を参照してください。

CICS Transaction Server for z/OS バージョン 4 リリース 1 の変更点

このリリースに加えられた変更点に関する情報は、インフォメーション・センターの「*リリース・ガイド*」または以下の資料を参照してください。

- *CICS Transaction Server for z/OS リリース・ガイド*
- *CICS Transaction Server for z/OS V3.2 からのアップグレード*
- *CICS Transaction Server for z/OS V3.1 からのアップグレード*
- *CICS Transaction Server for z/OS V2.3 からのアップグレード*

© Copyright IBM Corp. 1989, 2009

第 1 部 CICS アプリケーションの作成

© Copyright IBM Corp. 1989, 2009

第 1 章 概要: CICS アプリケーションの作成

CICS アプリケーション、および CICS で実行されるプログラムの作成の手順について簡単に説明します。

CICS アプリケーションとは

アプリケーションとは、製品注文の処理や会社の給与計算の作成などのビジネス・オペレーションを協働して実行する、関連したプログラムのコレクションです。CICS アプリケーションは、プログラムやファイルへのアクセスを行う CICS のサービスとインターフェースを使用して、CICS 制御の下で実行されます。

CICS は、トランザクション処理サブシステムです。つまり、CICS は、アプリケーションをオンラインで実行するサービスを要求に応じてユーザーに提供し、それは他の多数のユーザーが同じファイルとプログラムを使う同じアプリケーションの実行要求を実行依頼するのと同時に行われます。CICS は、リソースの共用、データの保全性および実行の優先順位付けをすばやい応答で管理します。

CICS アプリケーションは、従来、**トランザクション**要求の実行依頼により実行されます。トランザクションの実行は、必要な機能をインプリメントした 1 つ以上の**アプリケーション・プログラム**の実行で構成されます。CICS の資料では、CICS アプリケーション・プログラムが**プログラム**と呼ばれたり、トランザクションという用語が、アプリケーション・プログラムによって実行される処理を意味するために用いられることがあります。

現在、**トランザクション**という用語が、IT 産業において、**リカバリー単位**または CICS でいう**作業単位**を表現するのに、広く使用されていることに注意する必要もあります。トランザクションは、一般的にはリカバリー可能な完全な操作のことで、プログラム式コマンドまたはシステム障害の結果として、全体でのコミットまたはバックアウトが可能です。多くの場合、CICS トランザクションが意味する範囲には単一の作業単位もありますが、CICS の資料を読む際は、意味の違いに注意する必要があります。

CICS プログラム、トランザクション、およびタスク

CICS アプリケーションを開発および実行するには、CICS プログラム、トランザクション、およびタスクの間の関係を理解する必要があります。

これらの用語は CICS の資料全般において使用され、多くのコマンドに現れます。

トランザクション

トランザクションは、単一の要求によって開始される 1 つの処理です。通常、要求は端末のエンド・ユーザーから出されますが、Web ページや、リモート・ワークステーション・プログラム、別の CICS システムのアプリケーションから出される場合や、あるいは事前定義された時刻に自動的に起

© Copyright IBM Corp. 1989, 2009

動される場合もあります。「*CICS インターネット・ガイド*」および「*CICS 外部インターフェース・ガイド*」では、CICS トランザクションのさまざまな実行方法について説明します。

単一トランザクションは、実行時に必要な処理を実行する 1 つ以上のアプリケーション・プログラムで構成されています。

ただし、CICS において、トランザクションという用語は、単一イベントおよび同タイプの他のすべてのトランザクションのどちらを意味する場合にも使用されます。CICS に対し、それぞれのトランザクション・タイプを、TRANSACTION リソース定義を使用して記述します。この定義によって、トランザクション・タイプに名前が付けられ (トランザクション ID または TRANSID)、最初に起動するプログラム、トランザクションの実行全般にわたって必要な認証の種類など、実行する作業についてのいくつかの事項が CICS に伝えられます。

トランザクションは、その TRANSID を CICS に発信して実行します。CICS は TRANSACTION 定義に記録された情報を使用して、正しい実行環境を確立し、最初のプログラムを始動します。

現在、トランザクションという用語は、IT 産業において、**リカバリー単位**または CICS でいう**作業単位**を表現するのに、広く使用されています。トランザクションは、一般的にはリカバリー可能な完全な操作のことで、プログラム式コマンドまたはシステム障害の結果として、全体でのコミットまたはバックアウトが可能です。多くの場合、CICS トランザクションが意味する範囲には単一の作業単位もありますが、CICS 以外の資料を読む際は、意味の違いに注意する必要があります。

タスク タスクという用語も、CICS の資料で広く使用されています。CICS では、この用語も特別の意味を持っています。CICS はトランザクションの実行要求を受け取ると、そのトランザクション・タイプの実行の 1 つのインスタンスに関連した新規のタスクを始動します。すなわち、特定の一組のデータを使用したトランザクションの 1 つの実行で、通常、特定端末の特定ユーザーのための実行です。スレッドに類似するものと見なすこともできます。トランザクションが完了すると、タスクも終了します。

CICS プログラミング

CICS プログラムの作成方法は、他のプログラムを作成する場合とほとんど同じです。CICS アプリケーション・プログラムは、COBOL、C、C++ 、Java、PL/I、またはアセンブラー言語を使用して作成することができます。ほとんどの処理ロジックは標準言語ステートメントで表しますが、CICS サービスを要求する場合は CICS コマンド、あるいは Java および C++ のクラス・ライブラリーを使用します。

この情報では、COBOL、C、C++、PL/I またはアセンブラーのプログラムで使用可能な CICS コマンド・レベル・プログラミング・インターフェース EXEC CICS の使用について説明します。これらのコマンドについては、「*CICS アプリケーション・プログラミング・リファレンス*」で詳しく定義されています。以下のプログラミング情報も使用可能です。

- JCICS クラス・ライブラリーを使用した Java によるプログラミングについては、「*Java Applications in CICS*」のJCICS を使用した Java プログラミングで説明されています。
- CICS C++ クラスを使用した C++ によるプログラミングについて詳しくは、*CICS C++ OO Class Libraries*を参照してください。
- HTTP の要求および応答を処理する Web アプリケーション作成については、「*CICS インターネット・ガイド*」のCICS Web サポートの概念と構造を参照してください。

CICS で使用する言語の詳細なガイダンスは、25 ページの『第 3 章 COBOL でのプログラミング』、 53 ページの『第 4 章 C および C++ でのプログラミング』、および65 ページの『第 5 章 PL/I でのプログラミング』を参照してください。

CICS では、プログラムの中で、CICS コマンドの他に、SQL ステートメント、DLI 要求、CPI ステートメント、および CICS フロントエンド・プログラミング・インターフェース (FEPI) コマンドを使用できます。これらを使用する場合には、以下のマニュアルを参照する必要があります。

- **SQL:** 「*DB2 Universal Database for z/OS SQL 解説書*」、SC88-8772、および「*DB2 Universal Database for z/OS アプリケーション・プログラミングおよび SQL ガイド*」、SC88-8763
- **DL/I:** 「*IMS: アプリケーション・プログラミング: EXEC DLI コマンド (CICS および IMS)*」、SD88-6267、および 「*IMS: アプリケーション・プログラミング: データベース管理プログラム*」、SD88-6265
- **CPI:** 「*IBM SAA: CPI Reference*」 マニュアル、および 「*SAA Common Programming Interface for Resource Recovery Reference*」 マニュアル
- **FEPI:** 「*CICS フロントエンド・プログラミング・インターフェース 使用者の手引き*」

CICS プログラミング・コマンド

CICS コマンドの一般的な形式は、EXECUTE CICS (または EXEC CICS) とそれに続く必要なコマンド名であり、1 つ以上のオプションがあればさらにそれに続きます。

CICS 制御ブロックおよびストレージ域のフィールドについての知識や解説書がなくても、CICS コマンド・レベル・インターフェースを使用すれば、多くのアプリケーション・プログラムを作成することができます。しかし、ユーザー・アプリケーション・プログラムのローカル環境の外で有効な情報を入手することが必要になることがあります。

ADDRESS コマンドおよび ASSIGN コマンドを使用すると、該当する情報にアクセスできます。

ADDRESS コマンドおよび ASSIGN コマンドを使用すると、各種のフィールドを読み取ることができますが、それらを他の方法で設定したり使用することはできません。これは、CICS フィールドは EXEC インターフェース・モジュールによって変更されることがあるので、CICS コマンドの引数として CICS フィールドを使用してはいけないということです。

INQUIRE コマンド、SET コマンド、および PERFORM コマンドによって、アプリケーション・プログラムは CICS リソースについての情報にアクセスすることができます。これらのコマンドは、システム・プログラミング・コマンドとして知られています。アプリケーション・プログラムは、CICS データ・セット、端末、システム項目、モード名、システム属性、プログラム、およびトランザクションについての情報の検索および修正を行うことができます。これらのコマンドと JES への CICS インターフェースのスプール・コマンドは、基本的に、システム・プログラマーが使用するためのコマンドです。

EXEC インターフェース・ブロック (EIB)

コマンド・レベル環境の各タスクには、通常の CICS 制御ブロックに加えて、タスクに関連した EXEC インターフェース・ブロック (EIB) と呼ばれる制御ブロックがあります。

アプリケーション・プログラムは、名前によって EIB の全フィールドにアクセスすることができます。 EIB には、アプリケーション・プログラムの実行時に有用な、トランザクション ID、時刻および日付 (最初はタスクの開始時点、以後は、アプリケーション・プログラムによって ASKTIME を使用して更新された場合)、およびディスプレイ装置のカーソル位置などの情報が入ります。また、EIB には、プログラムをデバッグするためにダンプを使用する場合に役立つ情報も入ります。EIB フィールドに関するプログラミング情報については、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

変換

ほとんどのコンパイラー (およびアセンブラー) は、CICS コマンドを直接処理できません。つまり、ユーザー・プログラムを実行可能コードに変換する追加のステップが必要です。このステップは**変換**と呼ばれ、CICS コマンドを、プログラムの残りの部分をコーディングしている言語に変換して、コンパイラー (またはアセンブラー) が解釈できるようにします。

CICS サービス・ルーチンの呼び出しが可能になるように、CICS コマンドを自動的に解釈して変換する**統合変換プログラム**が、一部のコンパイラーに含まれるようになりました。これらのコンパイラーのいずれかを使用する場合、 84 ページの『変換のプロセス』で説明する変換タスクを実行する必要はありません。

CICS は、EXEC CICS および EXEC DLI の両方のステートメントを処理する変換プログラムを、ユーザーがプログラムを作成する場合に使用することがあるそれぞれの言語ごとに提供します。

変換処理にはいくつかのオプションを指定することができ、プログラムのタイプによっては、指定する必要がある場合があります。例えば、EXEC DLI を使用する場合には、変換プログラムにこの事実を指示する必要があります。 88 ページの『CICS 変換プログラムの使用』では、オプションの指定方法について説明し、 90 ページの『変換プログラムのオプションの定義』では、使用可能なオプションを定義します。

CICS のテスト

このタスクについて

ユーザーのプログラムは、次の 2 種類の方法で、それが CICS で実行されるかどうかを自ら判断することができます。

iscics
既存の C 言語プログラムを適用したり、CICS だけでなく CICS 以外でも動作するように設計された新規プログラムを作成したりする場合には、C 言語の iscics() 関数が便利です。この関数は、プログラムが現在 CICS で実行されている場合には非ゼロ値を、それ以外の場合にはゼロを返します。この関数は、C ライブラリーの拡張版です。

DFH3QSS
ユーザーのプログラムは、DFH3QSS プログラムを呼び出して、CICS 環境と API 機能を照会することができます。DFH3QSS を静的に所有アプリケーションにリンクします。この照会に対して、レジスター 15 には、ハーフワード長 (それ自体を含む)、それに続く予約済みハーフワード (現在はゼロ)、およびそれに続くビット・ストリング (下記を参照) から構成される結果構造のアドレスが指定されます。

ビット 0
1 に設定されている場合、呼び出し側プログラムが (CICS 管理 TCB またはその子孫のいずれかにおける) CICS 環境で実行されていることを意味します。

ビット 1
1 に設定されている場合は、(現 PSW キー、ASC モード、AMODE および仮想記憶間環境において) 呼び出し側にとって CICS API が使用可能であることを意味します。

要求が発行された TCB が終了しておらず、なおかつ DFH3QSS 自体が仮想記憶域にまだ存在する限り、出力構造はアクセス可能なままです。実行状態 (現 PSW キー、ASC モード、AMODE、または仮想記憶間環境など) のどんな変化も CICS API の可用性に影響する可能性があります。レジスターは保存されます。

CICS プログラミング・ロードマップ

このタスクについて

EXEC CICS コマンド・レベル・プログラミング・インターフェースを使用する CICS アプリケーションを開発するには、以下のステップを実行してください。

1. 使用する CICS リソースおよびサービスを確認して、アプリケーションを設計する。CICS アプリケーションの設計についてのガイダンスは、 241 ページの『第 17 章 アプリケーション設計』および 271 ページの『第 18 章 パフォーマンスの設計』を参照してください。
2. 選択した言語で、CICS サービスを要求する EXEC CICS コマンドを組み込んで、プログラムを作成する。CICS コマンドのリストについては、 「*CICS* アプリケーション・プログラミング・リファレンス」を参照してください。

3. 81 ページの『統合 CICS 変換プログラム』を取り込んだコンパイラーを使用している場合は、 120 ページの『プログラムのインストール・ステップ』で説明する処理を使用した、プログラムのコンパイルと CICS へのインストールのみを行う。統合変換プログラムを使用しない場合は、 88 ページの『CICS 変換プログラムの使用』で説明する処理を使用して、ユーザー・プログラム用の変換オプションを定義し、その後プログラムの変換とコンパイルを実行し、さらに 120 ページの『プログラムのインストール・ステップ』で説明する処理を使用して、CICS にインストールする必要があります。
4. PROGRAM リソース定義および TRANSACTION リソース定義を使用してプログラムおよび関連したトランザクションを定義する。これについては、「*CICS Resource Definition Guide*」で説明されています。
5. ファイル、キュー、端末など、プログラムで使用する CICS リソースをすべて定義する。
6. 「*CICS Resource Definition Guide*」で説明している CEDA INSTALL コマンドを使用して、リソースを CICS に認識させる。
7. CICS 端末からトランザクション ID を入力するか、「*CICS External Interfaces Guide*」や、「*CICS インターネット・ガイド*」で説明する方法のいずれかを使用して、プログラムを実行する。

第 2 部 プログラミング言語と言語環境

Language Environment®に関するプログラマー向けの情報、および COBOL、C、PL/I およびアセンブラーでのプログラミングに関する言語固有のガイダンスです。

© Copyright IBM Corp. 1989, 2009

第 2 章 言語環境プログラム

z/OS の要素として提供される言語環境プログラムにより、ランタイム・ライブラリーの共通セットが提供されます。言語環境プログラムの導入以前は、高水準言語 (HLL) のそれぞれが、個別にランタイム環境を提供する必要がありました。言語環境プログラムを使用すると、プログラム言語やシステム・リソース要件とは関係なく、ご使用のアプリケーションで同じランタイム環境を使用できます。これは、システム依存関係のほとんどが除去されるためです。

言語環境プログラムにより提供されるランタイム・ライブラリーは、VS COBOL II、OS PL/I、および C/370™ などの、従来のコンパイラーで提供されるネイティブ・ランタイム・ライブラリーに置き換わります。共通環境には、2 つの重要な利点があります。

1. 単一のプログラムで、CICS がサポートするすべての言語を混合することができる。
2. すべてのプログラムで、同じ言語環境プログラム呼び出し可能サービスが使用できる。これによって、例えば次のようなことが可能になります。
   - PL/I プログラムで、言語環境プログラム呼び出し可能サービスを使用して取得したストレージで作成したリンク・リストを後で処理したり、COBOL ルーチンから呼び出し可能サービスを使用して、ストレージを解放したりすることができる。
   - 一連のレポートで使用する通貨記号を、レポート自体は COBOL プログラムで作成していても、アセンブラー・ルーチンで設定することができる。
   - 異なる言語で作成されているプログラムからのシステム・メッセージを、すべて同じ出力宛先に送信する。

詳しくは、「*z/OS Language Environment 概念*」を参照してください。このような利点により、CICS での高水準言語サポートは、言語環境プログラムに依存しています。

CICS プログラミングに関するガイド資料では、ご使用の CICS システムで言語環境プログラムのサービスを使用することを必要としています。言語環境プログラムにより、アセンブラー、および CICS のサポートする高水準言語 (HLL) (COBOL、PL/I、C、および C++) についての IBM® インプリメンテーション用の共通ランタイム環境が提供されます。

CICS は、言語環境プログラムによりサポートされるコンパイラーのほとんどを使用してコンパイルされるアプリケーション・プログラムをサポートします。CICS Transaction Server for z/OS の現在のリリースでサポートされるコンパイラーのリストについては、「*CICS Transaction Server for z/OS リリース・ガイド*」のを参照してください。

CICS および言語環境プログラムによりサポートされるコンパイラーのほとんどは、言語環境プログラム準拠のコンパイラーです。つまり、これらのコンパイラーによりコンパイルされるプログラムは、CICS 領域で使用可能な言語環境プログラムのすべての機能の利点を利用できます。また、CICS および言語環境プログラムは、言語

© Copyright IBM Corp. 1989, 2009

環境プログラム以前の (言語環境プログラム準拠ではない) 一部のコンパイラーによりコンパイルされたプログラムもサポートします。しかし、CICS は、言語環境プログラムによりサポートされる、言語環境プログラム以前のコンパイラーをすべてサポートするわけではありません。

言語環境プログラム以前のコンパイラーを使用してコンパイルおよびリンクされたアプリケーションは、通常、言語環境プログラムにより提供されるランタイム・サポートを使用して正常に実行されます。通常、再コンパイル、または再リンク・エディットは必要ありません。一部の環境では、これらのアプリケーションが適切に実行されるように、言語環境プログラムのランタイム・オプションを調整する必要があります。使用中の言語について詳しくは、「*z/OS Language Environment ランタイム・アプリケーション マイグレーション・ガイド*」、および「*コンパイラーおよびランタイム マイグレーション・ガイド*」を参照してください。言語環境プログラム以前のコンパイラーは、言語環境プログラム準拠ではないため、これらのコンパイラーによりコンパイルされるプログラムは、CICS 領域で使用可能な一部の言語環境プログラムの機能の利点を利用できません。

言語環境プログラム以前のコンパイラーで提供されるネイティブ・ランタイム・ライブラリーは、サポートされません。言語環境プログラムのライブラリー以外の言語ライブラリーは、CICS 始動 JCL 内に含めないようにしてください。

既存のアプリケーション・プログラムを変更するか、または新規プログラムを作成する場合は、言語環境プログラムによりサポートされているコンパイラーを使用する必要があります。このためには、アプリケーション・プログラムを、言語環境プログラムの SCEELKED ライブラリーを使用してリンク・エディットする必要があります。この結果作成されるアプリケーション・ロード・モジュールは、言語環境プログラム下でのみ実行できます。

CICS Transaction Server for z/OS では、言語環境プログラム準拠のアセンブラー MAIN プログラムを生成できます。アセンブラー・プログラムについて詳しくは、71 ページの『第 6 章 アセンブラー言語でのプログラミング』を参照してください。

言語環境プログラムの呼び出し可能サービス

言語環境プログラムは呼び出し可能サービスを提供します。このサービスは、CICS の元で実行されるプログラムによりアクセスできます。

言語環境プログラムにより提供される呼び出し可能サービスは、以下の各カテゴリーに分類されます。

ストレージ・サービス
このサービスを使用すると、言語環境プログラムのヒープからストレージの割り振りおよび解放ができます。

エラー処理サービス
エラーを処理するための情報を取得するには、一般にこのサービスを使用します。

メッセージ・サービス
メッセージを処理したり発行したりするには、一般にこのサービスを使用します。

日時
日時を表す値の読み取り、計算、および書き込みができます。言語環境プログラムには独自のパターン・マッチング機能があります。この機能によって、入力レコードに含まれる日時形式、あるいはオペレーティング・システム・サービスが作成する日時形式の、ほとんどすべてを処理することができます。

各国語サポート
この機能によって、言語環境プログラムの出力 (メッセージ、RPTOPTS レポート、RPTSTG レポート、ダンプなど) を、特定の国用にカスタマイズすることができます。

ロケール
この機能を使用すると、ロケール名を指定して、ある文化に特有の出力を、特定の国語、国、およびコード・セット向けにカスタマイズすることができます。

概説
呼び出し可能サービスをまとめたもので、特定の言語環境プログラム機能 (例えばダンプ) に直接関係するものではありません。

数学
これによって、標準的な計算が実行できます。

これらのサービスは、通常、言語環境プログラムに準拠したコンパイラーを使用してコンパイルされたプログラムで**のみ**使用可能です。例外的に、VS COBOL II プログラムでは、日時呼び出し可能サービスに対する動的呼び出しを行うことができますが、言語環境プログラムの呼び出し可能サービスに対するそれ以外の呼び出しは、静的にしろ動的にしろ、サポートされていません。

これらのサービスに関する情報について詳しくは、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。いずれかのサービスを呼び出すのに必要な構文について詳しくは、「*z/OS Language Environment プログラミング・リファレンス*」を参照してください。

メッセージおよびダンプ・サービス

言語環境プログラムのサービス CEEMOUT (メッセージのディスパッチ) および CEE3DMP (ダンプの生成) が CICS の元で実行されている場合、そのメッセージおよびダンプは、通常の宛先ではなく、CESE と呼ばれる一時データ・キューに送られます。

言語環境プログラムの通常の宛先は、メッセージの場合は、MSGFILE ランタイム・オプションで指定した *ddname* であり、ダンプの場合は、CEE3DMP サービスの *fname* 引数で指定した *ddname* です。CICS は、これらの *ddnames* をいずれも無視します。

言語環境プログラムの異常終了および条件処理

言語環境プログラムの異常終了処理は、CICS HANDLE ABEND の使用に応じて行います。CICS HANDLE ABEND がアクティブでない場合に、ユーザー作成条件ハンドラーを使用することがあります。言語環境プログラムは、CICS が定義する例外条件の処理、またはアテンション ID (AID) の検出にはかかわりません。

異常終了処理

CICS アプリケーションが言語環境プログラムの下で実行されている場合、異常終了のためのタスクがスケジュールされているときにとられる処置は、CICS HANDLE ABEND がアクティブであるかどうかに応じて異なります。

HANDLE ABEND がアクティブであれば、言語環境プログラムの条件処理は、異常終了またはプログラム割り込みを制御せず、CEEHDLR で設定したユーザー作成の条件ハンドラーは無視されます。その代わり、CICS HANDLE ABEND で定義した処置が行われます。

CICS HANDLE ABEND がアクティブでない場合、言語環境プログラムの条件処理は、ランタイム・オプション TRAP(ON) が指定されていると、異常終了およびプログラム割り込みを制御**します**。その後、通常の言語環境プログラムの条件処理が実行されます。TRAP(OFF) が指定されている場合は、エラー処理は行われません。異常終了が先行します。言語環境プログラムの通常の条件処理で行われる処置について詳しくは、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。

ユーザー作成の言語環境プログラム条件処理ルーチン

言語環境プログラムのランタイム・オプション USRHDLR を使用すると、ユーザー作成条件ハンドラーを最高レベルで登録することができます。サブルーチン CALL の後などの低いレベルで、CEEHDLR サービスを使用して、そのレベルの条件ハンドラーを登録することができます。この低いレベルのハンドラーは、その低いレベルから戻るときに自動的に登録抹消されます。必要であれば、CEEHDLU サービスを使用して明示的に抹消することもできます。スタック・レベルの説明、および USRHDLR ランタイム・オプション、CEEHDLR と CEEHDLU のサービスについて詳しくは、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。

言語環境プログラムのユーザー作成条件ハンドラー (COBOL 以外) を独自に作成する場合は、ほとんどの CICS コマンドを使用して、その条件ハンドラーの実行時にさらに条件が発生するのを防ぐことができます。ただし、NOHANDLE、RESP、または RESP2 オプションを指定してコーディングする必要があります。以下のコマンドだけは使用できません。これらのコマンドは、条件ハンドラーにもそれが呼び出すプログラムにも入れないようにしてください。

- ABEND
- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION

- POP HANDLE
- PUSH HANDLE

NOLINKAGE 変換プログラム・オプションを使用する場合以外は (99 ページの『NOLINKAGE』を参照)、CEEHDLR サービスを使用するルーチンに登録した COBOL ユーザー作成条件ハンドラーの変換に、CICS 変換プログラムを使用しないでください。これは、CICS 変換プログラムが、COBOL プログラムの PROCEDURE DIVISION ヘッダーに、2 つの追加引数、EXEC インターフェース・ブロック (EIB) および COMMAREA を追加するためです。これらの引数は、言語環境プログラムで渡される引数と一致しません。したがって、COBOL 条件ハンドラーには、CICS コマンドを含めることはできません。

しかし、ユーザー作成条件ハンドラーは、サブルーチンを呼び出して、CICS コマンドを実行することができます (これが COBOL ルーチンとなる場合もあります)。このサブルーチンに引数を渡す必要がある場合は、呼び出し側で、その引数の前に 2 つのダミー引数を置いてください。呼び出されたサブルーチンは、まず EXEC CICS ADDRESS EIB(DFHEIPTR) を発行してからでなければ、その他の CICS コマンドを実行することはできません。

言語環境プログラムの条件処理ルーチンに必要なインターフェースについて詳しくは、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。

CICS 条件および アテンション ID (AID) 処理

言語環境プログラムの条件処理では、CICS HANDLE CONDITION コマンド、または HANDLE AID コマンドを使用するアプリケーションの振る舞いは変更されません。言語環境プログラムは、CICS が定義する例外条件の処理にはかかわりません。この例外条件は、CICS でのみ発生し、処理されます。同様に、AID 検出は、言語環境プログラムの影響を受けない CICS 機能です。

言語環境プログラムのストレージ

言語環境プログラムは、実行単位ごとに、CICS から取得するストレージを使用します。各プログラムを初めて使用するとき、言語環境プログラムは CICS に、実行単位作業域 (RUWA) で必要とされるストレージの容量を示します。ストレージの割り振りは、CICS のシステム初期設定パラメーター RUWAPOOL の設定値によって決まります。

RUWAPOOL=NO を指定すると、CICS は、個々の CICS リンク・レベルの開始時に、このストレージ用に GETMAIN を発行し、それを言語環境プログラムに渡して、その制御ブロック、および STACK、LIBSTACK、HEAP などのストレージ域で使用します。ストレージは、トランザクション上で指定されているデフォルト・キーで獲得されます。プログラムが終了すると、ストレージは解放されます (FREEMAIN を使用)。

RUWAPOOL=YES を指定すると、トランザクションを初めて実行するときには RUWAPOOL=NO の場合と同じですが、CICS は、そのトランザクションを実行する際に要求される RUWA の総ストレージのヒストリーを保持します。これはつまり、トランザクションが再度実行された場合、CICS はストレージ全体に単一の GETMAIN (そしてタスク終了時に単一の FREEMAIN) を発行して、RUWAPOOL

を作成するということです。トランザクションが同じパスに従う場合は、CICS は RUWAPOOL からストレージを割り振るので、さらに GETMAIN を発行する必要はありません。CICS リンクが異なっていたり、追加されたりしたために、RUWA にさらにストレージが必要な場合、CICS は GETMAIN を発行してヒストリーを更新します。したがって、次回には、単一 GETMAIN (および FREEMAIN) の容量はより大きくなります。多数の CICS LINK コマンドを発行するトランザクションでは、これでパフォーマンスをかなり向上させることができます。

CICS システム初期化パラメーター AUTODST=YES を指定すると、CICS は言語環境プログラムに対して、動的ストレージ・チューニングがサポート可能であることを示します。

プログラムがランタイム・オプション ALL31(OFF) を指定し、言語環境プログラムが 16MB 境界より下のストレージを使用する必要がある場合には、2 つのストレージ域 (1 つは 16MB 境界より下、もう 1 つは 16MB 境界より上) が割り振られます。

どのアプリケーションでも、必要であれば、CICS GETMAIN コマンドを使用して CICSDATAKEY または USERDATAKEY ストレージを取得することができます。ただし、USER の EXECKEY を備えたプログラムでは、CICSDATAKEY ストレージは使用できません。

言語環境プログラムにおける言語の混合

言語環境プログラムを使用すると、それぞれ異なる高水準ソース言語、およびアセンブラー言語で書かれた、複数のプログラムで構成されるアプリケーションを作成できます。

アセンブラー言語サブルーチンを HLL プログラムから呼び出すことは、ごく簡単なので、よく行われています。ある HLL から呼び出すが、作成は別の HLL で行うサブルーチンの場合は、それよりはるかに慎重に考慮する必要があります。このようなサブルーチンには、言語間通信 (ILC) が含まれています。言語環境プログラム では、ILC アプリケーションは、複数の HLL およびアセンブラー言語 (オプション) で作成されているアプリケーションと定義されます。この主題についての完全な情報は、「*z/OS 言語環境プログラム ILC アプリケーションの作成*」を参照してください。

言語環境プログラムでは、CICS の実行単位内に ILC がある場合は、それぞれのコンパイル単位を言語環境プログラム準拠のコンパイラーでコンパイルしなければならないと規定されています。CICS は、C/C++、COBOL、および PL/I の 3 種類の HLL をサポートしています。ここでは、2 種類の HLL を組み合わせたインターフェースについて考えます。ご使用のアプリケーションに含まれている HLL が 2 つだけの場合は、該当するセクションを参照してください。ご使用のアプリケーションに 3 種類の HLL がすべて含まれている場合は、そのアプリケーション内の個々のインターフェースに対応するセクションを参照してください。

C/C++ および COBOL

言語環境プログラムが C/C++ および COBOL で作成されたルーチン間の ILC をサポートする場合の条件は、以下の点によって決まります。

- 言語が C か C++ か。
- どの COBOL コンパイラーが使用されているか。また、コンパイラー・オプションとして DLL が指定されているかどうか。
- 呼び出しは静的か動的か。
- 呼び出される関数はモジュール内にあるか、または DLL からエクスポートされるか。
- プログラムは再入可能かどうか。
- C プログラムに #pragma リンケージ・ステートメントがあれば、その種類。
- C プログラムがエクスポートするのは関数か変数か。
- C++ プログラムに extern ステートメントがあれば、その種類。

これらすべての結果は、「*z/OS 言語環境プログラム ILC アプリケーションの作成*」で指定されています。ご使用のアプリケーションで C/C++ と COBOL が混合しているかどうかについて詳しくは、この資料を参照してください。

C/C++ および PL/I

CICS では、C/C++ および PL/I アプリケーションのすべてのコンポーネントが再入可能である場合、言語環境プログラムは、C/C++、および PL/I でコンパイルしたルーチン間の ILC を、以下のようにサポートします。

- C/C++ ルーチンは PL/I ルーチンを静的に呼び出すことができ、PL/I ルーチンは C/C++ ルーチンを静的に呼び出すことができる。
- C/C++ ルーチンは、OPTIONS(FETCHABLE) が指定されている PL/I ルーチンを、fetch() を使用して取り出すことができる。呼び出し先ルーチンに CICS コマンドが含まれている場合、C/C++ は、call ステートメントの最初の 2 つのパラメーターとして、EIB と COMMAREA を渡す必要があります。
- PL/I ルーチンが取り出すことができるのは、CICS 変換プログラムで処理されていない C/C++ ルーチンだけである。これは、動的呼び出し中には、変換プログラムが作成したある種の静的フィールドが正しく設定できないためです。

COBOL および PL/I

CICS では、言語環境プログラムは、言語環境プログラムがサポートする COBOL および PL/I コンパイラーを使用してコンパイルしたルーチン間の ILC を、以下のようにサポートしています。

- COBOL ルーチンは PL/I ルーチンを静的に呼び出すことができ、PL/I ルーチンは COBOL ルーチンを静的に呼び出すことができる。
- COBOL プログラムは、OPTIONS(FETCHABLE) が指定されている PL/I ルーチンを動的に呼び出すことができ、PL/I ルーチンは COBOL プログラムを取り出すことができる。

呼び出し先ルーチンに CICS コマンドが含まれている場合、呼び出し元ルーチンは、CALL ステートメントの最初の 2 つのパラメーターとして、EIB と COMMAREA を渡さなければなりません。

アセンブラー言語

- 言語環境プログラム準拠の任意の HLL プログラムから、言語環境プログラム準拠のアセンブラー・サブルーチンを静的または動的に呼び出すことができる。逆に、言語環境プログラム準拠のアセンブラー言語ルーチンは、言語環境プログラムのマクロ CEEFETCH または CEELOAD を使用して、言語環境プログラム準拠の任意のルーチンを静的に呼び出したり、別のルーチン (アセンブラー言語または HLL) を動的にロードしたりすることができます。
- CEELOAD を使用してロードした ILC モジュールは、削除 (解放) できない。
- CEERELES マクロを使用すると、CEEFETCH を使用して取り出した ILC モジュールを解放することができる。
- アセンブラー言語ルーチンを削除するには、そのルーチンを取り出した言語を使用する。解放されるモジュールに PL/I を使用した ILC がない場合、これは、C/C++、COBOL、および PL/I からしか行うことはできません。

さらに、言語環境プログラム準拠の任意の HLL プログラムまたはアセンブラー言語サブルーチンから、準拠していないアセンブラー言語サブルーチンを静的に呼び出すこともできます。ただし、準拠していないアセンブラー言語ルーチンは、言語環境プログラムのマクロが使用できないので、言語環境プログラム準拠の任意のルーチンを静的に呼び出すことも、準拠ルーチンを取り出したりロードしたりすることもできません。

C または C++ を呼び出すアセンブラー言語では、以下のステートメントを組み込む必要があります。

C `#pragma linkage(,OS)`

C++ `extern "OS"`

DL/I

CICS での ILC アプリケーションで DL/I を使用する場合は、EXEC DLI ステートメントまたは CALL xxxTDLI のいずれかを使用した DL/I 呼び出しは、メインプログラムと同じ言語によるプログラムでのみ行うことができます。

言語環境プログラムは、CICS での CALL CEETDLI はサポートしていません。

-

ダイナミック・リンク・ライブラリー (DLL)

z/OS ダイナミック・リンク・ライブラリー (DLL) 機能によって、プログラムおよびデータをロード・モジュールにパッケージ化して (DLL)、他のロード・モジュールからそれにアクセスできるメカニズムが提供されます。

DLL は、DLL の外側から呼び出せるルーチンを表す記号をエクスポートし、他の DLL にルーチンまたはデータ、あるいはその両方を表す記号をインポートすることができます。これにより、ターゲット・ルーチンを参照ルーチンと同じロード・モジュールにリンクする必要がなくなります。アプリケーションが最初に個別の DLL を参照するときに、システムはその DLL を自動的にメモリーにロードします。

考えられる限りの DLL 実行可能モジュールはすべて、CICS に対するプログラム・リソースとして定義しなければなりません。

DLL サポートは、CICS でのアプリケーションで使用できます。使用するためには、「*z/OS Language Environment プログラミング・ガイド*」でリストされているコンパイラーのいずれかを使用してコードをコンパイルします。 DLL の作成および使用について詳しくは、マニュアルを参照してください。

言語環境プログラムのランタイム・オプションの定義

言語環境プログラムは、プログラムの処理を制御するランタイム・オプションを提供します。CICS では、特定プログラムの実行にどのオプションを適用するかは、そのプログラムだけではなく、プログラムの実行方法によっても異なります。

Java プログラムおよび Web または CICS IIOP サービスから開始されるプログラムには、言語環境プログラムの事前初期設定モジュール CEEPIPI が使用されます。これには、CEEDOPT CSECT の独自のバージョンがありますが、そのようなプログラムでは、それぞれのランタイム・オプションをこの CSECT から取得します。

通常の CICS のタスク (端末から開始されるタスクなど) では、以下にリストする方式のいずれかを使用して、言語環境プログラムのランタイム・オプションを設定してください。これらの方式は、処理される順に表示されています。各設定値は、あとの設定値で指定変更することができます。これは、実質的には、優先順位の逆順です。

1. CEEDOPT CSECT。CICS のインストール・デフォルト・オプションはここにあります。この CSECT は CEECOPT サンプルから生成され、CEECCICS ロード・モジュールに取り込まれます。
2. CEEROPT CSECT。領域全体にわたるデフォルト・オプションはここにあります。この CSECT は、同じ名前のロード・モジュールにリンク・エディットされ、CICS ジョブ用の DFHRPL ライブラリー連結のデータ・セットに配置されます。
3. ユーザー置換可能プログラム DFHAPXPO (XPLINK プログラムにのみ適用)。
4. CEEUOPT CSECT。ユーザー提供アプリケーション・プログラム・レベルのランタイム・オプションは、ここにあります。この CSECT は、アプリケーション・プログラム本体とリンクされています。
5. アプリケーション・ソース・コード。ここでは、以下のようにプログラム言語のオプション・ステートメントを使用します。
   - C プログラムでは、プログラム・ソースの #pragma runopts ステートメントを使用。例:

     ```
     #pragma runopts(rptstg(on))
     ```

   - PL/I プログラムでは、プログラム内の PLIXOPT 宣言ステートメントを使用。例:

     ```
     DECLARE PLIXOPT CHARACTER(18) VARYING STATIC EXTERNAL
                     INIT('RPTOPTS(ON) NOSTAE');
     ```

   注: COBOL プログラムまたは C++ プログラムには、ランタイム・オプションの設定を許可するソース・コード・メカニズムはありません。

6. 言語環境プログラム・オプション。このオプションは、デバッグ・プロファイルで指定します。詳しくは、228 ページの『デバッグ・プロファイル』を参照してください。

ほとんどのインストール済み環境の場合、上記のリストの最初の方式はアプリケーション・プログラマーが使用できないもので、2 番目の方式も往々にして使用できません。しかし、アプリケーション・プログラマーは方式 4 または方式 5 を使用できます。いずれか 1 つの方式だけを選択して、方式 4 と方式 5 の両方を使用しないでください。ご使用のアプリケーションとリンクするために CEEUOPT CSECT を生成する方法については、「*z/OS Language Environment カスタマイズ*」を参照してください。

CEEDOPT も CEEROPT も、あとで指定したもので指定変更できないように、任意のオプションを設定することができます。

言語環境プログラムのランタイム・オプションを指定する方法、およびオプションの意味について詳しくは、「*z/OS Language Environment プログラミング・リファレンス*」を参照してください。

CICS の元では無視されるランタイム・オプション

CICS では、言語環境プログラムのランタイム・オプション設定の多くが無視されます。これらは、以下のものも含めて、すべて Fortran 専用のオプションです。

- ABPERC
- AIXBLD
- CBLOPTS
- CBLQDA
- DEBUG
- EXECOPS
- INTERRUPT
- LIBRARY
- MSGFILE
- NONIPTSTACK
- PLITASKCOUNT
- POSIX (XPLINK または Java プログラム以外)
- RTEREUS
- RTLS
- SIMVRD
- THREADHEAP
- VERSION

使用されたランタイム・オプションの判別

プログラムの実行時に言語環境プログラムのどのランタイム・オプションが有効になったかを知りたい場合は、オプション RPTOPTS(ON) を指定します。プログラムの終了時に、使用されたすべてのランタイム・オプションを表示したリストが作成

されます。このリストは、CESE TD キューに書き込まれます。このリストには、オプションの実際の設定値だけではなく、その起点、すなわち、それがそのシステムまたは領域のデフォルトかどうか、あるいはプログラマーによって設定されたのか出口のいずれかで設定されたのかも含まれています。

注: 実稼働環境では RPTOPTS(ON) は使用しないでください。オーバーヘッドが有効になっているので、CESE キューに大量のデータが書き込まれることになります。

子エンクレーブのランタイム・オプション: パフォーマンスの考慮

CICS では、CICS LINK コマンドを実行することによって、言語環境プログラムで子エンクレーブと呼ばれているものが作成されます。新規環境が初期設定され、子エンクレーブは、ランタイム・オプションを取得します。これらのランタイム・オプションは、作成元のエンクレーブにあったオプションからは完全に独立しています。

EXEC CICS LINK を頻繁に使用して、多数のランタイム・オプションを個別に設定すると、パフォーマンスに影響を与えることがあります。静的または動的呼び出しでは、このようなオーバーヘッドは発生しません。オプションを指定するのに CEEUOPT を使用する必要がある場合は、デフォルトとは異なるオプションだけを指定するようにすれば、パフォーマンスが向上します。

CICS XCTL コマンドを実行するときにも、同様のことが起こります。この場合は、子エンクレーブは取得されませんが、新規プログラム用のランタイム・オプションが決定されると、既存のエンクレーブは、いったん終了してから再び初期設定されます。同じパフォーマンス考慮事項が適用されます。

CEEBXITA および CEECSTX ユーザー出口

言語環境プログラムのこれら 2 つのユーザー出口では、言語環境プログラムの一部のランタイム・オプションを変更できます。

- CEEBXITA ユーザー出口の CEEAUE_A_OPTION リターン・パラメーターを設定すると、オプションを変更できます (LIBRARY、RTLS、STACK、VERSION の各オプションを除く)。
- ストレージ・チューニング・ユーザー出口 CEECSTX。ここでは、オプション STACK、LIBSTACK、HEAP、ANYHEAP、および BELOWHEAP が設定できます。

出口は上記のリストの順序で呼び出されます。

ストレージ・チューニング出口 CEECSTX も、CEEROPT CSECT と同様、領域全体にわたっていますが、CEEBXITA はすべてのプログラムにリンクされています。

言語環境プログラムは、CEECSTX と同様、環境が完全に確立される前に呼び出され、したがってアセンブラーでコーディングする必要があるので、アセンブラー出口 CEEBXITA を呼び出します。

言語環境プログラムでは、SCEESAMP ライブラリーの CEEBXITA のサンプル・ソース版を提供しています (これは、その呼び出し元に、呼び出された理由を返すプログラムです)。これをそのまま使用することも、変更してインストール・デフォル

ト版として使用することもできます。しかし、CEEBXITA の特に調整したバージョンは、どのアプリケーション・プログラムともリンク・エディットすることができ、以後は、インストール・デフォルト・バージョンの代わりにこれが使用されます。このメソッドを選択する場合は、十分注意してください。プログラム実行中に、CEEBXITA は最高で 5 つまでの異なる理由で呼び出されるので、CEEBXITA のアプリケーション固有のバージョンは、これらの呼び出しすべてを処理できる必要があるためです。

CEEBXITA の独自のバージョンを作成する場合は、アセンブラー言語で作成する必要があります。NOHANDLE、RESP、または RESP2 オプションを指定する場合は、すべての CICS コマンド (以下に挙げたものは除く) を使用して、出口の実行時に提示される条件を阻止することができます。これらは、CEEBXITA 内、あるいは CEEBXITA が呼び出す任意のルーチン内では使用できないコマンドです。

- ABEND
- HANDLE ABEND
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- POP HANDLE
- PUSH HANDLE

CEEBXITA および CEECSTX について詳しくは、「*z/OS Language Environment カスタマイズ*」を参照してください。

CICSVAR、CICS 環境変数

CICS は、CICSVAR と呼ばれる環境変数を提供して、CONCURRENCY および API プログラム属性がアプリケーション・プログラム自身と密接に関連付けられることを可能にします。言語環境プログラムのランタイム・オプション ENVAR を使用して、この環境変数を指定できます。

CICSVAR を CEEDOPT CSECT 内で使用して、インストールをデフォルトに設定できますが、個別のプログラムを使用して CEEUOPT CSECT リンク・エディット内で設定するか、C または C++ プログラムのソースの #pragma ステートメントにより設定するか、または PL/I プログラムの PLIXOPT ステートメントにより設定するのが最も便利です。例えば、プログラムがスレッド・セーフ標準でコーディングされている場合、PROGRAM リソース定義を変更する必要なくそのように定義できます。あるいは、インストール時に定義された命名標準に従って、プログラム自動インストール出口が適切な属性を使用してインストールできます。

プログラムが言語環境プログラム準拠のコンパイラーを使用してコンパイルされた場合、PL/I、COBOL、C、および C++ プログラム用 (XPLINK オプションの使用不使用にかかわらずコンパイルされたもの) の言語環境プログラム準拠のアセンブラー言語に対して CICSVAR を使用できます。CICSVAR は、言語環境プログラムに準拠していないアセンブラー言語プログラム、または Java プログラムでは使用できません。

CICSVAR の使用により、標準 RDO インターフェース、またはプログラムの自動インストールを使用してインストールされた PROGRAM リソース定義の設定が指

定変更されます。プログラムを初めて実行する前に INQUIRE PROGRAM コマンドを実行すると、プログラム定義からのキーワード設定が表示されます。アプリケーションを 1 度実行すると、INQUIRE PROGRAM コマンドにより、CICSVAR で指定変更が適用された設定が表示されます。

CICSVAR は、QUASIRENT、THREADSAFE、または OPENAPI の 3 つの値のいずれかにすることができます。

CICSVAR=QUASIRENT
属性 CONCURRENCY(QUASIRENT) および APIST(CICSAPI) を使用したプログラムを生成します。

CICSVAR=THREADSAFE
属性 CONCURRENCY(THREADSAFE) および APIST(CICSAPI) を使用したプログラムを生成します。

CICSVAR=OPENAPI
属性 CONCURRENCY(THREADSAFE) および APIST(OPENAPI) を使用したプログラムを生成します。

CEEUOPT CSECT でコーディングされた、言語環境プログラムのランタイム・オプション ENVAR の例は、以下のとおりです。

```
CEEUOPT CSECT
CEEUOPT AMODE ANY
CEEUOPT RMODE ANY
        CEEXOPT ENVAR=('CICSVAR=THREADSAFE')
        END
```

これはアセンブルされてロード・モジュールにリンク・エディットされ、次に CEEUOPT ロード・モジュールが言語環境プログラムによりサポートされる言語のプログラムを使用してまとめてリンク・エディットされます。

または、C および C++ プログラムの場合、プログラム・ソースの始めの、他の C ステートメントの前に、以下のステートメントを追加します。

```
#pragma runopts(ENVAR(CICSVAR=THREADSAFE))
```

PL/I プログラムの場合、PL/I MAIN プロシージャー・ステートメントの後に、以下のステートメントを追加します。

```
DCL PLIXOPT CHAR(25) VAR STATIC EXTERNAL  INIT('ENVAR(CICSVAR=THREADSAFE)');
```

言語環境プログラムの CEEBINT 出口

言語環境プログラムの下で実行されているプログラムはすべて、プログラムの初期設定時、CEEBXITA および CEECSTX 出口の呼び出し直後に、CEEBINT と呼ばれるサブルーチンを呼び出します。ランタイム環境は、この時点で完全に作動可能です。言語環境プログラムでは、このプログラムは高水準言語 (HLL) ユーザー出口と呼ばれます。

言語環境プログラムは、SCEELKED ライブラリーで、このプログラムを含むモジュールを提供します (その呼び出し元に戻ります)。したがって、これがインストールのデフォルト・バージョンになります。ただし、独自のバージョンを作成して任意のプログラムにリンク・エディットし、デフォルトと置き換えることもできます。

通常の言語環境プログラムのコーディング規則が CEEBINT に適用されます。これは、C、C++、PL/I、または言語環境プログラム準拠のアセンブラー言語で作成できます。CEEBINT は、他のすべてと同様に COBOL プログラムに適用されますが、COBOL プログラムで作成することも、COBOL プログラムを呼び出すこともできません。CEEBINT がプログラムに 2 番目の HLL を導入する場合は、16 ページの『言語環境プログラムにおける言語の混合』で説明した HLL の混合に関する規則が適用されます。

CEEBINT について詳しくは、「*z/OS Language Environment Programming Guide*」を参照してください。

第 3 章 COBOL でのプログラミング

以下の情報を使用すると、CICS アプリケーション・プログラムとして使用する COBOL プログラムをコーディング、変換、およびコンパイルするのに役立ちます。

「*CICS Transaction Server for z/OS リリース・ガイド*」には、CICS Transaction Server for z/OS バージョン 4 リリース 1 でサポートされている COBOL コンパイラー、および z/OS におけるサービス状況がリストされています。

CICS Transaction Server for z/OS バージョン 4 リリース 1 の資料において COBOL に言及している場合は、特に注記されていなければすべて、サポートされる言語環境プログラムに準拠しているコンパイラー、例えば Enterprise COBOL for z/OS などを使用することを意味します。CICS Transaction Server for z/OS バージョン 4 リリース 1 でランタイム・サポートがあるが、言語環境プログラム準拠**ではない**唯一の COBOL コンパイラーは、VS COBOL II コンパイラーです。

COBOL コンパイラー間のマイグレーションについて詳しくは、「*Enterprise COBOL for z/OS: コンパイラーおよびランタイム移行ガイド*」を参照してください。

VS COBOL II のサポート

CICS Transaction Server for z/OS バージョン 4 リリース 1 では、VS COBOL II コンパイラーを使用してコンパイルされたアプリケーションは、言語環境プログラムのランタイム・ライブラリー・ルーチンを使用して実行されます。VS COBOL II で提供されるネイティブ・ランタイム・ライブラリーは、サポートされていません。

31 ページの『VS COBOL II プログラム』で、VS COBOL II コンパイラーを使用してコンパイルされたプログラムに関連する制約事項および考慮事項をいくつかリストします。

一部の環境では、これらのアプリケーションが適切に実行されるように、言語環境プログラムのランタイム・オプションを調整する必要があります。言語環境プログラムのランタイム内での VS COBOL II プログラムの実行、および VS COBOL II プログラムの Enterprise COBOL への変換について詳しくは、「*Enterprise COBOL for z/OS: コンパイラーおよびランタイム移行ガイド*」を参照してください。

OS/VS COBOL のサポート

CICS Transaction Server for z/OS バージョン 4 リリース 1 では、OS/VS COBOL プログラムのランタイム・サポートは廃止されています。OS/VS COBOL プログラムを使用しようとすると、異常終了コード ALIK が発行され、CICS によりタスクは異常終了され、プログラムが使用不可になります。

© Copyright IBM Corp. 1989, 2009

OS/VS COBOL プログラムは、言語環境プログラムに準拠した COBOL にアップグレードし、CICS がサポートする COBOL コンパイラーのレベルに再コンパイルする必要があります。Enterprise COBOL for z/OS バージョン 3 が、推奨コンパイラーです。

OS/VS COBOL プログラムの Enterprise COBOL への変換における注意事項について詳しくは、50 ページの『OS/VS COBOL プログラムのアップグレード』を参照してください。言語の差異、および変換に役立つ機能について詳しくは、「*Enterprise COBOL for z/OS: コンパイラーおよびランタイム移行ガイド*」を参照してください。

OO COBOL のサポート

CICS Transaction Server for z/OS バージョン 4 リリース 1 では、COBOL クラス定義およびメソッド (オブジェクト指向 COBOL) は使用できません。この制限には、Java クラスおよび COBOL クラスの両方が含まれます。

以前の CICS リリースで OOCOBOL 変換プログラム・オプションを使用してコンパイルされたモジュールは、CICS Transaction Server for z/OS バージョン 4 リリース 1 では実行できません。OOCOBOL 変換プログラム・オプションは、以前の SOM ベース (システム・オブジェクト・マネージャー・ベース) の OO COBOL に対して使用されたもので、OO COBOL のこの形式のランタイム・サポートは、z/OS V1.2 で廃止されました。新しい Java ベースの OO COBOL は、Enterprise COBOL で使用されますが、CICS 変換プログラムではサポートされません。

既存の SOM ベースの OO COBOL プログラムが存在する場合、OO COBOL を手続き型 (非 OO) COBOL に書き直し、Enterprise COBOL コンパイラーを使用できるようにする必要があります。新しい Java ベースの OO COBOL は、SOM ベースの OO COBOL プログラムと互換性がなく、SOM ベースの OO COBOL プログラムに対するマイグレーション・パスとして意図するものではないことに注意してください。

作業用ストレージ

コンパイラー・オプション DATA(24) では、WORKING-STORAGE SECTION は 16MB 境界より下のアドレスに割り振られます。コンパイラー・オプション DATA(31) では、WORKING-STORAGE SECTION は 16MB 境界より上のアドレスに割り振られます。

COBOL プログラミングの制約事項および要件

CICS アプリケーション・プログラムとして使用される COBOL プログラムには、いくつかの制約事項および要件があります。

デフォルトでは、CICS 変換プログラムも COBOL コンパイラーも、以下の制約事項の影響を受ける COBOL ワードが使用されていてもそれを検出しません。CICS 環境で制限されているワードを使用すると、実行時に障害を引き起こすことがあります。ただし、COBOL は CICS アプリケーション・プログラム用の予約語テーブル IGYCCICS を提供しています。コンパイラー・オプション WORD(CICS) を指定する場合、コンパイラーは IGYCCICS を使用し、CICS でサポートされていない

COBOL ワードは、コンパイラーによりエラー・メッセージ付きでフラグが立てられます (デフォルトの IBM 提供の予約語テーブルにより通常制限されている COBOL ワードにも、フラグが立てられます)。IGYCCICS により制限されているワードの現行リストについて詳しくは、「*Enterprise COBOL for z/OS: プログラミング・ガイド*」を参照してください。

使用できない関数およびステートメント

- CICS で COBOL の入り口点を使用することはできません。
- ほとんどの入出力処理では、CICS コマンドを使用しなければなりません。したがって、ファイルの記述、または
  OPEN、CLOSE、READ、START、REWRITE、WRITE、DELETE ステートメントのコーディングはしないでください。代わりに、CICS コマンドを使用して、データを検索、更新、挿入、および削除します。
- format-1 の ACCEPT ステートメントを CICS プログラムで使用しないでください。format-2 の ACCEPT ステートメントは、言語環境プログラムを使用可能なコンパイラーによってサポートされます。
- DISPLAY . . . UPON CONSOLE と DISPLAY . . . UPON SYSPUNCH は使用しないでください。システム論理出力装置 (SYSOUT、SYSLIST、SYSLST) に対する DISPLAY はサポートされています。
- STOP「リテラル」を使用しないでください。
- SORT ステートメントの使用には、制約事項があります。詳しくは、「*Enterprise COBOL for z/OS: プログラミング・ガイド*」を参照してください。MERGE を使用しないでください。
- 次の項目は使用しないでください。
  - USE 宣言
  - データ管理に関連した ENVIRONMENT DIVISION 記述項目および FILE SECTION 記述項目。データ管理は、CICS が扱います (これらは、上記で説明した制限された SORT 機能に関連する場合に使用できます)。
  - メインプログラムへの、ユーザー指定のパラメーター。

コーディングの要件

- デバッグ行をコメントとして使用する場合は、対になっていない引用符を含めてはなりません。
- OCCURS DEPENDING ON など、可変長の区域を作成するステートメントは、WORKING-STORAGE SECTION 内では注意して使用する必要があります。
- 宣言のセクションで EXEC CICS コマンドを使用しないでください。
- IDENTIFICATION DIVISION が存在しなければ、CICS コマンドだけが展開されます。IDENTIFICATION DIVISION しか存在しない場合は、
  DFHEIVAR、DFHEIBLK、および DFHCOMMAREA のみが作成されます。
- 言語環境プログラムのランタイムを使用した VS COBOL II プログラムの場合、以下の制限が WORKING-STORAGE の長さに適用されます。
  - コンパイラー・オプション DATA(24) が使用されている場合、制限は 16MB 境界より下の使用可能なスペースになります。
  - コンパイラー・オプション DATA(31) が使用されている場合、制限は 128MB です。

ストレージのアカウンティングおよび保管域のために 80 バイトが必要で、この値は制限内に含まれます。

- DLI オプションが指定され、ENTRY ステートメントが PROCEDURE DIVISION ヘッダーのすぐあとに続く場合は、PROGRAM-ID 名を ENTRY ステートメントのリテラルに変更し、ENTRY ステートメントを削除することを推奨します。
- HANDLE CONDITION または HANDLE AID を使用する場合には、SET(ADDRESS OF A-DATA) または SET(A-POINTER) を使用して、アドレッシングの問題を回避することができます。ここで、A-DATA は LINKAGE SECTION 内の構造体であり、A-POINTER は USAGE IS POINTER 文節を使用して定義されているものです。

言語環境プログラムのコーディングの要件

言語環境プログラムの下の COBOL で書かれた CICS アプリケーションを初めて実行する場合には、ご使用のシステムで使用している言語環境プログラムのランタイム・オプションを見直す必要がある場合があります。特に、アプリケーションが WORKING-STORAGE SECTION を確実に正しく初期設定するように (例えば、マップを送る前に 2 進数のゼロでクリアするように) コーディングされていない場合には、STORAGE ランタイム・オプションを使用する必要があります。言語環境プログラムのランタイム・オプションのカスタマイズについて、詳しくは「*z/OS Language Environment プログラミング・リファレンス*」を参照してください。

31 ビット・アドレッシング

16MB 境界より上で実行される COBOL プログラムには、31 ビット・アドレッシングに対して以下の制約事項が適用されます。

- 受信プログラムが AMODE (31) を使用してリンク・エディットされている場合には、渡されるアドレスの長さは 31 ビットにしなければなりません (または、左端のバイトがゼロに設定されている、24 ビット)。
- 受信プログラムが AMODE(24) を使用してリンク・エディットされている場合には、渡されるアドレスの長さは 24 ビットにしなければなりません。

あるプログラムが 31 ビット・アドレッシング・モードで実行しており、他のプログラムに 24 ビット・アドレッシング・モードでデータ引数を渡している場合、そのプログラムには DATA(24) コンパイラー・オプションを指定します。これにより、呼び出されたプログラムからデータがアドレッシング可能になります。

コンパイラー・オプション

- 次のコンパイラー・オプションは使用しないでください。
  - DLL
  - DYNAM (プログラムが変換される場合)
  - NOLIB (プログラムが変換される場合)
  - NORENT
- 次のコンパイラー・オプションは、CICS 環境に影響を与えません。
  - ADV
  - AWO
  - EXPORTALL

- FASTSRT
- NAME
- OOCOBOL
- OUTDD
- THREAD

- TEST(SYM,NOSEPARATE) コンパイラー・オプションを使用すると、プログラム・サイズの増加が非常に大きくなります。したがって、このオプションを使用した場合には、ストレージ不足の問題が起きることがあります。
  TEST(SYM,SEPARATE) を使用すれば、プログラム・サイズを増加させずに同じ機能を実現できます。TEST コンパイラー・オプションについて詳しくは、「*Enterprise COBOL for z/OS: プログラミング・ガイド*」を参照してください。
- 2 進データ項目を処理するためには、それらが PICTURE 定義に準拠する場合には、TRUNC(OPT) を使用してください。準拠しない場合には、TRUNC(OPT) をコンパイラー・オプションとして使用し、2 進数値が PICTURE 節より大きい可能性がある場合には、項目に対して USAGE COMP-5 の使用が許可されます。TRUNC(BIN) は、ランタイム・パフォーマンスを禁止するので、このオプションは、(例えば、コード生成プログラムにより生成された項目など) 2 進データの項目を制御できない場合にのみ使用します (TRUNC(STD) がデフォルトです)。
- RMODE(24) コンパイラー・オプションの使用は、プログラムが常に 16MB 境界の下にあることを意味するので、推奨されません。代わりに、RMODE(ANY) または RMODE(AUTO) を使用することをお勧めします。RMODE コンパイラー・オプションについて詳しくは、「*Enterprise COBOL for z/OS: プログラミング・ガイド*」を参照してください。

デバッグ・モード

COBOL EXEC CICS コマンドの最初の行の 7 桁目に「D」を入れた場合には、その「D」は変換済み CALL ステートメントの中でも検出されます。この変換済みコマンドが実行されるのは、WITH DEBUGGING MODE が指定されている場合に限られます。「D」を EXEC CICS ステートメントの 1 行目以外の行に置いた場合は、変換プログラムはこれを要求せず、無視します。

言語環境プログラム CBLPSHPOP オプション

CBLPSHPOP ランタイム・オプションは、COBOL サブルーチンが呼び出されるたびに、言語環境プログラムが、初期設定時には EXEC CICS PUSH HANDLE コマンドを、終了時には EXEC CICS POP HANDLE コマンドを、自動的に発行するかどうかを制御します。

ご使用のアプリケーションが、CICS で何度も COBOL サブルーチン呼び出しを行う場合は、CBLPSHPOP(ON) ではなく CBLPSHPOP(OFF) を指定する方が、パフォーマンスが向上します。CBLPSHPOP は、 19 ページの『言語環境プログラムのランタイム・オプションの定義』で説明しているように、CEEUOPT を使用して、個々のトランザクションごとに設定することができます。

ただし、条件処理がスタックされていないので、次の点に留意してください。

- 呼び出し先のルーチンが提示する条件に従って、CICS が呼び出しルーチンの条件処理ルーチンに制御を渡そうとすると、エラーが発生してトランザクションが異常終了する。
- 呼び出し先ルーチンの中で、PUSHable CICS コマンド (HANDLE ABEND、HANDLE AID、HANDLE CONDITION、または IGNORE CONDITION) のいずれかを使用すると、呼び出し元の設定を変更することになり、これがエラー発生の原因になることがある。
- アセンブラー・ルーチンを呼び出し、現在のハンドルを一時中断してから復元する必要がある場合は、そのアセンブラー・ルーチンで、プッシュ・ハンドルおよびポップ・ハンドルを要求する必要があります。これは、言語環境プログラムでは、COBOL プログラムがアセンブラー・ルーチンを呼び出すときに自動的に行われません。

DL/I CALL インターフェースの使用

CALL DL/I を使用する COBOL プログラムがあり、そのプログラムにまだ以下の変更を加えていない場合は、すぐに変更してください。

- ユーザー・インターフェース・ブロック (DLIUIB) 宣言、および LINKAGE SECTION 内の少なくとも 1 つのプログラム制御ブロック (PCB) 宣言を保存します。
- PCB 呼び出しを次のように変更して、UIB を直接指定する。

```
CALL 'CBLTDLI' USING PCB-CALL
               PSB-NAME
               ADDRESS OF DLIUIB.
```

- UIB のアドレス・リストから必要な PCB のアドレスを入手する。

31 ページの図 1 は上記のプロセス全体を説明しています。図中の例は、PSB に 3 つの PCB を定義してあり、データベース呼び出しでは 2 番目の PCB を使用したいものとします。したがって、LINKAGE SECTION グループの ADDRESS 特殊レジスターである PCB 項目を設定するときは、プログラムは作業用ストレージ・テーブル PCB-ADDRESS-LIST を指すために 2 を使用します。n 番目の PCB を使用するためには、PCB-ADDRESS-LIST を指すために番号 n を使用します。

```
WORKING-STORAGE SECTION.
    77 PCB-CALL              PIC X(4) VALUE 'PCB '.
    77 GET-HOLD-UNIQUE       PIC X(4) VALUE 'GHU '.
    77 PSB-NAME              PIC X(8) VALUE 'CBLPSB'.
    77 SSA1                  PIC X(40) VALUE SPACES.
    01 DLI-IO-AREA.
       02 DLI-IO-AREA1       PIC X(99).
*
LINKAGE SECTION.
    COPY DLIUIB.
    01 OVERLAY-DLIUIB REDEFINES DLIUIB.
       02  PCBADDR           USAGE IS POINTER.
       02  FILLER            PIC XX.
    01 PCB-ADDR-LIST.
       02  PCB-ADDRESS-LIST  USAGE IS POINTER
                             OCCURS 10 TIMES.
    01 PCB.
       02 PCB-DBD-NAME       PIC X(8).
       02 PCB-SEG-LEVEL      PIC XX.
       02 PCB-STATUS-CODE    PIC XX.
*
PROCEDURE DIVISION.
*SCHEDULE THE PSB AND ADDRESS THE UIB
    CALL 'CBLTDLI' USING PCB-CALL PSB-NAME ADDRESS OF DLIUIB.
*
*MOVE VALUE OF UIBPCBAL, ADDRESS OF PCB ADDRESS LIST (HELD IN UIB)
*(REDEFINED AS PCBADDR, A POINTER VARIABLE), TO
*ADDRESS SPECIAL REGISTER OF PCB-ADDR-LIST TO PCBADDR.
    SET ADDRESS OF PCB-ADDR-LIST TO PCBADDR.
*MOVE VALUE OF SECOND ITEM IN PCB-ADDRESS-LIST TO ADDRESS SPECIAL
*REGISTER OF PCB, DEFINED IN LINKAGE SECTION.
    SET ADDRESS OF PCB TO PCB-ADDRESS-LIST(2).
*PERFORM DATABASE CALLS ......
    ........
    MOVE ........ TO SSA1.
    CALL 'CBLTDLI' USING GET-HOLD-UNIQUE PCB DLI-IO-AREA SSA1.
*CHECK SUCCESS OF CALLS .......
    IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN
                                        ...... error diagnostic code
    ........
    IF PCB-STATUS-CODE IS NOT EQUAL SPACES THEN
                                        ...... error diagnostic code
    ........
```

図 1. DL/I CALL インターフェースの使用

VS COBOL II プログラム

言語環境プログラムは、VS COBOL II コンパイラーによりコンパイルされるプログラムの実行をサポートします。このコンパイラーのネイティブ・ランタイム・ライブラリーはサポートされません。ただし、このコンパイラーは、言語環境プログラムに準拠していないため (言語環境プログラム以前のコンパイラーです)、使用に関していくつかの制約事項および考慮事項があります。

言語環境プログラムのサポートへの VS COBOL II プログラムのアップグレードについて詳しくは、「*Enterprise COBOL for z/OS: コンパイラーおよびランタイム移行ガイド*」を参照してください。

言語環境プログラムの呼び出し可能サービス

言語環境プログラムに準拠した COBOL コンパイラーによってコンパイルしたプログラムでは、言語環境プログラム のすべての呼び出し可能サービスを、動的に、あるいは静的に使用できます。CICS アプリケーションに対して CEEMOUT (メッセージのディスパッチ) サービス、および CEE3DMP (ダンプの生成) サービスが異なる場合、その中で、そのメッセージおよびダンプは、MSGFILE ランタイム・オプションで指定される ddname ではなく、CESE 一時データ・キューに送信されます。

VS COBOL II プログラムでは、日時呼び出し可能サービスに対する動的呼び出しを行うことはできますが、VS COBOL II プログラムの場合、言語環境プログラムの呼び出し可能サービスに対するそれ以外の呼び出しは、静的にしろ動的にしろ、サポートされていません。

VS COBOL II プログラムの再リンク

言語環境プログラムにより提供されるランタイム・サポートを使用する既存の VS COBOL II プログラムを再リンクするときに、オブジェクト・モジュールが使用できない場合は、タスクを実行するためのサンプル・ジョブ・ストリームが、SCEESAMP サンプル・ライブラリーの IGZWRLKA メンバーで提供されます。

CICS スタブ

古い CICS スタブである DFHECI とリンクされている COBOL プログラムも言語環境プログラムの下で動作しますが、DFHELII スタブの使用をお勧めします。特に、混合言語環境では、DFHELII スタブの使用が必須です。DFHECI は、アプリケーションのトップでリンク・エディットする必要がありますが、DFHELII はアプリケーションのどこでもリンクできます。

CEEWUCHA の使用

VS COBOL II プログラムが言語環境プログラムにより提供されるランタイム・サポートを使用するように調整している場合は、SCEESAMP ライブラリー内の言語環境プログラムが提供する、サンプル・ユーザー条件処理ルーチン CEEWUCHA を使用すると便利です。これには以下のような機能があります。

- ランタイム検出エラーが発生すると、EXEC CICS HANDLE ABEND LABEL ステートメントが制御権を得るようにして、CICS の下で実行されている既存の VS COBOL II アプリケーションとの互換性を提供する。
- 未処理のランタイム検出エラーのすべてを、VS COBOL II が発行する、それに対応するユーザー 1xxx 異常終了に変換する。
- IGZ0014W メッセージをすべて抑制する。このメッセージは、IGZETUN または IGZEOPT を VS COBOL II アプリケーションとリンク・エディットしたときに生成されます (プログラムが IGZETUN または IGZEOPT とリンク・エディットされていない場合は、パフォーマンスが向上します)。

COBOL での基底付きアドレッシングの使用

COBOL は、ポインター変数および ADDRESS 特殊レジスターを使用して、LINKAGE SECTION で定義されたデータ域へのアドレッシングを可能にする単純な方法を提供します。

CICS アプリケーション・プログラムは、データが CICS 内部域にあり、プログラムにアドレスしか渡されない場合に、データに動的にアクセスする必要があります。例えば、次の通りです。

- ADDRESS コマンドを使用してアクセスされる、CWA、TWA、および TCTTE ユーザー域 (TCTUA) などの CICS 領域。
- SET オプションを指定した READ および RECEIVE などの EXEC CICS コマンドによって得られる入力データ。

ADDRESS 特殊レジスターは、LINKAGE SECTION で定義され、レベル 01 または 77 を持つレコードのアドレスを保持します。このレジスターは、ADDRESS モードのすべてのコマンドの SET オプションで使用することができます。これらのコマンドは、GETMAIN、LOAD、READ、および READQ などです。

図 2 は、COBOL における ADDRESS 特殊レジスターの使用を示しています。READ または REWRITE コマンドのレコードが固定長の場合は、LENGTH オプションは必要ではありません。この例は、可変長レコードを想定しています。読み取りの後で、LENGTH オプションに指定されたフィールド (ここでは LRECL-REC1) からレコードの長さを入手することができます。更新済みレコードを長さの異なるレコードによって置き換えたい場合には、REWRITE コマンドに LENGTH オプションをコーディングする必要があります。

```
WORKING-STORAGE SECTION.
77 LRECL-REC1    PIC S9(4) COMP.
LINKAGE SECTION.
01 REC-1.
   02 FLAG1      PIC  X.
   02 MAIN-DATA  PIC  X(5000).
   02 OPTL-DATA  PIC  X(1000).
01 REC-2.
   02 ...
PROCEDURE DIVISION.
   EXEC CICS READ UPDATE...
       SET(ADDRESS OF REC-1)
       LENGTH(LRECL-REC1)
       END-EXEC.
   IF FLAG1 EQUAL X'Y'
   MOVE OPTL-DATA TO ...

   EXEC CICS REWRITE...
       FROM(REC-1)
       END-EXEC.
```

図 2. 位置指定モードでの CICS データ域のアドレッシング

COBOL プログラムからのサブプログラムの呼び出し

CICS システムでは、制御がアクティブ・プログラムから外部プログラムに移動する際に、移動元のプログラムがアクティブのままで、制御をそのプログラムに戻すことができる場合には、制御の移動先のプログラムをサブプログラムと呼びます。COBOL では、サブプログラムに制御権を移動させるには、以下の 3 つの方法があります。

EXEC CICS LINK
呼び出し側のプログラムに、次のいずれかの形式でコマンドを入れます。

```
EXEC CICS LINK PROGRAM('subpgname')
EXEC CICS LINK PROGRAM(name)
```

最初の形式では、呼び出し先サブプログラムは、英数字リテラルとして指定されます。2 番目の形式では、*name* は、サブプログラムの名前に必要な長さを持つ COBOL データ域を指しています。

静的 COBOL 呼び出し
呼び出し側のプログラムに、次の形式の COBOL ステートメントを入れます。

```
CALL 'subpgname'
```

呼び出すサブプログラムを、リテラル・ストリングとして明示指定します。

動的 COBOL 呼び出し
呼び出し側のプログラムに、次の形式の COBOL ステートメントを入れます。

```
CALL identifier
```

identifier (ID) は、呼び出すサブプログラムの名前が入っている COBOL データ域の名前です。

サブプログラムを呼び出すこれらの各方法を使用した場合のパフォーマンスへの影響について詳しくは、「*Enterprise COBOL for z/OS: プログラミング・ガイド*」および「*IBM Enterprise COBOL Version 3 Release 1 Performance Tuning Paper*」を参照してください。ホワイト・ペーパーは、Web (www.ibm.com/software/ad/cobol/library) より入手可能です。

COBOL プログラムは、CICS がサポートする任意の言語のプログラムを、静的または動的に呼び出すことができます。LINK または XCTL は、COMMAREA などの CICS の機能を使用しない限り、言語間通信では必要ありません。言語間通信について詳しくは、16 ページの『言語環境プログラムにおける言語の混合』を参照してください。

呼び出されるか、またはリンクされるどのサブプログラムの内容も、言語用に CICS がサポートするすべての機能 (例えば、DB2® および DL/I などの外部データベースへの呼び出しを含みます) にすることが可能です。これには、アセンブラー言語サブプログラムは低レベルのサブプログラムを呼び出すことができないという例外があります。

プログラムとサブプログラム間の制御のフロー

COBOL メインプログラムとサブプログラムとの間で可能なフローは多数あります。

実行単位は、COBOL の静的 CALL ステートメントまたは動的 CALL ステートメントによって相互に連絡する 1 つ以上のプログラムの実行セットです。CICS 環境では、実行単位は CICS タスクの開始時に実行開始されるか、あるいは LINK または XCTL コマンドによって呼び出されます。動的 CALL については、呼び出されたプログラムに対して後続の PROGRAM 定義が必要ですが、実行単位は、PROGRAM リソース定義で定義されたプログラムの実行として定義することができます。制御権が XCTL コマンドによって渡された場合、制御権を受け取ったプログ

ラムは、RETURN コマンドまたは GOBACK ステートメントによって呼び出し側プログラムに制御権を返すことができないので、サブプログラムではありません。

各 LINK コマンドは、呼び出し側プログラムのレベルより 1 レベル低くなっている呼び出し先プログラムの新規 **CICS アプリケーション論理レベル**を作成します (CICS はレベル 0 になっていると見なされます)。図 3 に、リンクされて呼び出されたプログラムにおける RETURN コマンド および CALL ステートメントの論理レベルおよび影響を示します。

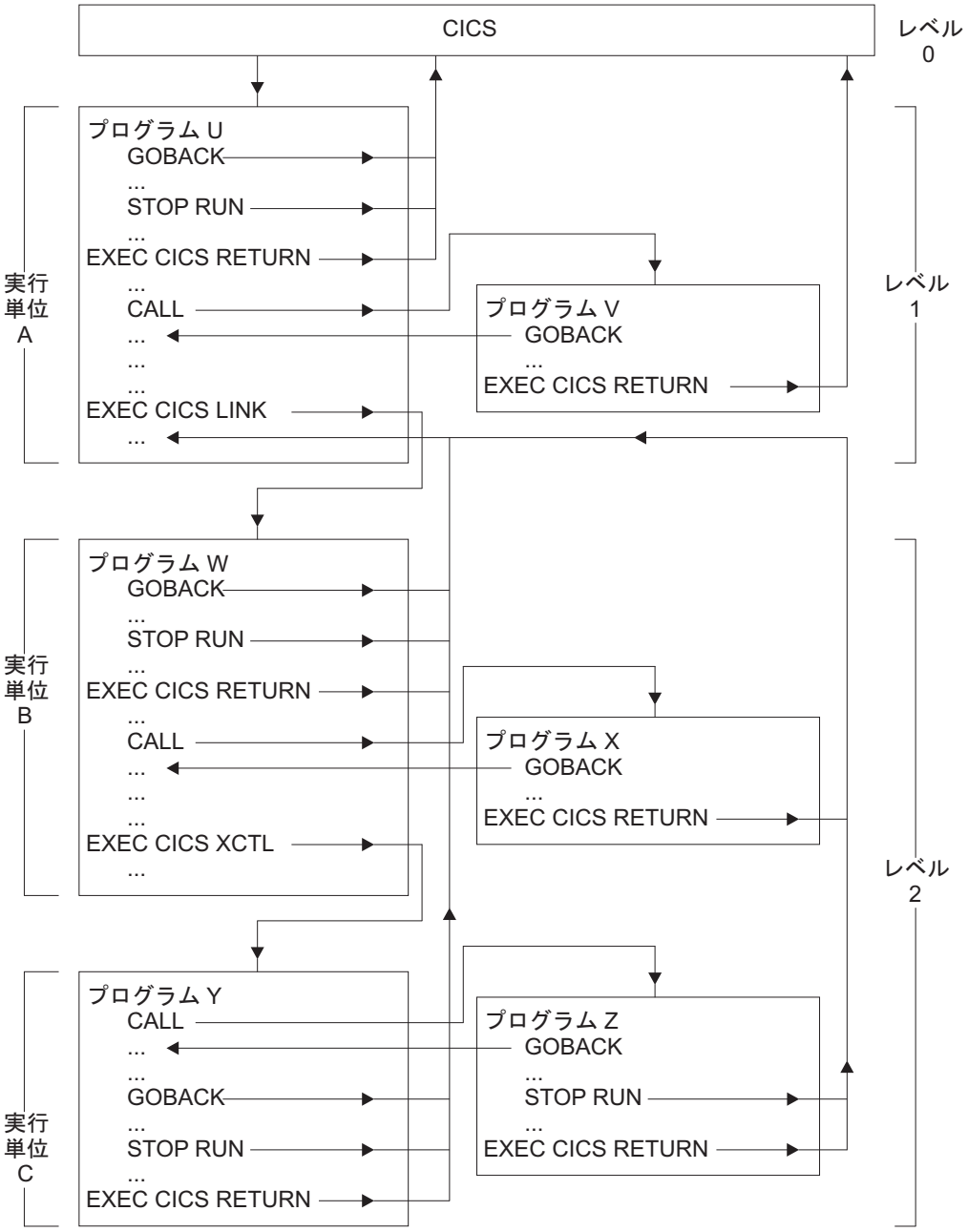


図 3. COBOL プログラム、実行単位、および CICS 間の制御のフロー

メイン、または*レベル 1* のプログラムは、COBOL GOBACK ステートメント、STOP RUN ステートメント、または CICS RETURN コマンドを使用して、終了し

たり、CICS に戻ることができます。メインは、COBOL CALL ステートメントを使用して、同一論理レベル (レベル 1) のサブプログラムを呼び出すか、または CICS LINK コマンドを使用して、論理レベルの低いサブプログラムを呼び出すことができます。レベル 1 の呼び出し先サブプログラムは、COBOL GOBACK ステートメントを使用して、呼び出し側に戻るか、または EXEC CICS RETURN を使用して終了し、CICS に戻ることができます。

レベル 2 で実行中のサブプログラムは、COBOL GOBACK ステートメント、STOP RUN ステートメント、または CICS RETURN コマンドを使用して、終了したり、レベル 1 の呼び出し側プログラムに戻ることができます。このサブプログラムは、COBOL CALL ステートメントまたは CICS XCTL コマンドを使用して、同一レベル (レベル 2) のサブプログラムを呼び出すことができます。レベル 2 の COBOL CALL を使用して呼び出されるサブプログラムは、COBOL GOBACK ステートメントを使用して、呼び出し側プログラム (レベル 2) に戻るか、または EXEC CICS RETURN を使用して、レベル 1 の呼び出し側プログラムに戻ることができます。レベル 2 の XCTL を使用して呼び出されるサブプログラムは、GOBACK、STOP RUN、あるいは EXEC CICS RETURN を使用して、レベル 1 の呼び出し側のプログラムにのみ戻ることができます。

プログラムの論理レベルの詳細については、 326 ページの『アプリケーション・プログラムの論理レベル』を参照してください。

サブプログラムの呼び出し規則

以下の規則は、COBOL プログラムから呼び出されるか、またはリンクされるサブプログラムの要件および振る舞いを説明しています。適用される規則は、制御権をサブプログラムに移す場合に、EXEC CICS LINK コマンド、静的 COBOL 呼び出し、または動的 COBOL 呼び出しのどれを使用するかにより異なります。

サブプログラムの場所

EXEC CICS LINK
サブプログラムはリモートであっても構いません。

静的 COBOL 呼び出しまたは動的 COBOL 呼び出し
サブプログラムはローカルでなければなりません。

変換

統合変換プログラムを備えたコンパイラーを使用する場合は、変換プログラムは必要ありません。

EXEC CICS LINK
リンクされるサブプログラム、またはそれによって呼び出されるサブプログラムに CICS 機能が含まれる場合は、変換しなければなりません。

静的 COBOL 呼び出しまたは動的 COBOL 呼び出し
呼び出し先サブプログラムに、CICS コマンドが含まれているか、EXEC インターフェース・ブロック (DFHEIBLK) または CICS 連絡域 (DFHCOMMAREA) への参照が含まれている場合、呼び出し先サブプログラムは変換しなければなりません。

コンパイル

CICS で実行する COBOL プログラムをコンパイルするとき、そのプログラムが動的呼び出しを発行する場合でも、常に NODYNAM コンパイラー・オプション (デフォルト) を使用する必要があります。

リンク・エディット

EXEC CICS LINK
リンクされるサブプログラムは、独立したプログラムとしてコンパイルしてリンク・エディットしなければなりません。

静的 COBOL 呼び出し
呼び出し先サブプログラムは、単一ロード・モジュールを形成するように呼び出し側のプログラムとリンク・エディットしなければなりません (しかし、プログラムは個別にコンパイルすることができます)。これにより、大きなプログラム・モジュールが生成され、同一プログラムを呼び出す 2 つのプログラムがそのプログラムのコピーを共用しなくなります。

動的 COBOL 呼び出し
呼び出し先サブプログラムは独立したロード・モジュールとして、コンパイルおよびリンク・エディットを行う必要があります。これは、リンク・パック域または、他の CICS 領域および非 CICS 領域で同時に共用されるライブラリーに、常駐することができます。

プログラム自動インストールを使用しない場合の CICS CSD 項目

プログラム自動インストールを使用する場合は、CSD 内に項目は必要ありません。

EXEC CICS LINK
リンクされるサブプログラムは RDO を使用して定義しなければなりません。リンクされるサブプログラムが不明または使用不能の場合には、自動インストールがアクティブでも、LINK は PGMIDERR 条件のために失敗します。

静的 COBOL 呼び出し
呼び出し側のプログラムは CSD に定義しなければなりません。プログラム A がプログラム B を呼び出してから、プログラム B がプログラム A を呼び出そうとした場合には、COBOL はメッセージを発行し、異常終了 (1015) します。サブプログラムは呼び出し側のプログラムの一部なので、CSD 項目は不要です。

動的 COBOL 呼び出し
呼び出し側のプログラムは CSD に定義しなければなりません。プログラム A がプログラム B を呼び出してから、プログラム B がプログラム A を呼び出そうとした場合には、COBOL はメッセージを発行し、異常終了 (1015) します。呼び出し先サブプログラムは CSD に定義しなければなりません。自動インストールがアクティブでも、呼び出し先サブプログラムがロードできないか、または使用不能の場合は、COBOL はメッセージを発行して異常終了 (1029) します。

サブプログラムへのパラメーターの引き渡し

呼び出されるかまたはリンクされるサブプログラムが CICS 変換プログラムによって処理されていれば、CICS のいずれかの標準方式
(COMMAREA、TWA、TCTUA、TS キュー) によって、データを渡すことができます。

EXEC CICS LINK
COMMAREA を使用する場合には、そのアドレスを LINK コマンドで渡さなければなりません。リンクされるサブプログラムが 24 ビット・アドレッシングを使用しており、COMMAREA が 16MB 境界より上のアドレスにある場合には、CICS はそれを 16MB 境界より下のアドレスにコピーし、戻るときにそれを再度コピーします。

静的 COBOL 呼び出し
呼び出し先プログラムが EXEC CICS 要求を発行する場合、または EXEC CICS ADDRESS コマンドを発行できる場合は、CALL ステートメントは、最初の 2 つのパラメーターとして DFHEIBLK および DFHCOMMAREA を渡すことができます。COMMAREA はオプションですが、これ以外のパラメーターが渡される場合は、ダミーの COMMAREA も渡す必要があります。ネストされたプログラムの場合、規則は異なります。

動的 COBOL 呼び出し
呼び出し先プログラムが EXEC CICS 要求を発行する場合、または EXEC CICS ADDRESS コマンドを発行できる場合は、CALL ステートメントは、最初の 2 つのパラメーターとして DFHEIBLK および DFHCOMMAREA を渡すことができます。COMMAREA はオプションですが、これ以外のパラメーターが渡される場合は、ダミーの COMMAREA も渡す必要があります。呼び出し先サブプログラムが 24 ビット・アドレッシングを使用しており、パラメーターのどれかが 16MB 境界より上のアドレスにある場合には、COBOL はメッセージを発行し、異常終了 (1033) します。

サブプログラムからの戻り

EXEC CICS LINK
リンクされたサブプログラムは、RETURN またはネイティブ言語戻りコマンド (COBOL ステートメント GOBACK など) のいずれかを使用して戻らなければなりません。

静的 COBOL 呼び出しまたは動的 COBOL 呼び出し
呼び出されたサブプログラムは、ネイティブ言語戻りステートメント (COBOL ステートメント GOBACK、または EXIT PROGRAM など) を使用して戻らなければなりません。呼び出されたサブプログラムで RETURN を使用すると、呼び出し側のプログラムが終了します。

ストレージ

EXEC CICS LINK
リンクされるサブプログラムに入るたびに、新しく初期設定されたその WORKING-STORAGE SECTION のコピーが提供され、実行単位が再初期設定されます (環境によっては、パフォーマンスの低下を引き起こすことがあります)。

静的 COBOL 呼び出しまたは動的 COBOL 呼び出し

CICS 論理レベル内の呼び出し先サブプログラムへの最初の入り口で、その WORKING-STORAGE SECTION の初期設定済みの新しいコピーが提供されます。同一論理レベルの呼び出し先サブプログラムへの後続の入り口で、同一の WORKING STORAGE が最後に使用された状態で提供されます。すなわち、ストレージの解放、獲得、または初期設定は行われません。LINK コマンドを使用したパフォーマンスが十分でない場合には、COBOL 呼び出しを使用するとパフォーマンスが向上する可能性があります。

CICS 条件、AID および異常終了処理

EXEC CICS LINK

呼び出し先サブプログラムへの入り口では、異常終了または条件処理はアクティブになっていません。サブプログラム内では、通常の CICS 規則が適用されます。サブプログラムの実行時に存在する異常終了または条件処理環境を設定するためには、サブプログラムへの入り口で新規 HANDLE コマンドを発行する必要があります。そのように作成される環境は、後から HANDLE コマンドが発行されるか、あるいはサブプログラムが制御を呼び出し側に戻すまで有効になったままです。

静的 COBOL 呼び出しまたは動的 COBOL 呼び出し

動的 COBOL CALL が失敗した場合には、CICS 異常終了処理は呼び出されず、COBOL 異常終了コード (1013) を受け取ることがあります。

- 言語環境プログラム および CBLPSHPOP ON を使用した動的 COBOL 呼び出しが失敗した場合、呼び出し先サブプログラムへの入り口では、異常終了または条件処理はアクティブになっていません。サブプログラム内では、通常の CICS 規則が適用されます。呼び出し先サブプログラムへの入り口では、呼び出し側のプログラムの条件または異常終了ハンドラーをスタックするために、COBOL は PUSH HANDLE を発行します。サブプログラムの実行時に存在する異常終了または条件処理環境を設定するためには、サブプログラムへの入り口で新規 HANDLE コマンドを発行する必要があります。これによって作成される環境は、後から HANDLE コマンドが発行されるか、あるいはサブプログラムが制御を呼び出し側に戻すまで有効になったままです。サブプログラムから呼び出し側のプログラムに制御が戻ると、COBOL は POP HANDLE を使用して、その条件と異常終了ハンドラーのスタックを解除します。
- CBLPSHPOP OFF を使用した動的 COBOL 呼び出しが失敗した場合、呼び出し側プログラムのための条件、AID、および異常終了の処理は有効なままになります。

COBOL2 および COBOL3 変換プログラム・オプション

CICS Transaction Server for z/OS バージョン 4 リリース 1 では、COBOL プログラムのために COBOL2 と COBOL3 の CICS 変換プログラム・オプションの間から選択できます。

ANSI85 変換プログラム・オプションは、CICS Transaction Server for z/OS バージョン 2 リリース 2 で使用できなくなりました。

以前の CICS リリースで OOCOBOL 変換プログラム・オプションを使用してコンパイルされたモジュールは、CICS Transaction Server for z/OS バージョン 4 リリース 1 では実行できません。OOCOBOL 変換プログラム・オプションは、以前の SOM ベース (システム・オブジェクト・マネージャー・ベース) の OO COBOL に対して使用されたもので、OO COBOL のこの形式のランタイム・サポートは、z/OS V1.2 で廃止されました。新しい Java ベースの OO COBOL は、Enterprise COBOL で使用されますが、CICS 変換プログラムではサポートされません。

COBOL2 オプションがデフォルトです。変換プログラムについては、CICS Transaction Server for z/OS バージョン 2 リリース 1 および以前のリリースと同じ効果はありません。COBOL2 は変換プログラムに COBOL3 として変換するよう指示しますが、追加で、EXEC CICS および EXEC DLI 要求で使用する一時変数の宣言も含めるよう指示します。

一時変数の使用を必要とする方法で作成された以前のプログラムを再変換する場合、COBOL2 オプションを選択します。特に、一時変数の使用は、プログラムの引数値が不適切に定義された場合に通常発生するエラーを回避する場合があることに注意してください。CICS Transaction Server for z/OS バージョン 2 リリース 1 および以前のリリースの COBOL2 オプションは、一時変数の宣言を提供していました。この機能のため、CICS Transaction Server の以前のリリースの COBOL2 オプションを使用して元々変換されたプログラムで、引数値の不適切な定義が存在するが、実行時には認識できない場合があります。COBOL3 オプションを使用してこれらのプログラムを変換すると、これらのエラーが初めて明らかになる場合があります。CICS の新しいリリースへのアップグレードを支援するために、新しい COBOL2 オプションを使用して、プログラムのエラーを修正する代わりに、回避し続けることができます。

プログラムで変換プログラムの一時変数が必要ないことがわかっている場合は、COBOL3 を使用して、作業用ストレージを節約できます COBOL3 オプションには、一時変数の宣言を除いて、以前の COBOL2 および ANSI85 変換プログラム・オプションのすべての機能が組み込まれています。

注: COBOL2 と COBOL3 を同時に指定することはできません。異なるメソッドで両方のオプションを指定する場合は、2 つのオプションが指定された場所に関係なく、COBOL3 オプションが *常に* 使用されます。この場合、変換プログラムが警告メッセージを出します。

CICS Transaction Server for z/OS バージョン 2 リリース 2 およびそれ以降のバージョンとリリースの CICS 変換プログラムのサポートは、以前の COBOL コンパイラーで使用可能であった CMPR2 コンパイラー・オプションの使用をサポートしません。これらの COBOL プログラムを NOCMPR2 機能にアップグレードするための情報については、「*Enterprise COBOL for z/OS: コンパイラーおよびランタイム移行ガイド*」を参照してください。

プログラムの変換とその実行準備について詳しくは、 81 ページの『第 7 章 変換およびコンパイル』を参照してください。

COBOL プログラムの CICS 変換プログラム・アクション

以下の注で、COBOL3 オプションが使用される場合に実行される特定の変換プログラムのアクションについて説明します。COBOL2 オプションを使用した処理は、一時変数の宣言を除き、すべての面で同一です。

リテラル内のブランク行

ブランク行は COBOL ソース・プログラムのどこに現れてもかまいません。ブランク行は、列 7 から 72 までのスペースを包括的に含み、それ以外は含みません。

COBOL ソース・プログラムのリテラル内にブランク行がある場合には、変換プログラムは変換出力からそれらを除去しますが、変換リストには含めます。

小文字

小文字は、ユーザー定義の名前、システム名、予約名など、COBOL ワードのどこにあってもかまいません。変換プログラムのリストおよび出力は、入力されたままの形の COBOL テキストを保持しています。

さらに、変換プログラムは、次において大文字と小文字の混合を受け入れます。

- 変換プログラム・オプション
- EXEC CICS コマンドのキーワードおよびキーワードの引数の両方に対して
- CBL および PROCESS ステートメント
- EJECT および SKIP1 のようなコンパイラー・ディレクティブ

変換プログラムは、小文字を大文字に変換しません。COBOL テキスト中の一部の名前、例えば、ファイル名およびトランザクション ID などは外部定義された名前と一致しなければなりません。このような名前は、常に、大文字小文字も含め外部定義されている通りに入力しなければなりません。

ユーザーが LINKAGE 変換オプションを指定、またはデフォルトを許可する場合には、EIB 構造体 (DFHEIBLC) の大小混合バージョンが、LINKAGE SECTION に挿入されます。

任意の文字を含むシーケンス番号

COBOL ソース・プログラムで、シーケンス番号フィールドには、コンピューターの文字セットの中の任意の文字を入れることができます。シーケンス番号フィールドはどのような順序になっていてもかまわず、固有にする必要はありません。

REPLACE ステートメント

COBOL プログラムには、識別されたテキストを定義済みの置換テキストによって置き換えることができる REPLACE ステートメントを含めることができます。置き換えるテキストおよび挿入するテキストは、疑似テキスト、ID、リテラル、または COBOL ワードとすることができます。REPLACE ステートメントは COPY ステートメントの後で処理されます。

COBOL ソース・ステートメントを CICS 提供変換プログラムを使用して処理する場合、変換プログラムは REPLACE ステートメントを受け入れますが、疑似テキス

ト区切り文字の間のテキストは変換しません。ただし例外があり、CICS 組み込み関数 (DFHRESP および DFHVALUE) は、どこにあっても変換されます。疑似テキスト区切り文字の間に CICS コマンドを入れてはいけません。

統合変換プログラムを使用する場合、この変換プログラムは REPLACE ステートメントを受け入れて、疑似テキスト区切り文字の間のテキストを変換します。疑似テキスト区切り文字の間に CICS コマンドを入れることができます。

参照修正

参照変更がサポートされます。これは、文字データ項目のサブストリングを参照使用する方式で、データ項目中のサブストリングの開始 (左端) 位置、およびオプションでサブストリングの長さを指定します。受け入れ可能な形式は次の通りです。

```
data-name (leftmost-character-position:)
data-name (leftmost-character-position: length)
```

データ名 は添え字付きデータ名または修飾データ名、あるいはその両方とすることができます。*開始位置* と *長さ* の両方は算術式とすることができます。参照変更、修飾、および添え字について詳しくは、「*Enterprise COBOL for z/OS: 言語解説書*」を参照してください。

変換プログラムは、COBOL プログラムまたは EXEC CICS コマンドの中で、文字変数の名前が使えるところであればどこでも、参照修正を受け入れます。

注: CICS コマンドが参照修正を使用してデータ値を定義する場合は、NOLENGTH 変換プログラム・オプションが使用されない限り、LENGTH オプションを組み込んでデータ長を指定する必要があります。そうしないと、変換プログラムは次の形式の LENGTH レジスター参照を使用する COBOL 呼び出しを生成します。

```
LENGTH OF (reference modification)
```

これは、コンパイラーによって拒否されます。

グローバル変数

GLOBAL 変数ストレージ・クラスがサポートされます。最上位レベルのプログラム (45 ページの『ネストされた COBOL プログラム』を参照) 内で、GLOBAL 変数ストレージ・クラスを使用して定義された変数は、直接または間接に含まれているネストされたプログラムの中で参照できます。

変換プログラムは GLOBAL キーワードを受け入れます。

区切り文字としてのコンマおよびセミコロン

分離文字としてのコンマは、後ろにスペースが続いているコンマのことです。また、分離文字としてのセミコロンは、後ろにスペースが続いているセミコロンのことです。分離文字コンマまたは分離文字セミコロンは、スペースを単独で使用できる場所ならどこでも、分離文字として使用することができます。

変換プログラムは、COBOL ステートメントでスペースを使用できる場所であれば、どこで分離文字コンマまたは分離文字セミコロンを使用しても受け入れます。例えば、変換プログラムは次のステートメントを受け入れます。

```
IDENTIFICATION; DIVISION
```

変換プログラムは、EXEC CICS コマンドでの区切り文字としての、分離文字コンマ、および分離文字セミコロンの使用は受け入れません。EXEC CICS コマンドの場合に受け入れられる区切り文字は、スペースだけです。

シンボリック文字の定義

記号文字は、ALPHABET 文節の後の SPECIAL-NAMES 段落で定義することができます。記号文字は、1 文字の表意定数を表すプログラム定義の語です。

変換プログラムは、規格で指定されている通りに記号文字の使用を受け入れます。

注: 一般に、コンパイラーは、CALL ステートメントの引数としては、表意定数および記号文字の使用を受け入れません。このため、変換プログラムが CALL ステートメントに変換する EXEC CICS コマンドで、表意定数または記号定数を使用しないでください。この制約事項には例外が 1 つあります。すなわち、表意定数のデータ・タイプが正しい場合には、表意定数は EXEC CICS コマンドで、値を**渡す** 引数として受け入れ可能です。例えば、数値表意定数は LENGTH オプションで使用することができます。

COBOL プログラムのバッチ・コンパイル

別々の COBOL プログラムを 1 つの入力ファイルとして一緒にコンパイルすることができます。END PROGRAM ヘッダー・ステートメントは各プログラムを終了させます。これは、バッチの最後のプログラムの場合はオプションです。変換プログラムは、別々の COBOL プログラムを 1 つの入力ファイルとして受け入れ、END PROGRAM ヘッダー・ステートメントを解釈します。

変換プログラムの呼び出し時にパラメーターとして指定した変換プログラム・オプションは、バッチ全体に対して適用されますが、コンパイル単位を開始する CBL カードまたは PROCESS カードにオプションを指定すれば、コンパイル単位ごとにオプションを変更することができます。

コンパイル単位のオプションは、次の優先順位に従って決まります。

1. インストール・ユーザー変更不可能オプションとして固定されたオプション
2. コンパイル単位を開始する CBL カードまたは PROCESS カードに指定したオプション
3. 変換プログラムの呼び出し時に指定したオプション
4. デフォルト・オプション

コンパイルの詳細については、 119 ページの『第 8 章 アプリケーション・プログラムのインストール』を参照してください。

バッチ・コンパイルを使用している場合は、コンパイルおよびリンク・エディットが成功するように、次の追加処置をとらなければなりません。

- コンパイラーを呼び出す JCL ステートメント、または最高レベル (ネストされていない) の各プログラムに対する CBL ステートメントに、コンパイラー NAME オプションをパラメーターとして組み込む。こうすると、各プログラムの最後に NAME ステートメントが組み込まれます。詳しくは、 44 ページの図 4を参照してください。

- 各オブジェクト・モジュールに対する CICS COBOL スタブに INCLUDE および ORDER ステートメントを追加するために、コンパイラー出力を編集する。これらのステートメントによって、リンケージ・エディターが各ロード・モジュールの開始時にスタブを組み込みます。これらのステートメントは、モジュールのどこに置いてもかまいませんが、規則では、先頭にくることになっています。これらをモジュールの末尾、つまり各 NAME ステートメントの直前に置くと便利です。 45 ページの図 5 に、このように編集した後の 図 4 からの出力を示します。

バッチ・コンパイルについては、 119 ページの『第 8 章 アプリケーション・プログラムのインストール』に記述されているプロシージャーを変更する必要があります。推奨する方法は次のとおりです。

1. 提供されたカタログ式プロシージャー DFHYITVL を 2 つのプロシージャーに分割する。つまり、変換およびコンパイル・ステップ (TRN および COB) を含む PROC1、および リンケージ・エディター・ステップ COPYLINK と LKED を含む PROC2 に分割します。
2. PROC1 で、コンパイラーに対する EXEC ステートメントのパラメーターに NAME オプションを追加する。そうすると、次のようになります。

   ```
   //COB    EXEC PGM=IGYCRCTL,REGION=..,
   //       PARM='....,NAME,....',
   ```

3. PROC1 で、コンパイラー出力データ・セット &&LOADSET の名前および後処理を変更する。すくなくとも、最初の && をデータ・セット名から除去し、後処理を CATLG に変更してください。そうすると、SYSLIN ステートメントは次のようになります。

   ```
   //SYSLIN DD DSN=LOADSET,DISP=(NEW,CATLG),
   //          UNIT=&WORK,SPACE=(80,(250,100))
   ```

4. PROC1 を実行する。

```
          ..................
          ....program a....
          ..................
NAME PROGA(R)
          ..................
          ..................
          ....program b....
          ..................
          ..................
NAME PROGB(R)
          ..................
          ....program c....
          ..................
NAME PROGC(R)
```

図 4. 編集前のコンパイラー出力

5. 45 ページの図 5 に示すように、INCLUDE および ORDER ステートメントを追加するために、データ・セット LOADSET 内のコンパイラー出力を編集する。バッチの中で多数のプログラムを使用する場合、ORDER および INCLUDE ステートメントを挿入するには、簡単なプログラムまたは REXX EXEC を書くことをお勧めします。
6. PROC2 では、CICS スタブを含むライブラリーに DD ステートメントを追加する。このライブラリーの標準名は、CICSTS41.CICS.SDFHLOAD です。スタブに対する INCLUDE ステートメントは DD 名によってこのライブラリーを参照し

ます。図 5 では、DD 名 SYSLIB (または SYSLIB に連結されたこのライブラリー) を使用されていることを前提にしています。推奨するステートメントは、次の通りです。

```
//SYSLIB DD DSN=CICSTS41.CICS.SDFHLOAD,
//       DISP=SHR
```

7. PROC2 では、SYSLIN 連結を単一ステートメントで置き換える。

```
//SYSLIN DD DSN=LOADSET,
//       DISP=(OLD,DELETE)
```

このステートメントでは、コンパイラー出力データ・セット LOADSET が名前変更されていることを前提としています。

8. PROC2 を実行する。

```
          ....program a....
          .................
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGA(R)
          .................
          .................
          ....program b....
          .................
          .................
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGB(R)
          .................
          ....program c....
          .................
INCLUDE SYSLIB(DFHELII)
ORDER DFHELII
NAME PROGC(R)
```

図 5. リンケージ・エディターの入力

注: DFHELII スタブの使用を推奨しますが、DFHECI は今でも提供されており、使用可能です。

ネストされた COBOL プログラム

COBOL プログラムに COBOL プログラムを含めることができます。含まれる側のプログラムは、含む側のプログラムの END PROGRAM ステートメントの直前に組み込まれます。含まれる側のプログラムを、含む側のプログラムにしてもかまいません。すなわち、含まれるプログラム自体を他のプログラムに含めることができます。プログラムは、含む側も含まれる側もそれぞれ、END PROGRAM ステートメントで終わります。

ネストされたプログラムの有効な呼び出し、および COMMON 属性について詳しくは、「*Enterprise COBOL for z/OS: カスタマイズ・ガイド*」を参照してください。

CICS 変換プログラムでは、最高レベルのプログラムとネストされたプログラムとの扱い方に、違いがあります。

1 つの点を除き変換プログラムは最高レベルのプログラム (他のどのプログラムにも含まれていないプログラム) を通常の方法で変換します。変換プログラムでは、WORKING-STORAGE SECTION にある変換プログラム生成の変数すべてに対し、GLOBAL 属性を割り当てます。

変換プログラムは、ネストされたプログラムまたは含まれる側のプログラムを、次のように特殊な方法で変換します。

- DATA DIVISION および LINKAGE SECTION が存在していなければ、それらを追加する。
- DFHEIBLK (EXEC インターフェース・ブロック) および DFHCOMMAREA (連絡域) の宣言を、LINKAGE SECTION に挿入する。
- EXEC CICS コマンドおよび CICS 組み込み関数を変換する。
- PROCEDURE DIVISION ヘッダーは修正しない。
- 呼び出し前の割り当てに使用される、変換プログラム生成の一時変数は、WORKING-STORAGE SECTION には挿入しない。

変換プログラムは、コメント以外の最初のレコードが次のいずれかである場合、入力ソースが最高レベルのプログラムで始まるものと解釈します。

- IDENTIFICATION DIVISION ステートメント
- CBL カード
- PROCESS カード

最初のレコードがこれらのいずれでもない場合には、変換プログラムは入力を、ネストされたプログラムの PROCEDURE DIVISION の一部として取り扱います。最初の CBL カードまたは PROCESS カードは、最高レベルのプログラムの初め、および新規コンパイル単位の初めを指示します。最初の最高レベルのプログラムの前に見付かるすべての IDENTIFICATION DIVISION ステートメントは新規のネストされたプログラムの始めを指示します。

これらの規則の実際の効果は、ネストされたプログラムを個別のファイルに保持して、個別に変換することができないという点です。最上位のプログラム、およびそれが直接または間接に含んでいるすべてのプログラムが、単一のコンパイル単位を構成し、それらを一緒に変換プログラムに実行依頼する必要があります。

ネストされたプログラムのコメント

変換プログラムは、END PROGRAM ステートメントに続くコメントを、入力ソースの次 のプログラムに属しているものとして取り扱います。IDENTIFICATION DIVISION ステートメントの前のコメントは、リスト中では IDENTIFICATION DIVISION ステートメントの後に現れます。

混乱を避けるために、コメントは必ず次のいずれかの場所に入れてください。

- コメントが参照しているプログラムを開始する IDENTIFICATION DIVISION ステートメントの後
- コメントが参照しているプログラムを終了させる END PROGRAM ステートメントの前

別の変換プログラムを使用している場合

EXEC CICS コマンドを含むネストされたプログラムに対して、別の変換プログラムを使用しており、組み込まれた CICS 変換プログラムを使用していない場合は、このセクションで説明するように、CALL の USING 句、および PROCEDURE DIVISION で EIB および COMMAREA を明示的にコーディングする必要があります。

組み込まれた CICS 変換プログラムを使用している場合、EXEC CICS コマンドを含むネストされたプログラムに対しては、上記の処置を行う必要は**ありません**。CICS 変換プログラムが有効なコンパイラーは、DFHEIBLK および DFHCOMMAREA を最高レベルのプログラムでグローバルとして宣言します。つまり、明示的なコーディングは必要ありません。

別の変換プログラムを使用している場合:

1. EXEC CICS コマンド、CICS 組み込み関数、または EIB か COMMAREA への参照を含むネストされた各プログラムでは、以下のように、PROCEDURE DIVISION ヘッダーの最初の 2 つのパラメーターとして、DFHEIBLK および DFHCOMMAREA をコーディングする。

   ```
   PROCEDURE DIVISION USING DFHEIBLK
           DFHCOMMAREA PARM1 PARM2 ...
   ```

2. EXEC CICS コマンド、CICS 組み込み関数、または EIB か COMMAREA への参照を含むネストされたプログラムの呼び出しでは、CALL ステートメントの最初の 2 つのパラメーターとして、次のように DFHEIBLK および DFHCOMMAREA をコーディングする。

   ```
   CALL 'PROGA' USING DFHEIBLK
           DFHCOMMAREA PARM1 PARM2 ...
   ```

3. 最上位のプログラムと、EXEC CICS コマンド、CICS 組み込み関数、あるいは EIB または COMMAREA への参照を含むネストされたプログラムとの間で、制御階層を形成するすべての呼び出しでは、CALL ステートメントの最初の 2 つのパラメーターとして、DFHEIBLK と DFHCOMMAREA をコーディングする。呼び出し先プログラムの PROCEDURE DIVISION にも、DFHEIBLK と DFHCOMMAREA をコーディングする。これが必要なのは、EIB および COMMAREA へのアドレッシングを可能にして、最高レベル・プログラムに直接は含まれていないプログラムに渡せるようにするためです。
4. 上記のいずれの理由に照らしても、ネストされたプログラムの PROCEDURE DIVISION に、DFHEIBLK および DFHCOMMAREA を挿入する必要がない場合は、プログラムを呼び出す CALL ステートメントのパラメーター・リストには、DFHEIBLK および COMMAREA を含めない。

プログラムのネストの例

コンパイル単位は、最上位レベルのプログラム W およびネストされた 3 つのプログラム X、Y、Z (すべて W に直接含まれている) から構成されています。

プログラム W

初期設定および終了時に、Y および Z を呼び出して、初期 CICS 処理および非 CICS ファイル・アクセスを実行します。X を呼び出してメイン処理を実行します。

プログラム X
Z を呼び出して非 CICS ファイル・アクセス、および Y を呼び出して CICS 処理を実行します。

プログラム Y
CICS コマンドを発行します。非 CICS ファイル・アクセスのために Z を呼び出します。

プログラム Z
バッチ・モードでファイルにアクセスします。

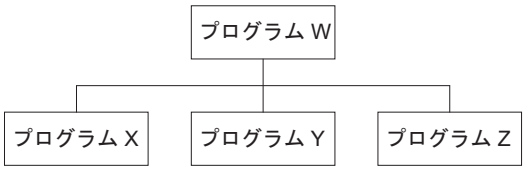


図 6. ネストされたプログラムの例 - ネスト構造

規則を適用すると、次のようになります。

- Y を COMMON にして、X からの呼び出しを可能にする必要がある。
- Z を COMMON にして、X および Y からの呼び出しを可能にする必要がある。
- Y は CICS コマンドを実行するので、別の変換プログラムを使用している場合:
  - Y を呼び出す際は必ず、最初の 2 つのパラメーターとして、DFHEIBLK および COMMAREA を指定する。
  - Y の PROCEDURE DIVISION ヘッダーには、最初の 2 つのパラメーターとして DFHEIBLK および DFHCOMMAREA を指定する必要がある。
- X は EIB または連絡域にアクセスしないが、CICS コマンドを発行する Y を呼び出す。したがって、別の変換プログラムを使用している場合、X の呼び出しでは、最初の 2 つのパラメーターとして DFHEIBLK および COMMAREA を指定しなければならず、X の PROCEDURE DIVISION ヘッダーでは、最初の 2 つのパラメーターとして DFHEIBLK および DFHCOMMAREA を指定しなければなりません。

49 ページの図 7 に、これらの要点を示します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. W.
.
.
PROCEDURE DIVISION.
.
.
    CALL Z.
.
.
    CALL Y USING DFHEIBLK COMMAREA.
.
.
    CALL X USING DFHEIBLK COMMAREA.
.
.
  IDENTIFICATION DIVISION.
  PROGRAM-ID. X.
.
.
  PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA
.
.
    CALL Z.
.
.
    CALL Y USING DFHEIBLK COMMAREA.
.
.
  END PROGRAM X.
  IDENTIFICATION DIVISION.
  PROGRAM-ID. Y IS COMMON.
.
.
  PROCEDURE DIVISION USING DFHEIBLK DFHCOMMAREA.
.
.
    CALL Z.
.
.
    EXEC CICS...
.
.
  END PROGRAM Y.
  IDENTIFICATION DIVISION.
  PROGRAM-ID. Z IS COMMON.
.
.
  PROCEDURE DIVISION.
.
.
  END PROGRAM Z.
END PROGRAM W.
```

図 7. ネストされたプログラムの例 - コーディング

OS/VS COBOL プログラムのアップグレード

OS/VS COBOL プログラムのランタイム・サポートは、現在は廃止されています。これらのプログラムは、CICS Transaction Server for z/OS バージョン 2 ではランタイム・サポートがありましたが、CICS Transaction Server for z/OS バージョン 3 では実行できません。このセクションでは、OS/VS COBOL プログラムをサポートされるレベルの COBOL にアップグレードする方法について説明します。

OS/VS COBOL プログラムは、言語環境プログラム に準拠した COBOL にアップグレードし、CICS がサポートする COBOL コンパイラーのレベルに再コンパイルする必要があります。「*CICS Transaction Server for z/OS リリース・ガイド*」のでは、CICS Transaction Server for z/OS バージョン 4 リリース 1 のサポートする COBOL コンパイラーがリストされています。Enterprise COBOL for z/OS が、推奨コンパイラーです。

CICS-COBOL インターフェースにおける変更の多くは、Enterprise COBOL が手順を単純化しているために生じます。これは、ユーザーがいくつかの CICS 特定 OS/VS COBOL プログラミング手法を使用する必要がないということを意味します。

このセクションで説明する変更の 1 つで、必須である唯一のものは、BLL セルに対するすべての PROCEDURE DIVISION 参照の置換 (除去) です。

人工的割り当て

OCCURS DEPENDING ON オブジェクトから人工的割り当てを除去してください。これらは、OS/VS COBOL では、アドレス可能度を確保するために必要です。

基底付きアドレッシング

BLL セルを定義および操作しないでください。CICS SET オプションおよび BLL セルを使用するプログラムを見直して、以下の変更を行います。

- BLL セルと FILLER フィールドを定義している全構造を、リンケージ・セクションから除去する。詳しくは、51 ページの表 1を参照してください。
- チェーン・ストレージ域を取り扱っているコードを修正して、ADDRESS 特殊レジスターおよび POINTER 変数を利用する。
- SET(ADDRESS OF A-DATA) または SET(A-POINTER) に対する CICS コマンドのすべての SET (BLL セル) オプションを変更する。ここで、A-DATA は、リンケージ・セクション内の構造体あり、A-POINTER は、USAGE IS POINTER 文節を使用して定義されてます。
- すべての SERVICE RELOAD ステートメントを除去する。
- OS/VS COBOL で、4KB より長いリンケージ・セクション内で構造体をアドレッシングするために必要なすべてのプログラム・ステートメントを除去する。典型的なステートメントは以下のとおりです。

  ```
  ADD 4096, D-PTR1 GIVING D-PTR2.
  ```

- BLL セルを使用してチェーン・ストレージ域にアドレッシングする人工的パラグラフ名を除去する。
- 自身のリンケージ・セクションで BMS マップ・データ構造を使用するプログラムを検討する。考慮すべき点は以下のとおりです。

– OS/VS COBOL プログラムでは、作業用ストレージは、コンパイルされ、保管されたプログラムの一部です。linkage section にマップを置くことは、保管されたプログラム、および保管ライブラリー・スペースのサイズを削減します。Enterprise COBOL では、作業用ストレージは、コンパイルされたプログラムの一部ではなく、動的に取得されます。
– ユーザーのマップがリンケージ・セクションにある場合、CICS GETMAIN コマンドと FREEMAIN コマンドを使用して、マップ・ストレージを動的に取得したり、解放したりできます。これは、ストレージ使用の最適化に役立ち、長い会話型トランザクションに有用です。この linkage section マップの利点は、Enterprise COBOL でも適用できます。
– ユーザーのマップがリンケージ・セクションにある場合、マップに対するストレージを獲得するには、CICS GETMAIN コマンドを発行する必要があります。OS/VS COBOL を使用して、ストレージの必要な量を決定しなければなりません。これは、ユーザーのマップ・セットの最大のマップに対して十分でなければなりません。これを決定することは、なかなか困難なことであり、おそらく、すべてのマップのアセンブリーを調べなければなりません。Enterprise COBOL では、LENGTH 特殊レジスターを使用してください。

```
EXEC CICS GETMAIN
     SET(ADDRESS OF DATAREA)
     LENGTH(LENGTH OF DATAREA)
```

表 1. 位置指定モードでの CICS データ域のアドレッシング

OS/VS COBOL	Enterprise COBOL

WORKING-STORAGE SECTION.
 77 LRECL-REC1 PIC S9(4) COMP.
LINKAGE SECTION.
 01 BLLCELLS.
  02 FILLER PIC S9(8) COMP.
  02 BLL-REC1A PIC S9(8) COMP.
  02 BLL-REC1B PIC S9(8) COMP.
  02 BLL-REC2 PIC S9(8) COMP.
 01 REC-1.
  02 FLAG1 PIC X.
  02 MAIN-DATA PIC X(5000).
  02 OPTL-DATA PIC X(1000).
 01 REC-2.
  02 ....
  .
  .
 PROCEDURE DIVISION.
  EXEC CICS READ UPDATE.
  .
  .
  SET(BLL-REC1A)
  LENGTH(LRECL-REC1)
  END-EXEC.
  ADD 4096 BLL-REC1A GIVING BLL-REC1B.
  SERVICE RELOAD REC-1.
  IF FLAG1 EQUAL X'Y'
  MOVE OPTL-DATA TO ...
  EXEC CICS REWRITE...
  FROM(REC-1)
  LENGTH(LRECL-REC1)
  END-EXEC.
	WORKING-STORAGE SECTION.
	 77 LRECL-REC1 PIC S9(4) COMP.
	LINKAGE SECTION.
	01 REC-1.
	  02 FLAG1 PIC X.
	  02 MAIN-DATA PIC X(5000).
	  02 OPTL-DATA PIC X(1000).
	01 REC-2.
	  02 ...
	  .
	  .
	PROCEDURE DIVISION.
	EXEC CICS READ UPDATE
	  .
	  .
	  SET(ADDRESS OF REC-1)
	  LENGTH(LRECL-REC1)
	  END-EXEC.
	IF FLAG1 EQUAL X'Y'
	  MOVE OPTL-DATA TO ...
	  EXEC CICS REWRITE
	  .
	  .
	  FROM(REC-1)
	  END-EXEC.

この表は、Enterprise COBOL で ADDRESS 特殊レジスターを使用することにより、OS/VS COBOL の BLL セルと SERVICE RELOAD を置換することを示しています。READ または REWRITE コマンドのレコードが固定長の場合、Enterprise COBOL は、LENGTH オプションを必要としません。この例は、可変長レコードを想定しています。読み取りの後で、LENGTH オプションに指定されたフィールド (ここでは LRECL-REC1) からレコードの長さを入手することができます。更新済みレコードを長さの異なるレコードによって置き換えたい場合には、REWRITE コマンドに LENGTH オプションをコーディングする必要があります。

表 2 は、リンケージ・セクションで、BMS マップを処理する新旧の方式を示しています。この例では、OS/VS COBOL プログラムが LANGLVL(1) オプションを使用してコンパイルされ、以下のマップ・セットがインストールされていると想定しています。

```
MAPSET1 DFHMSD TYPE=DSECT,
               TERM=2780,LANG=COBOL,
               STORAGE=AUTO,
               MODE=IN
```

例で使用されている新しい ADDRESS 特殊レジスターは、32 ページの『COBOL での基底付きアドレッシングの使用』で説明されています。

表 2. リンケージ・セクションでの BMS マップ・セットのアドレッシング

OS/VS COBOL	言語環境プログラム準拠の COBOL
WORKING-STORAGE SECTION. 77 FLD0 PIC X VALUE IS LOW-VALUE. LINKAGE SECTION. 01 BLLCELLS. 02 FILLER PIC S9(8) COMP. 02 BLL-DATAA PIC S9(8) COMP. 01 DATA1 COPY MAPSET1. PROCEDURE DIVISION. EXEC CICS GETMAIN LENGTH(1000) SET(BLL-DATAA) INITIMG(FLD0) END-EXEC.	WORKING-STORAGE SECTION. 77 FLD0 PIC X VALUE IS LOW-VALUE. LINKAGE SECTION. COPY MAPSET1. **01 MAP1** **02 FILLER PIC X(12).** **02 FILLER1L COMP PIC S9(4).** . . **02 FIELD90 PIC X(20).** PROCEDURE DIVISION EXEC CICS GETMAIN FLENGTH(LENGTH OF MAP1I) SET(ADDRESS OF MAP1I) INITIMG(FLD0) END-EXEC.

強調表示されているコンポーネントは、MAP1I COBOL コピーブックの内容を説明しています。

第 4 章 C および C++ でのプログラミング

以下の情報を使用すると、CICS アプリケーション・プログラムとして使用する C および C++ プログラムをコーディング、変換、およびコンパイルするのに役立ちます。

「*CICS Transaction Server for z/OS リリース・ガイド*」には、CICS Transaction Server for z/OS バージョン 4 リリース 1 でサポートされている C および C++ コンパイラー、および z/OS におけるサービス状況がリストされています。

CICS Transaction Server for z/OS バージョン 4 リリース 1 の資料において C および C++ に言及する場合、特に注記されていなければ、サポートされている言語環境プログラム対応のコンパイラーを使用することを意味します。

COBOL、PL/I、およびアセンブラー言語のアプリケーションで使用可能なすべての EXEC CICS コマンドは、C および C++ のアプリケーションでも使用できますが、非構造化例外処理に関連するコマンドは例外となります。

C++ のアプリケーションでは、CICS C++ OO クラスを使用して、EXEC CICS インターフェースの代わりに CICS サービスにアクセスすることもできます。このインターフェースについて詳しくは、*CICS C++ OO Class Libraries*を参照してください。

C++ は、オブジェクト指向プログラミングをサポートしており、C 言語と同じ方法で使用することができます。変換プログラムでは、C++ の変換には CPP オプションを使用するように指定する必要があります。また、C++ プログラムは、LANGUAGE(LE370) オプションも使用して定義しなければなりません。

作業用ストレージ

C および C++ では、作業用ストレージはスタックおよびヒープで構成されています。スタックおよびヒープの位置は、16MB 境界に関して、スタックおよびヒープのランタイム・オプションの ANYWHERE および BELOW オプションによって制御されます。デフォルトでは、スタックおよびヒープは両方とも 16MB 境界より上のアドレスに置かれています。

サンプル・プログラム

C または C++ 言語で書かれたプログラムで、EXEC CICS コマンドを使用する方法を示すために、一組のサンプル・アプリケーション・プログラムが提供されています。

表 3. サンプル・プログラム

サンプル・プログラム	マップ・セット	マップ・ソース	トランザクション ID
DFH$DMNU **オペレーター用指示 (3270)**	DFH$DGA	DFH$DMA	DMNU
DFH$DALL **更新 (3270)**	DFH$DGB	DFH$DMB	DINQ、DADD、DUPD

© Copyright IBM Corp. 1989, 2009

表 3. サンプル・プログラム (続き)

サンプル・プログラム	マップ・セット	マップ・ソース	トランザクション ID
DFH$DBRW **ブラウズ (3270)**	DFH$DGC	DFH$DMC	DBRW
DFH$DREN **受注 (3270)**	DFH$DGK	DFH$DMK	DORD
DFH$DCOM **受注キュー印刷 (3270)**	DFH$DGL	DFH$DML	DORQ
DFH$DREP **レポート (3270)**	DFH$DGD	DFH$DMD	DREP

トランザクションおよびプログラム定義は CSD のグループ DFH$DFLA に提供されており、次のコマンドを使用してインストールする必要があります。

```
CEDA INSTALL GROUP(DFH$DFLA)
```

次のレコード記述ファイルは、C または C++ 言語のヘッダー・ファイルとして提供されます。

- DFH$DFIL—FILEA レコード記述子
- DFH$DL86—L860 レコード記述子

C および C++ プログラミングの制約事項および要件

CICS アプリケーション・プログラムとして使用される C または C++ プログラムには、幾つかの制約事項および要件があります。

使用できない関数およびコマンド

非構造化例外処理に関連した EXEC CICS コマンドは、以下のとおりです。

- HANDLE ABEND LABEL(label)
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- PUSH HANDLE
- POP HANDLE

上記のコマンドは、C および C++ アプリケーションではサポートされません。これらのコマンドを使用すると、変換プログラムによる診断が行われます。HANDLE ABEND PROGRAM コマンドは許可されています。

CICS は system() 関数はサポートしていませんが、2 つの CICS コマンド (LINK と XCTL) が同等の働きをします。

CICS では、拡張精度浮動小数点はサポートされません。

C++ は、パック 10 進データをサポートしません。アプリケーションは、文字ストリング・データ・タイプを使用して、パック 10 進データにアクセスする必要があります。パック 10 進データで演算を行うために使用できる関数は、C++ の標準ライブラリーにはありませんが、自分で作成することはできます。時間を指定するオプションを持つ CICS コマンド (DELAY コマンドや POST コマンドなど) を使用する場合は、HOURS、MINUTES、および SECONDS オプションを使用することを

お勧めします。アプリケーションでパック 10 進データ型を処理するための関数を用意すれば、パック 10 進データ型の TIME または INTERVAL オプションを使用して、時間を定義することができます。

C および C++ では、マクロで CICS コマンドを使用することはできません。

固有の C または C++ ファイル操作は、*type=memory* を指定してオープンされるファイルのみを操作します。CICS サポートのアクセス方式への I/O は CICS API を使用しなければなりません。

C および C++ 固有の関数はすべてソース・プログラムで使用することはできますが、以下の関数は推奨されていません。その一部は実行不能で、その結果、関数が失敗したことを示す戻りコードまたはポインターが返されます。使用できるものが一部にありますが、CICS のパフォーマンスまたは実行に影響する場合があります。

- CDUMP
- CSNAP
- CTEST
- CTRACE
- CLOCK (clock() 関数は、値 (time_t) -1 を返します。)
- CTDLI
- SVC99
- SYSTEM
- SETLOCALE

コーディングの要件

- 大文字のみにする必要がある #pragma ディレクティブの CICS キーワードを除き、すべての CICS キーワードは大文字と小文字を混合して入力することができます。
- CICS が、プログラム名、マップ名、キュー名などの固定長の文字ストリングを予期しており、それが予期しているより短い場合には、必要な長さまでリテラルにブランクを埋め込まなければなりません。EXEC DLI コマンドの場合は、リテラルが渡される場合、SEGMENT 名は変換プログラムによって埋め込みが行われます。
- フィールド名の中にはアセンブラーでは受け入れ可能であっても、C または C++ コンパイラーでは異常終了する原因になるものがあります。このようなフィールド名を使用しないように注意してください。$、#、および @ がこれに該当します。
- C++ は、単一行のコメントについては「//」を使用します。EXEC CICS コマンドの中ほどにコメントを置かないでください。例えば、次は作動しない例です。

```
EXEC CICS SEND TEXT FROM(errmsg)
          LENGTH(msglen)  // Send error message to screen
          RESP(rcode)
          RESP2(rcode2);
```

次は、有効な例です。

```
EXEC CICS SEND TEXT FROM(errmsg)
          LENGTH(msglen)
          RESP(rcode)
          RESP2(rcode2);     //Send error message to screen

EXEC CICS SEND TEXT FROM(errmsg)
          LENGTH(msglen)     /* Send error message to screen */
          RESP(rcode)
          RESP2(rcode2);
```

条件処理

C または C++ アプリケーションでは、すべての EXEC CICS コマンドは、NOHANDLE オプションまたは RESP オプションが指定されている場合と同様に取り扱われます。これはつまり、条件が発生したのに処理されないことから生ずる、「システム処置」のトランザクション異常終了が、C または C++ のアプリケーションでは起こり得ないということです。制御は常に次の命令に流れるので、正常応答に関するテストはアプリケーションの責任で行います。

COMMAREA

連絡域のアドレスは、引数として C または C++ main 関数には渡されません。これは、C および C++ 関数は ADDRESS COMMAREA を使用して連絡域のアドレスを入手しなければならないことを意味します。

EIB

EXEC インターフェース・ブロック (EIB) のアドレスは、引数として C または C++ main 関数には渡されません。これは、C および C++ 関数は ADDRESS EIB を使用して EIB のアドレスを入手しなければならないことを意味します。詳しくは、60 ページの『C および C++ から EIB へのアクセス』を参照してください。

LENGTH

LENGTH をサポートするコマンド (例えば、READ、READNEXT、READPREV、および WRITE コマンド) で LENGTH オプションを指定しないと、変換プログラムからデフォルト値が提供されません。事実上、C プログラムの場合、NOLENGTH が暗黙的に指定されます。

OVERFLOW 条件

ACCUM オプションで SEND MAP コマンドから戻るときに、RESP フィールドに OVERFLOW 条件を表示したい場合は、NOFLUSH オプションを指定する必要があります。

AMODE

CICS の下で実行される C および C++ 言語プログラムはすべて、属性 AMODE(31)、RMODE(ANY) を指定してリンク・エディットしなければなりません。これらは、16MB 境界より上のアドレスに常駐する必要があります。

したがって、システム間製品 (CSP) の対話式アプリケーション生成プログラムで作成されたプログラムにパラメーターを渡す際には、以下のいずれかを行う必要があります。

- 16MB より下でパラメーターを渡す。あるいは、
- CSP ロード・ライブラリーと AMODE(31) を再リンクする。

戻り値

C または C++ プログラムを、**exit()** 関数を使用して終了する場合、あるいは CICS RETURN コマンドではなく **return** ステートメントを使用して終了する場合、exit() 関数を経由して渡される値は、プログラムから戻るときに、EIB の EIBRESP2 フィールドに保管されます。

注: プログラムが DPL を使用して別の CICS 領域にあるプログラムにリンクする場合、そのリモート領域からの EIBRESP2 値は、DPL を実行中のプログラムには**戻されません**。

データ宣言

CICS は、C および C++ 用に次のデータ宣言を提供します。

- 実行インターフェース・ブロック定義 (EIB)。EIB 宣言は #ifndef 行と #endif 行で囲まれ、すべての変換済みファイルに含まれます。C または C++ コンパイラーは、重複する宣言を無視します。挿入されたコードには、C または C++ でコーディングされた、EIB のすべてのフィールドの定義が含まれています。
- BMS 画面属性定義: C および C++ バージョンの DFHBMSCA、DFHMSRCA、および DFHAID ファイルが CICS によって提供されており、アプリケーション・プログラマーが BMS を使用する場合に組み込むことができます。
- DL/I サポート: 変換プログラム・オプションを指定した場合、DLI 変換プログラムによって C 言語バージョンの DFHDIB が組み込まれます (CALL DLI インターフェースを使用する場合には、ユーザーが DLIUIB を組み込まなければなりません)。

取り出し機能

言語環境プログラムに準拠したプログラムは、fetch() 関数および release() 関数をサポートします。取り出したモジュールは、自動インストールによって明示的または暗黙的に、CICS に対する PROGRAM リソースとして定義しなければなりません。

ロケール機能

CSD で定義されたロケールに対しては、すべてのロケール機能がサポートされています。IBM 提供のロケールの CSD 定義は、SCEESAMP ライブラリーの CEECCSD メンバーで提供されます。setlocale() 関数は、ロケールが定義されていない場合は NULL を返します。

デバッグ機能

ダンプ関数 csnap()、cdump()、および ctrace() がサポートされています。出力は、CESE 一時データ・キューに送信されます。キューに十分なレコード長 (LRECL) がない場合は、ダンプの書き込みができません。LRECL は、少なくとも 161 は確保することをお勧めします。

iscics 関数

既存のプログラムを適用したり、CICS だけでなく CICS 以外でも動作するように設計されている新規プログラムを作成したりする場合には、iscics() 関数が便利です。この関数は、プログラムが現在 CICS で実行されている場合には非ゼロ値を、それ以外の場合にはゼロを返します。この関数は、C ライブラリーの拡張版です。

ストリング操作関数

C または C++ 標準ライブラリーのストリング処理関数は、ヌル文字をストリングの終わりマーカーとして使用します。CICS はヌル文字をストリングの終わりマーカーとして認識しません。したがって、C または C++ 関数 (例えば、strcmp) を使用して CICS データ域を操作する場合には、注意する必要があります。

argc および argv 引数

通常、2 つの引数 argc および argv が、C または C++ main 関数に渡されます。argc は渡された変数の数を示し、argv はゼロ終了した変数ストリングの配列です。CICS では、argc の値は 1 で、argv[0] はトランザクション ID で、argv[1] は NULL です。

C および C++ での引数の渡し

C および C++ 言語では、引数は、ランタイムにプログラム・スタックにコピーされ、そこで関数によって読み取られます。これらの引数は、値そのものとするか、あるいは渡すデータが入っているメモリーの区域へのポインターとすることができます。ポインターの受け渡しは、**参照による**値の受け渡しとも呼ばれます。

COBOL および PL/I などの他の言語は、通常、参照によって引数を渡します。これは、受け渡しする引数を指すアドレスのリストを、コンパイラーが渡すことを意味します。これが CICS によってサポートされる呼び出しインターフェースです。参照によって引数を渡すためには、変数が既にポインターでない限り、配列を受け渡しする場合のように、**&** によって変数名に接頭部を付けます。

構築プロセスの一部として、コンパイラーは、引数をあるデータ・タイプから別のデータ・タイプに型変換することがあります。例えば **char** 型の引数が、**short** 型または **long** 型に、型変換されることがあります。

C または C++ プログラムから CICS に値を送る場合には、変換プログラムは、正しい形式の引数リストが CICS に渡されるようなコードを生成するために、必要な処置を行います。変換プログラムは、この変換を可能にするために十分な情報を常に持っているわけではありませんが、一般に、引数が単一文字変数またはハーフワード変数の場合には、変換プログラムは、正しいデータ・タイプの変数への事前呼び出しの割り当てを行って、呼び出しで、この一時変数のアドレスを渡します。

CICS からデータを受け取る場合には、変換プログラムは受取変数名に **&** の接頭部を付けます。これにより、C または C++ コンパイラーは、*値によって* ではなく、*参照によって* 値を渡すようになります (文字ストリング名は例外で、未変更のままです)。**&** を追加しない場合、コンパイラーは受取変数をコピーしてから、そのコピーのアドレスを CICS に渡します。このコピー時に起こるどのプロモーションによっても、CICS によって戻されるデータの消失が起こることがあります。

表 4 は、EXEC CICS コマンドで、引数として値を渡す場合に適用される規則を示しています。

表 4. EXEC CICS コマンド内の引数として値を渡す場合の規則

データ・タイプ	使用法	引数のコーディング
文字リテラル	データ値 (送信側)	ユーザーは文字リテラルを直接指定する必要があります。変換プログラムは必要なすべての間接指定を処理します。
文字変数 (char)	データ域 (受信側)	ユーザーは、変数名に接頭部として **&** を付けて、変数へのポインターを指定する必要があります。
文字変数 (char)	データ値 (送信側)	ユーザーは文字変数を直接指定する必要があります。変換プログラムは必要なすべての間接指定を処理します。
文字ストリング・リテラル	名前 (送信側)	ユーザーは、ストリングをリテラル・ストリングとして直接コーディングするか、あるいはストリングの先頭文字を指すポインターを使用することができます。
文字ストリング変数	データ域 (受信側) 名前 (送信側)	受け取るにしても送るにしても、引数は、ストリング (配列の第 1 エレメントのアドレス) を含む文字配列の名前でなければなりません。
整数変数 (short、long、または int)	データ域 (受信側)	ユーザーは、変数名に接頭部として **&** を付けて、変数へのポインターを指定する必要があります。
整数変数 (short、long、または int)	データ値 (送信側)	ユーザーは変数の名前を指定する必要があります。変換プログラムは必要なすべての間接指定を処理します。
整数定数 (short、long、または int)	データ値 (送信側)	ユーザーは整数定数を直接指定する必要があります。変換プログラムは必要なすべての間接指定を処理します。
構造体または共用体	データ域 (送信側) データ域 (受信側)	ユーザーは、名前に接頭部として **&** を付けて、構造体または共用体の先頭のアドレスをコーディングする必要があります。
配列 (任意の型)	データ域 (受信側) データ値 (送信側)	変換プログラムはなにもしません。ユーザーは、配列の先頭メンバーのアドレスをコーディングする必要があります。通常、配列の名前をコーディングして、コンパイラーがそれを先頭メンバーのアドレスとして解釈することによって、これが行われます。
ポインター (任意の対象を指す)	ポインター参照 (受信側) データ域 (送信側)	受け取るにしても送るにしても、引数は、対象アドレスを示す変数の名前とする必要があります。変換プログラムは、CICS がポインターを更新できるようにするために必要な、間接指定の特別なレベルを処理します。
注: 受信側 は CICS からデータを受け取る側で、送信側は CICS にデータを渡す側です。		

C および C++ から EIB へのアクセス

EXEC インターフェース・ブロック (EIB) のアドレスは、引数として C または C++ main 関数には渡されません。これは、C および C++ 関数は ADDRESS EIB コマンドを使用して EIB のアドレスを入手しなければならないことを意味します。

アドレッシング可能にするには、次のコマンドを使用します。

```
EXEC CICS ADDRESS EIB(dfheiptr);
```

これ以外に、外部プロシージャーを呼び出す CALL ステートメントの引数として、EIB アドレス、またはその中の特定のフィールドを渡す方法もあります。

EIB へのアクセスが必要な場合には、各プログラムの開始時に、ADDRESS EIB コマンドが必要です。

C または C++ アプリケーション・プログラムでは、EIB 内のフィールドは小文字で参照され、完全に修飾されます。例えば、「dfheiptr->eibtrnid」のようになります。

データ・タイプには、次のマッピングが使用されます。

- ハーフワード 2 進整数は、「short int」として定義します。
- フルワード 2 進整数は、「long int」として定義します。
- 1 文字フィールドは、「unsigned char」として定義します。
- 文字ストリングは、「unsigned char」配列として定義します。

C および C++ の地域サポート

CICS 変換プログラムはデフォルトでは、C または C++ 言語で書かれたプログラムを、EBCDIC Latin-1 コード・ページ IBM-1047 で編集したものと想定します。

別のコード・ページを使用した場合は、アプリケーション・プログラムの始動時にそのコード・ページをプラグマ・ファイル・タグ・ディレクティブで指定することができます。プラグマ・ステートメントは、そのプログラム内の最初の非コメント・ステートメントである必要があります。また、ファイル・タグ・ディレクティブは、そのプラグマ・ステートメント内で、他のどのディレクティブより前に指定する必要があります。CICS 変換プログラムは、ファイル・タグ・ディレクティブがあるかどうかを、スキャンして調べます。CICS 変換プログラムがサポートしているのは、デフォルトのコード・ページ IBM-1047、デンマーク語の EBCDIC コード・ページ IBM-277、ドイツ語の EBCDIC コード・ページ IBM-273、および中国語の EBCDIC コード・ページ IBM-935 および IBM-1388 のみです。

例えば、ドイツ語の EBCDIC コード・ページを使用するエディターでプログラムを作成した場合、そのプログラムは次のようなディレクティブで始めます。

```
??=pragma filetag ("IBM-273")
```

アプリケーション・プログラムが複数の異なるコード・ページを混ぜて使用する場合 (例えば、通常のソース・ファイルに使用したコード・ページとは異なるコード・ページで編集されたヘッダー・ファイルを組み込む場合) は、デフォルトのコ

ード・ページ IBM-1047 に入っているものも含め、すべてのファイルにプラグマ・ファイル・タグ・ディレクティブを組み込む必要があります。

既にサービスは終了しているが、CICS 変換プログラムではまだ使用可能な一部の古い IBM C コンパイラーの中には、プラグマ・ファイル・タグ・ディレクティブの使用をサポートしていないものもあります。ご使用のコンパイラーでサポートされているかどうか定かではない場合は、そのコンパイラーの資料を確認してください。

XPLink と C および C++ プログラミング

CICS は、XPLINK オプションを使用してコンパイルされた C および C++ プログラムをサポートしています。CICS XPLink サポートを使用するプログラムは、すべて再入可能かつスレッド・セーフである必要があります。

通常は XPLink と省略される Extra Performance Linkage は、ハイパフォーマンス・サブルーチン呼び出しおよび返しのメカニズムを提供する、z/OS の機能です。これにより、実行パスの長さが、短くて高度に最適化されたものとなります。

オブジェクト指向プログラミングは、「メッセージ」をオブジェクトに送信することで、オブジェクトに何らかのアクションを実行させるという概念を基に、構築されています。メッセージ送信アクティビティーは、サブルーチン呼び出しとしてインプリメントされます。C++ の用語でメンバー関数と呼ばれるサブルーチンは、通常はコードの小さな断片です。典型的な C++ プログラムの実行フローは、コードの小さな断片に対する多数のサブルーチン呼び出しに特徴があります。こうした性質を持つプログラムは、XPLink 最適化テクノロジーから恩恵を受けます。

MVS には、初期の System/360 までたどることができる標準サブルーチン呼び出し規則があります。この規則は、複雑なサブルーチンが存在する環境や、サブルーチンが比較的少ない環境、および、サブルーチン呼び出しが比較的少ない環境に対して最適化されていました。これは、オブジェクト指向プログラミングの規則において変更されました。サブルーチンは単純になりましたが、非常に多くなり、サブルーチン呼び出しの頻度はその重要性に応じて増加しました。このようにサブルーチンのサイズ、数、および使用パターンが変化したことにより、関係するシステム・オーバーヘッドの最適化が望まれるようになりました。その最適化の結果が XPLink です。

XPLink を使用するには、その C または C++ アプリケーションのコードが、**再入可能**かつ**スレッド・セーフ**である必要があります。同じコード・インスタンスを複数の MVS TCB で実行することが可能なため、スレッド・セーフ・メカニズムによって共用リソースを保護しないと、アプリケーション・コードの実行時の振る舞いは予測不能となります。これはそれほど強調する必要はありません。

XPLINK オプションを持つ CICS 環境用に C および C++ プログラムをコンパイルする場合、CICS XPLink サポートを活用するためには、アプリケーション開発者は以下を行う必要があります。

- スレッド・セーフ・プログラミングの規則および技法を厳守して、コード開発を行う。

- C または C++ プログラムのコンパイルの際に、XPLINK オプションを設定する。
- PROGRAM リソース定義において、そのプログラムがスレッド・セーフであることを示す。
- CEEUOPT または #pragma で CICSVAR を使用することを検討する (詳しくは、19 ページの『言語環境プログラムのランタイム・オプションの定義』の注を参照してください)。

CICS XPLink サポートを使用するプログラムは、すべて再入可能かつスレッド・セーフである必要があります。こうした要件を確実に満たすように特定のアプリケーション・コードを作成するのは、アプリケーション開発者の責任です。

XPLink による X8 および X9 モード TCB の使用

CICS は、CICS Open Transaction Environment (OTE) テクノロジーで複数の TCB 機能を使用することで、XPLINK オプションを指定してコンパイルされた C および C++ プログラムをサポートしています。X8 および X9 モード TCB は、CICS キーおよび USER キー内の XPLink タスクをサポートするように定義されています。XPLink プログラムの各インスタンスは、X8 または X9 TCB を 1 つ使用します。

XPLINK オプションを指定してコンパイルされたプログラムの CICS サポートに必要なのは、PROGRAM リソース定義で、そのプログラムがスレッド・セーフであることを示すことだけです。この指示とロード・モジュール内の XPLink の「シグニチャー」があれば、タスクを X8 または X9 TCB に置くことができます。

特定のプログラムに適した TCB の選択では、PROGRAM リソース定義に API 属性の OPENAPI 値が存在しても、XPLink が優先されます。

XPLink オブジェクトと非 XPLink オブジェクトの間での制御の引き渡し

XPLink オブジェクトから非 XPLink オブジェクトへ、またはその逆への制御の移動を行うと、その度に QR TCB とオープン TCB (X8 または X9 TCB のいずれか) の間で切り替えが行われます。パフォーマンスの点から言えば、TCB 交換は負荷が大きいため、パフォーマンスのオーバーヘッドを考慮に入れる必要があります。

XPLink オブジェクトは、EXEC CICS インターフェースまたは言語環境プログラムのインターフェースを使用して、非 XPLink オブジェクトを呼び出すことができます。

非 XPLink オブジェクトは、EXEC CICS インターフェースのみを使用して、XPLink オブジェクトを呼び出すことができます。言語環境プログラムのインターフェースを使用した呼び出しは、サポートされていません。

グローバル・ユーザー出口と XPLink

XPCFTCH 出口および XPCTA 出口は、XPLINK オプションの使用によって影響を受けます。CICS は、XPCFTCH がエントリー・ポイントを変更しようとしても、

XPCTA がレジューム・アドレスを定義しようとしてもすべて破棄します。その他のグローバル・ユーザー出口は、XPLink のサポートによる影響はありません。

XPCFTCH

XPLINK オプションを使用してコンパイルされた C または C++ プログラム用に出口 XPCFTCH が呼び出されると、その出口によって指定された変更済みのエントリー・ポイント・アドレスを無視することを示すフラグが設定されます。

XPCTA

XPLINK オプションを使用してコンパイルされた C または C++ プログラム用に出口 XPCTA が呼び出されると、その出口によって指定されたレジューム・アドレスを無視することを示すフラグが設定されます。

XPLink プログラムに使用されるバッチ言語環境プログラム・ランタイムはプログラムの異常終了時に CICS に制御を与えず、独自の異常終了処理を行うので、これらのアクティビティーは無視されます。制御が CICS に到達すると言語環境プログラムのエンクレーブは終了されるので、CICS はエントリー・ポイント・アドレスまたはレジューム・アドレスを引き継ぐことができません。

これらのアクティビティーを実行するアプリケーション・プログラムがある場合は、これらの要求を管理するために他の方法を検索するか、そのプログラムが XPLINK の最適化に適切ではないと判断する必要があります。可能性のある解決方法の 1 つは、「*z/OS 言語環境プログラム カスタマイズ*」マニュアルの『ユーザー出口のカスタマイズ』で説明されているように、言語環境プログラムの異常終了出口を書き込みすることです。

第 5 章 PL/I でのプログラミング

以下の情報を使用すると、CICS アプリケーション・プログラムとして使用する PL/I プログラムをコーディング、変換、およびコンパイルするのに役立ちます。

「*CICS Transaction Server for z/OS リリース・ガイド*」には、CICS Transaction Server for z/OS バージョン 4 リリース 1 でサポートされている PL/I コンパイラー、および z/OS におけるサービス状況がリストされています。

CICS Transaction Server for z/OS バージョン 4 リリース 1 の資料において PL/I に言及する場合はすべて、特に注記されていなければ、サポートされている言語環境に準拠したコンパイラーを使用することを意味しています。

OPTIONS(MAIN) の指定

OPTIONS(MAIN) を指定した PL/I アプリケーション・プログラムは、トランザクションの最初のプログラムにすることも可能ですし、LINK コマンドまたは XCTL コマンドを使用してそのプログラムへ制御を渡すことも可能です。

OPTIONS(MAIN) が指定されていない PL/I アプリケーション・プログラムは、トランザクションの最初のプログラムとなることも、LINK または XCTL コマンドによって制御を渡されることもできませんが、メインプログラムにリンク・エディットすることはできます。

FLOAT コンパイラー・オプション

Enterprise PL/I については、FLOAT オプションの指定によって追加浮動小数点レジスターの使用が制御されます。

- ユーザーのプログラムで浮動小数点をほとんど使用しない場合は、FLOAT(NOAFP) オプションを指定します。そのプログラムでは、従来からある 4 つの浮動小数点レジスターのみを使用し、レジスターの保管時の処理は少ないです。
- ユーザーのプログラムで浮動小数点を主に使用する場合は、FLOAT(AFP) オプション または FLOAT(NOVOLATILE) オプションを指定します。そのプログラムでは、16 の浮動小数点レジスターのすべてを使用し、CICS ではプログラムによって使用された浮動小数点レジスターが保存されます。
- FLOAT(AFP(VOLATILE)) オプションを指定する場合は、CICS と PL/I の**両方**で浮動小数点レジスターが保存されます。余計なコードが生成され、それによりパフォーマンスが損なわれます。

注: CICS では浮動小数点レジスターが CICS Transaction Server for z/OS バージョン 4 リリース 1 から保存されています。

PL/I プログラミングの制約事項と要件

CICS アプリケーション・プログラムとして使用する PL/I プログラムには、いくつかの制約事項および要件があります。

© Copyright IBM Corp. 1989, 2009

使用できない関数およびステートメント

- 次のマルチタスク組み込み関数を使用することはできません。

```
COMPLETION
PRIORITY
STATUS
```

- 次のマルチタスク・オプションを使用することはできません。

```
EVENT
PRIORITY
TASK
```

- 次の PL/I ステートメントは使用しないでください。

```
CLOSE                   DELAY
DELETE                  DISPLAY
EXIT                    GET
HALT                    LOCATE
OPEN                    PUT
READ                    REWRITE
STOP                    WRITE
UNLOCK
```

  FETCH ステートメントおよび RELEASE ステートメントはサポートされています。データの保管および検索のため、および端末との通信のために、EXEC CICS コマンドが提供されます (しかし、SYSPRINT には CLOSE、PUT、および OPEN を使用することができます)。

- PL/I のソート・マージを使用することはできません。
- 静的ストレージを使用することはできません (ただし、読み取り専用データの場合は除きます)。

コーディングの要件

- 変数を STATIC 属性および EXTERNAL 属性を使用して宣言する場合には、INITIAL 属性も含める必要があります。含めない場合には、このような宣言は、CICS が処理できない共通の CSECT を生成します。
- 変換プログラムによって生成される変数名と同じ変数名をもつ変数、または構造体を定義しないでください。これらは DFH で始まります。LIKE キーワードの使用によって、このような変数名が暗黙のうちに生成されるのを避けるように注意する必要があります。
- 小文字でもかまわない PROCEDURE 名は例外として、すべての PROCEDURE ステートメントは大文字でなければなりません。
- *PROCESS ステートメントの XOPTS オプションのサブオプションは、大文字でなければなりません。
- EXEC CICS ステートメントでは、PL/I の 48 文字セット・オプションは使用できません。
- CICS コマンドがデータ値の定義で SUBSTR 組み込み関数を使用する場合には、LENGTH オプションを組み込んでデータ長を指定する必要があります。ただし、変換プログラム・オプション NOLENGTH が指定されている場合はその必要はありません。データ長を指定しない場合には、変換プログラムは、次の形式の CSTG 組み込み関数の呼び出しを含む PL/I 呼び出しを生成します。

```
CSTG(SUBSTR(..,..,..))
```

  これは、コンパイラーによって拒否されます。

PL/I アプリケーションにおける言語環境プログラムのコーディング要件

すべての PL/I プログラムは、言語環境プログラムで提供されるランタイム・サポートの下で実行されます。言語環境プログラム以前の PL/I プログラムに比べて、いくつか追加のコーディング要件があります。

言語環境プログラムのランタイム・オプションは、必要に応じて **plixopt** 文字ストリングで指定できます。ランタイム・オプションのカスタマイズについて詳しくは、19 ページの『言語環境プログラムのランタイム・オプションの定義』 および「*z/OS Language Environment* プログラミング・リファレンス」を参照してください。

言語環境プログラムに準拠していないコンパイラーを使用して以前にコンパイルした PL/I プログラムを変換する場合は、**plixopt** ストリングで NOSTAE も NOSPIE も指定しないようにしてください。指定すると、言語環境プログラムが TRAP (OFF) に設定されてしまいます。アプリケーションが正常に動作するには、TRAP (ON) が有効になっていなければなりません。

エントリー・ポイント

CEESTART は、言語環境プログラムの下で実行されている PL/I アプリケーションでは、唯一の入り口点です。この入り口点は、言語環境プログラム準拠のコンパイラーを使用してコンパイルされているプログラム用に設定されています。

言語環境プログラムに準拠していないコンパイラーで作成したオブジェクト・モジュールは、以下のリンケージ・エディター・ステートメントを使用して、言語環境プログラムの下で実行されるように再リンクすることができます。

```
INCLUDE SYSLIB(CEESTART)
INCLUDE SYSLIB(CEESG010)
INCLUDE SYSLIB(DFHELII)
REPLACE PLISTART
CHANGE PLIMAIN(CEEMAIN)
INCLUDE  mainprog
INCLUDE   subprog1
.......
.......
ORDER CEESTART
ENTRY CEESTART
NAME  progname(R)
```

オブジェクト・モジュールの INCLUDE ステートメントは、CHANGE ステートメントの直後に配置する必要があります。また、言語環境プログラムでは、メインプログラムはサブルーチンの前に組み込む必要があるという要件もあります (準拠していないコンパイラーで作成されたモジュールには、このような要件はありません)。

OPTIONS(FETCHABLE) を使用してコンパイルされた Enterprise PL/I プログラムの場合には、バインダーの ENTRY ステートメントを PROCEDURE の名前にする必要があります。

PL/I の再リンク・ユーティリティー

準拠していないコンパイラーでコンパイルされた CICS プログラム用のロード・モジュールしかない場合は、CICS プログラム用の特定のリンケージ・エディター入力

ファイル IBMWRLKC が、サンプル・ライブラリー SCEESAMP にあります。この入力ファイルにより、非準拠の実行可能プログラム内の OS PL/I ライブラリー・ルーチンが、言語環境プログラムのルーチンに置き換えられます。

IBMWRLKC の使用について詳しくは、「*PL/I MVS & VM コンパイラーおよびランタイム移行ガイド*」を参照してください。

準拠と非準拠の各 PL/I ルーチン間の通信

言語環境プログラム準拠の PL/I プログラムは、FETCH または RELEASE ステートメントで表示されるプログラムを呼び出したり、そのプログラムをあとで解放したりすることができます。

言語環境プログラムに準拠していない PL/I サブルーチンを、言語環境プログラム準拠のメインプログラムとリンク・エディットすることができます。

静的呼び出しは、PL/I のどのバージョンからでもサポートされていますが、動的呼び出しは、言語環境プログラム準拠のプロシージャーからしかサポートされていません。

呼び出し先のサブルーチンは、そのサブルーチンで EIB のアドレスが使用できるなら、CICS コマンドを発行することができます。これは、EIB のアドレスをサブルーチンに渡すか、他の CICS コマンドを発行する前に、サブルーチンで EXEC CICS ADDRESS EIB(DFHEIPTR) をコーディングするかのいずれかの方法で、達成できます。

異常終了処理

言語環境プログラムで CICS PL/I プログラムが異常終了する場合は、CICS 異常終了ハンドラーには、PL/I 異常終了コードではなく、言語環境プログラム異常終了コードが与えられます。

プログラムの変更を避けるために、SCEESAMP ライブラリー内の言語環境プログラムが提供する、サンプル・ユーザー条件ハンドラー CEEWUCHA を変更することができます。このユーザー条件ハンドラーが、言語環境プログラムのコードではなく、PL/I 異常終了コードを返すようにすることができます。USRHDLR ランタイム・オプションを使用し、これを実行するように登録してください。このオプションについて詳しくは、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。

取り出した PL/I ルーチン

PL/I プロシージャーの取り出しができるようにするには、PROCEDURE ステートメント上の OPTIONS で、オプション FETCHABLE をコーディングします。

FETCHABLE オプションは、プロシージャーを動的にのみ呼び出すように指示します。OPTIONS(MAIN) プロシージャーを取り出すことはできません。FETCHABLE と MAIN は、互いに排他的なオプションです。

OPTIONS(FETCHABLE) を使用してコンパイルされた Enterprise PL/I プログラムの場合には、バインダーの ENTRY ステートメントを PROCEDURE の名前にする必要があります。

FETCHABLE プロシージャーは、通常の CICS プログラムと同様に処理してください。つまり、CSD で、あるいはプログラムの自動インストールを使用して、必要なサブルーチンとリンク・エディットし、CICS アプリケーション・プログラム・ライブラリーに入れ、定義し、プログラムとしてインストールします。

言語環境プログラム準拠の PL/I プログラムは、FETCH または RELEASE ステートメントで表示されるプログラムを呼び出したり、そのプログラムをあとで解放したりすることができます。

取り出したプロシージャーで使用可能な PL/I for MVS & VM のステートメントには、いくつかの制約事項がありました。これらの制約事項は、「*PL/I MVS & VM 言語解説書*」で説明されています。VisualAge® PL/I では、その制約事項の多くが除去されています。「*VisualAge PL/I Compiler and Runtime Migration Guide*」を参照してください。

取り出しプログラムと取り出されるプログラムの AMODE 属性が同じ場合は、FETCH の使用に適用される特別な考慮事項はありません。ただし、言語環境プログラムは、FETCH を発行するプログラムとは異なる AMODE 属性を持つロード・モジュールの取り出しもサポートしています。この場合には、言語環境プログラムは AMODE 切り替えを実行し、以下の制約が適用されます。

- 取り出されるモジュールが 24 ビット・アドレッシング・モードで実行されている場合は、取り出しモジュールは、その AMODE 属性に関係なく、RMODE(24) 属性を持っていなければならない。
- 取り出されるルーチンに渡される任意の変数は、取り出されるプロシージャーの AMODE にアドレッシング可能でなければならない。

第 6 章 アセンブラー言語でのプログラミング

以下の情報を使用すると、CICS アプリケーション・プログラムとして使用するアセンブラー言語プログラムをコーディングするのに役立ちます。

作業用ストレージ

アセンブラー言語プログラムの作業用ストレージは、CSD 内の PROGRAM 定義上の **DATALOCATION** パラメーター値に従って、16MB 境界の上下いずれかに割り振られます。

アセンブラー言語プログラミングの制約事項および要件

CICS アプリケーション・プログラムとして使用されるアセンブラー言語プログラムには、幾つかの制約事項および要件があります。

使用できない命令

CICS アプリケーション・プログラムとして使用するアセンブラー言語プログラムでは、次の命令を使用することはできません。

COM ブランクの共通制御セクションを識別します。

ICTL 形式制御を入力します。

OPSYN
命令コードを等価にします。

LEASM オプション

アセンブラー言語プログラムが LEASM オプションで変換される場合には、以下の制約事項が適用されます。

- レジスター 2 をコード基底レジスターとして使用することはできません。
- レジスター 12 は、言語環境プログラムが言語環境プログラム共通アンカー域 (CAA) を指すために予約済みであるため、適切に保管および復元されたプログラム以外のプログラムでは、一切使用できません。
- レジスター 13 は、唯一の作業用ストレージ基底レジスターとして使用しなければなりません。
- プログラムが、Global User Exit (GLUE) プログラムまたは Task-Related User Exit (TRUE) プログラムであってはいけません。
- プログラムでは AMODE(24) コードを使用できません。またはこのコードに依存してはいけません。

BAKR 命令 (分岐およびスタック)

BAKR 命令 (分岐およびスタック) を使用して、アセンブラー言語プログラム間のリンクを提供する場合には、リンクされているプログラムが EXEC CICS 要求を発行しないようにします。CICS が制御を受け取り、リンクされているプログラムが

© Copyright IBM Corp. 1989, 2009

PR 命令 (プログラム戻り) によって戻る前に、タスク切り替えを実行する場合には、他のタスクはディスパッチされて、BAKR / PR 呼び出しを発行することができます。これらによってリンケージ・スタックが修正されると、元のタスクが PR 命令を発行したときに、誤った環境がリストアされることになります。

HANDLE ABEND LABEL

CICS では、DFHEIENT および DFHEIRET を使用しないアセンブラー言語プログラムで HANDLE ABEND LABEL を使用することはできません。言語環境プログラムのスタブ CEESTART を使用するアセンブラー言語プログラムは、HANDLE ABEND PROGRAM、あるいは CEEHDLR のような言語環境プログラム・サービスを使用しなければなりません。CEEHDLR の詳細については、14 ページの『言語環境プログラムの異常終了および条件処理』を参照してください。

31 ビット・アドレッシング

31 ビット・モードで実行しているアセンブラー言語アプリケーション・プログラムには、次の制約事項が適用されます。

- インターバル制御コマンド WAIT EVENT は、関連するイベント制御ブロック (ECB) が 16MB の境界より上にある場合にはサポートされません。その代わりに、タスク制御コマンド WAIT EXTERNAL に ECBLIST オプションを指定して、使用することができます。
- COMMAREA オプションは、混合アドレッシング・モードのトランザクション環境では制限されています。制約事項については、332 ページの『混合アドレッシング・モードの使用』を参照してください。

アクセス・レジスター

ESA/370 プロセッサーの拡張アドレッシング機能を活用するために、アクセス・レジスターを使用するアセンブラー言語アプリケーション・プログラムには、次の制約事項が適用されます。

- CICS サービスを呼び出す際には、基本アドレッシング・モードにいなければなりません。基本アドレス・スペースは、ホーム・アドレス・スペースにしてください。CICS に渡されるすべてのパラメーターを、基本アドレス・スペースに常駐させてください。
- CICS は、アクセス・レジスターを***常に*** 保持するとは限りません。アクセス・レジスターは、CICS サービスを呼び出す前に保管する必要があり、再度使用する前に、復元する必要があります。

アクセス・レジスターの使用についてのガイダンス情報は、「*z/OS MVS Programming: Extended Addressability Guide*」を参照してください。

64 ビット・レジスター

64 ビットのアドレッシング・モードまたは 64 ビットの 2 項演算を活用するために、64 ビット・レジスターを使用するアセンブラー言語アプリケーション・プログラムには、以下の制約事項が適用されます。

- CICS は、64 ビット・レジスターの上位ワードを***常に保存しているとは限りません***。アクセス・レジスターは、CICS サービスを呼び出す前に保管する必要があり、64 ビット・レジスターを再度使用する前に、復元する必要があります。

64 ビット・アドレッシング・モードおよび 64 ビット 2 項演算の使用についてのガイダンス情報は、「*z/OS MVS プログラミング: アセンブラー・サービスガイド*」を参照してください。

アセンブラー言語アプリケーションのための言語環境プログラムのコーディング要件

HLL プログラムと同様、アセンブラー言語プログラムも、言語環境プログラムに準拠しているか準拠していないかのいずれかに分類されます。アセンブラー言語プログラムの場合、準拠しているかどうかは、使用されるアセンブラーではなく、リンケージおよびレジスターの規則に準拠しているかどうかによって決まります。定義上、言語環境プログラム準拠のアセンブラー言語ルーチンは、CEEENTRY およびそれに関連した言語環境プログラム・マクロを使用してコーディングされたものとして定義されます。

準拠しているかどうかによって、HLL プログラムからの呼び出しによりアセンブラー・プログラムを使用するかどうかが決まります。準拠しているアセンブラー言語サブルーチンも準拠していないアセンブラー言語サブルーチンも、C、C++、COBOL または PL/I から、静的あるいは動的に呼び出すことができます。しかし、この 2 つのタイプでは、レジスターの規則やその他の要件に違いがあります。例えば、言語環境プログラムに準拠しているアセンブラー言語ルーチンを使用して適切に通信を行うには、アセンブラー言語ルーチンへの入り口、その実行中、およびアセンブラー言語ルーチンからの出口で、一定のレジスター規則に従う必要があります。

アセンブラー言語を含む、言語混合の規則については、16 ページの『言語環境プログラムにおける言語の混合』で説明しています。

詳細について、あるいはこのセクションで使用されている用語の説明については、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。

MAIN プログラムへの準拠

言語環境プログラムのインターフェースに準拠するように新規のアセンブラー言語 MAIN プログラムをコーディングする場合、あるいはアセンブラー言語ルーチンで言語環境プログラムのサービスを呼び出す場合は、以下に従ってください。

- 言語環境プログラムが提供するマクロを使用する。これらのマクロのリストについては、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。
- CEEENTRY マクロに必ず、オプション MAIN=YES を含める (MAIN=YES がデフォルトです)。
- アセンブラー言語ルーチンを、*ASM XOPTS(LEASM) を使用して変換する。あるいは、アセンブラー言語ルーチンに CICS コマンドが含まれている場合は、*ASM XOPTS(LEASM NOPROLOG NOEPILOG) を使用して変換します。

サブルーチンの準拠

言語環境プログラムのインターフェースに準拠するように新規のアセンブラー言語サブルーチンをコーディングする場合、あるいはアセンブラー言語ルーチンで言語環境プログラムのサービスを呼び出す場合は、以下に従ってください。

- 言語環境プログラムが提供するマクロを使用する。これらのマクロのリストについては、「*z/OS Language Environment プログラミング・ガイド*」を参照してください。
- CEEENTRY マクロに必ず、オプション MAIN=NO (デフォルトは MAIN=YES) を含める。
- アセンブラー言語ルーチンに CICS コマンドが含まれている場合は、*ASM XOPTS(NOPROLOG NOEPILOG) を使用して変換する。
- ご使用のルーチンが VS COBOL II からの静的呼び出しで起動される場合は、CEEENTRY マクロに必ずオプション NAB=NO を含める (NAB とはストレージの **N**ext **A**vailable **B**yte (次に使用できるバイト) の意です。NAB=NO は、このフィールドは使用できない可能性があるという意味です。この場合、CEEENTRY マクロは、使用可能なストレージを検索するコードを生成します)。

準拠ルーチンへの入り口のレジスター規則

言語環境プログラム準拠のアセンブラー言語サブルーチンへの入り口では、CEEENTRY マクロで NAB=YES が指定されている場合は、これらのレジスターに以下の値が含まれていなければなりません。

R0 予約済み

R1 パラメーター・リストのアドレス、あるいはゼロ

R12 共通アンカー域 (CAA) のアドレス

R13 呼び出し元の動的ストレージ域 (DSA)

R14 リターン・アドレス

R15 入り口点のアドレス

言語環境プログラム準拠の HLL は、これらのレジスター規則に従ったコードを生成します。提供されるマクロも、ユーザーがそれらを使用して言語環境プログラム準拠のアセンブラー言語ルーチンを作成した場合、同じ動作をします。アセンブラー言語ルーチンへの入り口で、CEEENTRY は、呼び出し元のレジスター (R14 から R12 まで) を、その呼び出し元が提供する DSA に保管します。そして新規 DSA を割り振り、この新規 DSA に NAB フィールドを正しく設定します。新規 DSA のワードの前半は 2 進ゼロに設定され、2 番目のワードのバック・チェーンは、呼び出し元の DSA を指すように設定されます。

準拠ルーチンの実行中のレジスター規則

R13 は、言語環境プログラム準拠のアセンブラー言語ルーチンが実行中はいつでも、そのルーチンの DSA を指していなければなりません。

コード内の別のプログラムを呼び出す任意のポイントで、R12 には、共通アンカー域 (CAA) のアドレスが含まれていなければなりません。ただし、以下の場合は例外です。

- COBOL プログラムを呼び出すとき。
- 言語環境プログラムに準拠していないアセンブラー言語ルーチンを呼び出すとき。
- CEEENTRY マクロで NAB=NO を指定した、言語環境プログラム準拠のアセンブラー言語ルーチンを呼び出すとき。

準拠ルーチンからの出口のレジスター規則

言語環境プログラム準拠のアセンブラー言語ルーチンからの出口では、R0、R1、R14、および R15 が未定義です。それ以外のレジスターはすべて、入り口時点と同じ内容でなければなりません。

CEEENTRY マクロは、モジュールを自動的に AMODE (ANY) および RMODE (ANY) に設定します。既存のアセンブラー言語ルーチンを言語環境プログラムに準拠するように変換し、そのルーチンに、24 ビット・アドレッシング・モードを使用してコーディングされたデータ管理マクロが含まれている場合は、そのマクロを、31 ビット・モードを使用するように変更しなければなりません。プログラムを構成するすべてのモジュールが 31 ビット・アドレッシング・モードを使用するように変更できない場合、および、明示的に RMODE (24) を設定するモジュールがない場合は、リンク・エディット・プロセス中にプログラムを RMODE (24) に設定してください。

言語環境プログラムの下で実行中の非準拠アセンブラー言語ルーチン

言語環境プログラムに準拠していないアセンブラー言語ルーチンを言語環境プログラムの下で実行する場合は、以下の規則に従ってください。

- R13 には、実行するルーチンのレジスター保管域のアドレスが含まれていなければならない。
- レジスター保管域の最初の 2 バイトは、2 進ゼロでなければならない。
- レジスター保管域のバック・チェーンを、有効な 31 ビット・アドレスに設定しなければならない (24 ビット・アドレスの場合は、高位バイトがゼロになっていなければなりません)。

ご使用のアセンブラー言語ルーチンが C、C++、COBOL、または PL/I 制御ブロックに依存している場合 (例えば、これらの制御ブロックでフラグまたはスイッチをテストするルーチン) は、これらの制御ブロックが言語環境プログラムの下でも変更されていないことを確認してください。詳細については、使用している言語の「*Compiler and Run-Time Migration Guide*」を参照してください。

準拠していないアセンブラー言語ルーチンでは、言語環境プログラムの呼び出し可能サービスは使用できません。

アセンブラー言語プログラムの呼び出し

コマンドを含んでいるアセンブラー言語アプリケーション・プログラムは、固有の RDO プログラム定義を持っている場合があります。このようなプログラムは、LINK コマンドまたは XCTL コマンドを使用して、COBOL、C、C++、PL/I、あるいはアセンブラー言語アプリケーション・プログラムによって、呼び出すことができます。しかし、コマンドを含むプログラムはシステム標準呼び出しによって呼び

出されるので、COBOL、C、C++、または PL/I の CALL ステートメントによって呼び出すか、あるいはアセンブラー言語の CALL マクロによって呼び出すこともできます。

RDO プログラム定義に定義されていれば、別個の CSECT を個別にコンパイルまたはアセンブルしたものを 1 つにリンクして、1 つの CICS アプリケーション・プログラムを構成することができます。

コマンドを含むアセンブラー言語アプリケーション・プログラムは、他のアセンブラー言語プログラムか、または 1 種類以上の高水準言語 (COBOL、C、C++、または PL/I) で書かれたプログラムとリンクすることができます。アプリケーション・ロード・モジュールにおける言語の混合について詳しくは、16 ページの『言語環境プログラムにおける言語の混合』を参照してください。

アセンブラー言語プログラム (個別にリンク・エディットされているもの) が、コマンド・レベル呼び出しを含んでいて、高水準言語プログラムから呼び出される場合は、アセンブラー言語プログラムは固有の CICS インターフェース・スタブを必要とします。アセンブラー・プログラムが、そのプログラムを呼び出す高水準言語プログラムとリンク・エディットされている場合は、そのアセンブラー・プログラムにはスタブは必要ありません。スタブを提供すると、メッセージ MSGIEW024I が出されますが、このメッセージは無視しても差し支えありません。

コマンドを含むアセンブラー言語アプリケーション・プログラムは、呼び出されると必ず、パラメーターの EIB と COMMAREA を渡されるので、CALL ステートメントまたはマクロは、これら 2 つのパラメーターと、その後に続くオプションのパラメーターを渡さなければなりません。

例えば、ファイル PLITEST PLI 内の PL/I プログラムは、アセンブラー言語プログラム ASMPROG を呼び出します。このプログラムは、ファイル ASMTEST ASSEMBLE 内にあります。PL/I プログラムはアセンブラー言語プログラムに 3 つのパラメーターを渡します。3 つのパラメーターとは、EIB、COMMAREA、およびメッセージ・ストリングです。

```
PLIPROG:PROC OPTIONS(MAIN);
  DCL ASMPROG ENTRY EXTERNAL;
  DCL COMA CHAR(20), MSG CHAR(14) INIT('HELLO FROM PLI');
  CALL ASMPROG(DFHEIBLK,COMA,MSG);
  EXEC CICS RETURN;
 END;
```

図 8. PLITEST PLI

アセンブラー言語プログラムは EXEC CICS SEND TEXT コマンドを実行します。これは、PL/I プログラムから渡されたメッセージ・ストリングを表示します。

```
DFHEISTG DSECT
MSG      DS    CL14
MYRESP   DS    F
ASMPROG  CSECT
         L     5,8(1)
         L     5,0(5)
         MVC   MSG,0(5)
         EXEC CICS SEND TEXT FROM(MSG) LENGTH(14) RESP(MYRESP)
         END
```

図 9. ASMTEST ASSEMBLE

以下のように、CICS で提供される JCL プロシージャーを使用して、アプリケーションのコンパイルとリンクを行うことができます。

1. DFHEITAL プロシージャーを使用して、ASMTEST のアセンブルおよびリンクを行います。

   ```
   //ASMPROG EXEC DFHEITAL
   //TRN.SYSIN DD *
       .... program source ...
   /*
   //LKED.SYSIN DD *
     NAME ASMTEST(R)
   /*
   ```

2. DFHYITPL プロシージャーを使用して PLITEST のコンパイルとリンクを行います。そして、DFHEITAL プロシージャーによって作成された ASMTEST ロード・モジュールを組み込むリンケージ・エディター制御ステートメントを指定します。

   ```
   //PLIPROG EXEC DFHYITPL
   //TRN.SYSIN DD *
       .... program source ...
   /*
   //LKED.SYSIN DD *
     INCLUDE SYSLIB(ASMTEST)
     ENTRY CEESTART
     NAME PLITEST(R)
   /*
   ```

   注: ステップ 2 は、DFHEITAL によって作成された ASMTEST ロード・モジュールが、SYSLIB データ・セット連結に含まれているライブラリーに格納されたことを前提としています。

DFHYITPL プロシージャーによって作成されたロード・モジュールは、(DFHEITAL によって組み込まれた) DFHEAI スタブと (DFHYITPL によって組み込まれた) DFHELII スタブの両方を組み込みます。この両方のスタブには DFHEII というエントリー・ポイントが含まれているので、これらのスタブの組み込みによって、リンケージ・エディターまたはバインダー・プログラムは警告メッセージを発行します。このメッセージは無視して構いません。

ユーザー独自の JCL を作成する場合は、DFHELII スタブにすべての言語に必要なエントリー・ポイントが含まれているので、DFHELII スタブを組み込むだけで済みます。

他のプログラムが呼び出すアセンブラー言語アプリケーション・プログラムは、DFHEIENT マクロで始まり、DFHEIRET マクロで終わります。CICS 変換プログラ

ムがこれらを挿入します。そのため、先の例のように、プログラムが EXEC CICS コマンドを含み、変換プログラムに渡される場合は、これらのマクロをコーディングする必要はありません。

第 3 部 アプリケーション・プログラムの変換、コンパイル、インストール、およびテスト

CICS 内でのアプリケーション・プログラムの変換、コンパイル、およびインストール方法、およびテスト・アプリケーションに備えられた機能の説明。

© Copyright IBM Corp. 1989, 2009

第 7 章 変換およびコンパイル

ほとんどの古いコンパイラー (およびアセンブラー) では、CICS コマンドを直接には処理できません。つまり、ユーザー・プログラムを実行可能コードに変換する追加のステップが必要です。このステップは**変換**と呼ばれ、CICS コマンドを、プログラムの残りの部分をコーディングしている言語に変換して、コンパイラー (またはアセンブラー) が解釈できるようにします。

最新のコンパイラーでは **統合 CICS 変換プログラム**の方法を使用し、コンパイル時に CICS のコンパイラー・インターフェースが CICS コマンドを解釈して、CICS サービス・ルーチンを呼び出すようそのコマンドを自動的に変換できます。統合 CICS 変換プログラムを使用すると、84 ページの『変換のプロセス』 で説明する変換タスクの多くはコンパイル時に実行されるので、変換プログラムのステップを追加で実行する必要はありません。

このセクションでは、以下について説明します。

- 『統合 CICS 変換プログラム』
- 84 ページの『変換のプロセス』
- 87 ページの『CICS 提供の変換プログラム』
- 88 ページの『CICS 変換プログラムの使用』
- 90 ページの『変換プログラムのオプションの定義』
- 105 ページの『COPY ステートメントの使用』
- 106 ページの『CICS 提供のインターフェース・モジュール』
- 107 ページの『EXEC インターフェース・モジュールの使用』

統合 CICS 変換プログラム

統合 CICS 変換プログラムを使用すると、単一ステップで高水準のソース・コードの変換とコンパイルを行うことができます。統合変換プログラムをサポートするコンパイラーは、アプリケーション・ソースをスキャンし、関連するポイントで統合変換プログラムを呼び出します。統合変換プログラムは、EXEC CICS コマンドをコメント化し、その言語に対応する呼び出しを生成します。

以下の言語には統合 CICS 変換プログラムがあります。

C
C++
COBOL
PL/I

統合変換プログラムを使用すると、個別の変換ステップがないためにアプリケーション開発が加速されます。元のソース・ステートメントと CICS エラー・メッセージがコンパイラー・リストに組み込まれてリストが 1 つになるので、アプリケーション開発が容易になります。CICS 提供の独立した変換プログラムでは、ソース・プログラムの行番号が変更されます。これは、変換プログラムが生成した CALL が含

© Copyright IBM Corp. 1989, 2009

まれる中間リストを必要とするという意味で、アプリケーション・プログラムをデバッグする際はこれを使用しなければなりません。

また、この変換とコンパイルのプロセスでは、組み込みメンバーを別々に変換する必要がなくなったため、エラーが起こりにくくなりました。

言語環境プログラムに準拠した、統合 CICS 変換プログラムをサポートする言語コンパイラーは、アプリケーション・ソースをスキャンし、関連するポイントで統合 CICS 変換プログラムを呼び出します。

CICS 統合変換プログラムをサポートする言語コンパイラーのリリースは、 「*CICS Transaction Server for z/OS リリース・ガイド*」にリストされています。その他のコンパイラーまたはアセンブラーを使用する場合は、プログラムをコンパイルする前に変換する必要があります。

統合 CICS 変換プログラムの使用

言語コンパイラーでは、統合 CICS 変換プログラムで使用できるさまざまなプロシージャーを提供しています。それらのプロシージャーは、Enterprise COBOL for z/OS、C/C++ for z/OS、および Enterprise PL/I for z/OS の各プログラミング・ガイドに記載されています。

このタスクについて

使用するプロシージャーでは、コンパイルのステップで STEPLIB 連結に CICSTS41.CICS.SDFHLOAD を追加する必要があり、リンク・エディットのステップの開始時にインターフェース・モジュール DFHELII を組み込む必要があります。

PL/I で統合 CICS 変換プログラムを使用するには、コンパイラー・オプション SYSTEM(CICS) を指定する必要があります。

COBOL で統合 CICS 変換プログラムを使用するには、コンパイラー・オプション CICS、LIB、NODLL、NODYNAM、および RENT を有効にする必要があります。NODLL および NODYNAM は、統合変換プログラムに固有の制限ではありません。DLL および DYNAM は、個別に変換およびコンパイルされたコードではサポートされていません。統合 CICS 変換プログラムの各サービス用のストレージをユーザー領域に残しておく必要があるため、SIZE(MAX) は使用しないでください。代わりに、ほとんどのプログラムで有効と考えられる SIZE(4000K) などの値を使用します。

C および C++ で統合 CICS 変換プログラムを使用するには、CICS オプションを使用します。

DB2 バージョン 7 以降を実行しており、統合変換プログラムが組み込まれたコンパイラーを使用して COBOL プログラムを準備する場合、コンパイラーにより SQL ステートメント・コプロセッサー (DBRM を生成する) も提供されるので、独立した DB2 プリコンパイラーを使用する必要はありません。SQL ステートメント・コプロセッサーの使用方法について詳しくは、 「*CICS DB2 Guide*」 および 「*DB2 for z/OS Application Programming and SQL Guide*」を参照してください。

CICS 変換プログラムのオプションの指定

このタスクについて

PL/I コンパイラーを使用する場合に CICS 変換プログラム・オプションを指定するには、コンパイラー・オプション PP(CICS) に、アポストロフィで囲んで括弧に入れた変換プログラム・オプションを付けて指定します。例:

```
PP(CICS('opt1 opt2 optn ...'))
```

PL/I コンパイラー・オプションの指定について詳しくは、「*Enterprise PL/I for z/OS プログラミング・ガイド*」を参照してください。

COBOL コンパイラーを使用する場合に CICS 変換プログラム・オプションを指定するには、コンパイラー・オプション CICS に、アポストロフィで囲んで括弧に入れた変換プログラム・オプションを付けて指定します。例:

```
CICS('opt1 opt2 optn ...')
```

注: XOPTS 変換プログラム・オプションは、CICS コンパイラー・オプションに変更する必要があります。統合 CICS 変換プログラムを使用する場合、XOPTS は使用できません。

COBOL コンパイラー・オプションの指定について詳しくは、「*Enterprise COBOL for z/OS: Programming Guide*」を参照してください。

XL C および C++ コンパイラーを使用する場合に CICS 変換プログラム・オプションを指定するには、コンパイラー・オプション CICS に、コンマで分けて括弧に入れた変換プログラム・オプションを付けて指定します。例:

```
CICS(opt1,opt2,optn ...)
```

別の方法として、XOPTS または CICS キーワードのプログラム・ソース内の #pragma ステートメントで変換プログラムのオプションを指定することができます。

C および C++ コンパイラー・オプションの指定の詳細については、「*z/OS XL C/C++ ユーザーズ・ガイド*」を参照してください。

すべての変換プログラム・オプションについての説明は、90 ページの『変換プログラムのオプションの定義』を参照してください。

変換プログラム・リストに関連したオプションのような変換プログラム・オプションの多くは、統合 CICS 変換プログラムを使用するときは適用されません。これらのオプションは、指定しても無視されます。EXCI オプションはサポートされていません。CICS オプションを前提とします。

統合 CICS 変換プログラムと合わせて効果的に使用できる変換プログラム・オプションは、以下のとおりです。

- APOST または QUOTE
- CPSM または NOCPSM
- CICS
- DBCS

- DEBUG または NODEBUG
- DLI
- EDF または NOEDF
- FEPI または NOFEPI
- GRAPHIC
- LENGTH または NOLENGTH
- LINKAGE または NOLINKAGE
- NATLANG
- SP
- SYSEIB

変換のプロセス

統合変換プログラムを使用しないコンパイラーの場合、CICS はユーザーが使用するそれぞれの言語ごとに変換プログラムを提供し、EXEC CICS と EXEC DLI の両方のステートメントを処理します。統合変換プログラムを使用するコンパイラーの場合、コンパイラーは CICS と対話して、EXEC CICS と EXEC DLI の両方のステートメントを処理します。

言語変換プログラムは、ユーザー・ソース・プログラムを読み取り、新規のソース・プログラムを作成します。通常の言語ステートメントのほとんどは、変更されずにそのままになっていますが、CICS コマンドは、コーディングに使用している言語に必要な形式の CALL ステートメントに変換されます。呼び出しにより CICS 提供の「EXEC」インターフェース・モジュールが起動されます。これらのインターフェース・モジュールは、後に、ロード・モジュールにリンク・エディットされ、実行時に、要求されたサービスを順番に呼び出します。

変換、コンパイル (アセンブル)、およびリンク・エディットという 3 つのステップがあります。 85 ページの図 10 には、この 3 つのステップが説明されています。

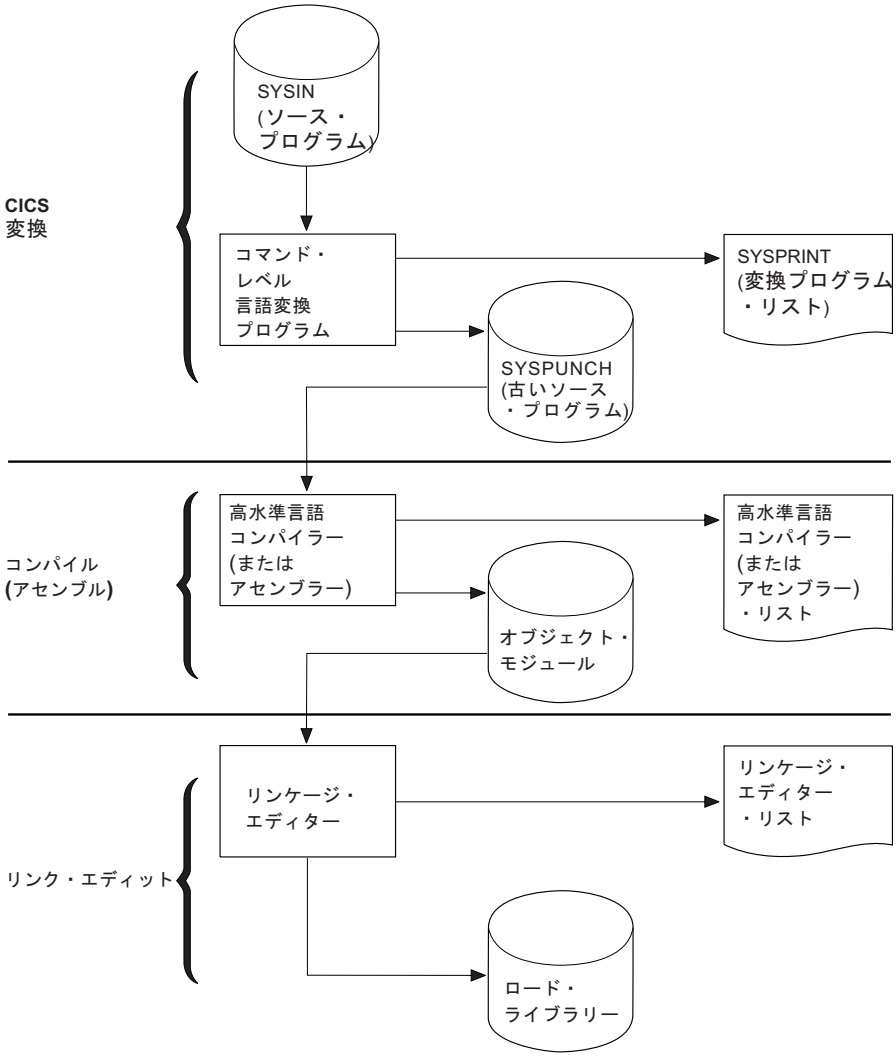


図 10. アプリケーション・プログラムの準備

すべての言語について、変換プログラムは、次のように入力ファイル 1 つと出力ファイル 2 つを使用します。

SYSIN (変換プログラムの入力) は、ソース・プログラムが入っているファイルです。

SYSIN ファイルが固定ブロック・データ・セットとして定義されている場合は、そのデータ・セットが保持できる最大レコード長は 80 バイトです。80 バイト以上のレコード長を持つ固定ブロック・データ・セットを変換プログラムに渡すと、変換プログラムの実行が終了する結果になります。SYSIN ファイルが可変ブロック・データ・セットとして定義されている場合は、そのデータ・セットが保持できる最大レコード長は 100 バイトです。100 バイト以上のレコード長を持つ可変ブロック・データ・セットを変換プログラムに渡すと、変換プログラムはエラーを起こして停止します。

SYSPUNCH

(変換済みソース) は、コンパイル (アセンブル) ステップへの入力となる、ユーザー・ソース・コードの変換されたバージョンです。このファイルでは、ソースは次のように変更されています。

- EXEC インターフェース・ブロック (EIB) 構造の挿入。
- EXEC CICS コマンド、EXEC CPSM コマンドおよび EXEC DLI コマンドの関数呼び出しステートメントへの変更。
- 組み込み関数 CICS DFHRESP、EYUVALUE、および DFHVALUE の処理。
- プログラムに EXEC DLI ステートメントが含まれている場合、データ交換ブロック (DIB) 構造および初期設定呼び出しの挿入。

変換対象の CICS コマンドは、ソースに残されていますが、コメントとしてあるだけです。一般的に、非 CICS ステートメントは未変更です。変換プログラムからの出力は、常に 80 バイトの固定レコード長データ・セットになります。

SYSPRINT

(変換プログラムのリスト) には、変換プログラムが生成したメッセージの数およびすべてのメッセージと関連した重大度コードのうち最高のコードが表示されます。NOOPTIONS オプションで抑制されていない限り、ユーザー・プログラムの変換に使用したオプションも表示されます。

COBOL、C、C++、および PL/I プログラムの場合には、SYSPRINT にメッセージそのものも含まれます。さらに、変換プログラムの SOURCE オプションを指定した場合には、SYSPRINT に注釈付きのソースのリストも含まれます。このリストには、後続のコンパイル・リストとほとんど同じ情報が含まれているので、多くのシステムではこのリストを (NOSOURCE オプションで) 省略します。しかし、コンパイル・リストに存在していなくて、このリストから必要になることがあるのは行番号です (変換プログラムが行番号を割り当てている場合)。実行診断機能 (EDF) を使用してデバッグする場合には、行番号はコード中の場所を示す 1 つの方法になります。VBREF オプションを指定した場合には、ユーザー・プログラム内のコマンドのリストが行番号による相互参照付きで得られるので、これを EDF 用のソース・リストの代わりとして使用することもできます。

アセンブラー言語のプログラムの場合には、SYSPRINT には、変換プログラムのオプション、メッセージ・カウント、および最大重大度コードしか含まれません。メッセージそのものは、SYSPUNCH ファイルの関連ステートメントの後ろにコメントとして挿入されます。これにより、アセンブラーは、ユーザーが検査できるようにメッセージをそのままアセンブラー・リストにコピーします。また、変換プログラムが見つけた問題の結果として、アセンブラーが生成する MNOTE が含まれることもあります。

注: EXEC SQL を使用する場合は、SQL ステートメントを変換してバインドする追加のステップが必要です。この追加ステップについて詳しくは、「*Application Programming and SQL Guide*」を参照してください。

CICS は、サポートするそれぞれの言語ごとに、これらのステップを順に実行するプロシージャーを提供します。 138 ページの『アプリケーション・プログラムをインストールするための CICS 提供プロシージャーの使用』 には、これらのプロシージャーの使用方法、およびその動作について説明されています。

オプションの数を指定すると、変換処理を制御することができます。例えば、EXEC DLI 呼び出しを使用する場合には、変換プログラムに指示する必要があります。

変換プログラムはエラー・メッセージを生成することがあり、このメッセージを検査することは、コンパイラーおよびリンケージ・エディターが生成するメッセージを検査することと同様に重要です。これらのメッセージがどこから出されるかについては、『CICS 提供の変換プログラム』を参照してください。

EXEC コマンドは、CICS インターフェース・モジュールを呼び出す CALL ステートメントに変換されます。これらのモジュールは、リンク・エディット・ステップでオブジェクト・モジュールに組み込まれ、リンク・エディット出力リストに表示されます。これらのモジュールについての詳細な説明は、106 ページの『CICS 提供のインターフェース・モジュール』にあります。

CICS 提供の変換プログラム

以下の CICS 提供の変換プログラムは、CICSTS41.CICS.SDFHLOAD ライブラリーにインストールされています。

アセンブラー DFHEAP1$
C DFHEDP1$
COBOL DFHECP1$
PL/I DFHEPP1$

個別の変換プログラムの動的な起動

このタスクについて

コマンド・レベルの言語変換プログラムは、バッチのアセンブラー言語プログラムから ATTACH、CALL、LINK、または XCTL マクロを使用するか、あるいは PL/I、C、または COBOL プログラムから CALL を使用して、動的に起動できます。ATTACH、LINK、または XCTL を使用する場合、該当する変換プログラムのロード・モジュール DFHExP1$ (アセンブラー言語の場合は x=A、COBOL の場合は x=C、C の場合は x=D、PL/I の場合は x=P です) を使用します。

CALL を使用する場合は、変換プログラムを呼び出す入り口点名として PREPROC を指定します。

すべての場合で、以下のアドレス・パラメーターを変換プログラムに渡します。

- 変換プログラム・オプション・リストのアドレス
- 変換プログラムが使用する DD 名リストのアドレス (これはオプションです)

これらのアドレスは、隣接するフルワードに入れて、フルワードの境界に位置合わせしなければなりません。レジスター 1 は、リストの最初のアドレスを指し、最後のアドレスの高位ビットは 1 にセットされて、リストの終了を示さなければなりません。これは、アドレスが 1 つまたは 2 つの場合どちらにも適用されます。

変換プログラムのオプション・リスト

変換プログラム・オプション・リストは、ハーフワード境界から開始しなければなりません。最初の 2 バイトには、リストのバイト数の 2 進カウントが入っていま

す (カウント・フィールドは除きます)。リストの後の部分には、変換プログラム・オプションのキーワードをどれでも、コンマ、ブランク、またはその両方で区切って入れることができます。

データ定義 (DD 名) リスト

DD 名リストは、ハーフワード境界から開始しなければなりません。最初の 2 バイトには、リストのバイト数の 2 進カウントが入っています (カウント・フィールドは除きます)。リストのそれぞれの項目は、8 バイトのフィールドを占めなければなりません。項目のシーケンスは以下のとおりです。

項目	標準 DD 名	項目	標準 DD 名	項目	標準 DD 名
1	適用されない	3	適用されない	5	SYSIN
2	適用されない	4	適用されない	6	SYSPRINT
				7	SYSPUNCH

適用できる項目を省略すると、変換プログラムは標準 DD 名を使用します。8 バイトより短い DD 名を使用する場合は、フィールドの右側にブランクを埋めてください。1 バイト目に X'FF' を入れると、項目を省略できます。リスト終了の項目はすべて省略することができます。

CICS 変換プログラムの使用

このタスクについて

言語変換プログラムは、ユーザー・ソース・プログラムを読み取り、新規のソース・プログラムを作成します。通常の言語ステートメントのほとんどは、変更されずにそのままになっていますが、CICS コマンドは、コーディングに使用している言語に必要な形式の CALL ステートメントに変換されます。呼び出しにより CICS 提供の「EXEC」インターフェース・モジュールが起動されます。これらのインターフェース・モジュールは、後に、ロード・モジュールにリンク・エディットされ、実行時に、要求されたサービスを順番に呼び出します。

変換プログラム・オプションを指定すると、変換処理を制御することができます。

選択できる変換プログラム・オプションについては、90 ページの『変換プログラムのオプションの定義』にリストがあります。次の 2 つのいずれかの方法で選択項目を指定することができます。

- オプションを指定するために、コンパイラー (アセンブラー) が提供するステートメント上の、XOPTS オプションのサブオプションとしてリストする。これらのステートメントは次の通りです。

  言語　　ステートメント

  COBOL
  　　CBL

  COBOL
  　　PROCESS

C #pragma

C++ #pragma

PL/I * PROCESS

アセンブラー

*ASM または *PROCESS[1]

- 変換ステップの EXEC ジョブ制御ステートメントの PARM オペランドにオプションをリストする。ほとんどのシステムでは、カタログ式プロシージャーを使用して、CICS プログラムの変換、コンパイル (アセンブル)、およびリンクを行うので、そのプロシージャーを呼び出す EXEC ジョブ制御ステートメントの PARM フィールドを指定します。

  例えば、COBOL プログラムの場合のプロシージャー名を DFHYITVL、その中の変換ステップの名前を TRN とすると、COBOL プログラムについて変換プログラム・オプションを設定するには、次のようなステートメントを使用します。

  ```
  //     EXEC DFHEITCL,PARM.TRN=(VBREF,QUOTE,SPACE(2),NOCBLCARD)
  ```

あるオプションをある方式で指定し、それと同じオプションまたはそれと対立するオプションを別の方式で指定した場合、言語ステートメントの指定は EXEC ステートメントの指定を上書きします。同様に、単一オプションに複数の値を指定するか、あるいはいずれかのタイプのステートメントに矛盾するオプションを指定した場合には、最後の設定値が優先されます。COBOL プログラムを除き、これらのステートメントは各ソース・プログラムに先行しなければなりません。複数のプログラムの処理を他の言語でバッチする方法はありません。

変換プログラム・オプションは、1 つ以上のブランクまたは単一のコンマで区切って任意の順序で指定することができます。これらのオプションをオプション用の言語ステートメントに指定する場合には、変換プログラムがその他のオプションを無視し、コンパイラーにそのまま渡すために、XOPTS パラメーターの後に括弧で囲んで指定する必要があります。次の COBOL の例は、変換プログラムおよびコンパイラーの両方のオプションを、一緒に渡す場合を示しています。

```
CBL LIB XOPTS(QUOTE SPACE(2))
```

次の例は、変換プログラムのオプションを単独で渡す場合を示しています。

```
#pragma XOPTS(FLAG(W) SOURCE);
* PROCESS XOPTS(FLAG(W) SOURCE);
*ASM XOPTS(NOPROLOG NOEPILOG)
```

EXEC ジョブ制御ステートメントの PARM オペランドを使用してオプションを指定する場合には、そこで使用できるオプションは変換プログラム・オプションだけなので、XOPTS キーワードは不要です。しかし、関連した括弧を使用するか、あるいは括弧を使用しないで XOPTS を使用することができます。括弧を付けて XOPTS を使用する場合には、必ず、すべての変換プログラム・オプションを括弧で囲んでください。例えば、次の形式は有効です。

```
PARM=(op1 op2 .. opn)
PARM=(XOPTS op1 op2 .. opn)
PARM=XOPTS(op1 op2 .. opn)
```

しかし、次の形式は正しく***ありません***。

```
PARM=(XOPTS(op1 op2) opn)
```

(EXEC DLI バッチ・プログラムを変換している場合を除き、前のリリースとの互換性を保つために、キーワード CICS を XOPTS の代わりに使用することができます。) PARM オペランドを使用して C または C++ #pragma カード処理に関するデフォルト・マージンを変更する場合には、シーケンス・マージンも変更する必要がある点に注意してください。これは、NOSEQUENCE オプションを使用して行うことができます。

注:

1. アセンブラー・プログラムの場合、*ASM ステートメントには変換プログラムのオプションのみが含まれます。このオプションを、アセンブラーはコメントとして処理します。*PROCESS ステートメントには、高水準アセンブラー HLASM 用の、変換プログラム・オプションまたはアセンブラー・オプションを含めることができます。
2. 同じ *PROCESS ステートメント上に、変換プログラム・オプションとアセンブラー・オプションを共存させることはできません。
3. *PROCESS および *ASM ステートメントは先頭に入力する必要があり、アセンブラー・ステートメントはこの両ステートメントの前に表示することはできません。これには、「PRINT ON」や「EJECT」のようなコメントやステートメントも含まれます。*PROCESS ステートメントおよび *ASM ステートメントは、順序を問わず組み込むことができます。
4. 変換プログラム・オプションのみを含む *PROCESS ステートメントには、変換プログラム専用の情報が入っており、このステートメントはアセンブラーに渡されません。
5. *PROCESS ステートメントは、変換プログラムが保存します。アセンブラー・オプションを含む *PROCESS ステートメントは、変換済みプログラムに配置されます。

変換プログラムのオプションの定義

特に指示がない限り、すべての言語に適用される変換プログラム・オプションを指定することができます。すべての変換プログラム・オプション、適用されるプログラム言語、およびすべての有効な省略形が 104 ページの表 5 にリストされています。

ご使用のシステムで、CICS 提供のプロシージャーを配布したままの形式で使用している場合には、デフォルト・オプションが使用されます。以下のオプションの説明では、これらを明示的に取り上げていきます。ご使用のシステムがデフォルトで使用するオプションは、変換ステップからの出力である SYSPRINT (変換プログラムのリスト出力) を調べるとわかります (87 ページの『CICS 提供の変換プログラム』を参照してください)。デフォルトではないオプションが必要な場合には、88 ページの『CICS 変換プログラムの使用』の説明のとおりにオプションを指定する必要があります。

変換プログラム・オプション

92 ページの『APOST』
92 ページの『CBLCARD』

92 ページの『CICS』
92 ページの『COBOL2』
93 ページの『COBOL3』
93 ページの『CPP』
93 ページの『CPSM』
93 ページの『DBCS』
93 ページの『DEBUG』
94 ページの『DLI』
94 ページの『EDF』
94 ページの『EPILOG』
94 ページの『EXCI』
95 ページの『FEPI』
95 ページの『FLAG (I、W、E、または S)』
95 ページの『GDS』
95 ページの『GRAPHIC』
96 ページの『LEASM』
96 ページの『LENGTH』
96 ページの『LINECOUNT(n)』
96 ページの『LINKAGE』
97 ページの『MARGINS(m,n[,c])』
98 ページの『NATLANG(EN または KA)』
98 ページの『NOCBLCARD』
98 ページの『NOCPSM』
98 ページの『NODEBUG』
98 ページの『NOEDF』
98 ページの『NOEPILOG』
99 ページの『NOFEPI』
99 ページの『NOLENGTH』
99 ページの『NOLINKAGE』
99 ページの『NONUM』
99 ページの『NOOPSEQUENCE』
99 ページの『NOOPTIONS』
100 ページの『NOPROLOG』
100 ページの『NOSEQ』
100 ページの『NOSEQUENCE』
100 ページの『NOSOURCE』
100 ページの『NOSPIE』
100 ページの『NOVBREF』
101 ページの『NUM』
101 ページの『OPMARGINS(m,n[,c])』
101 ページの『OPSEQUENCE(m,n)』
101 ページの『OPTIONS』
101 ページの『PROLOG』
102 ページの『QUOTE』
102 ページの『SEQ』
102 ページの『SEQUENCE(m,n)』
102 ページの『SOURCE』
103 ページの『SP』

103 ページの『SPACE(1、2、または 3)』
103 ページの『SPIE』
103 ページの『SYSEIB』
103 ページの『VBREF』

APOST

(COBOL のみ)

APOST は、リテラルをアポストロフィまたは単一引用符で区切ることを指定します。QUOTE は、代わりに、二重引用符を指定します。変換プログラム・ステップとそれに続くコンパイル・ステップには、同一の値を指定しなければなりません。

CICS 提供の COBOL コピーブックは、単一引用符 (APOST) で生成されています。ユーザー・アプリケーションにおいて CICS コンポーネントとのインターフェース用に CICS 提供の任意のコピーブックを使用する場合は、QUOTE オプションではなく APOST を有効にしてください。

CBLCARD

(COBOL のみ) 省略形: CBL

CBLCARD は、変換プログラムで CBL ステートメントを生成することを指定します。これはデフォルトです。変更する場合は NOCBLCARD を指定します。

CICS

CICS は、変換プログラムで EXEC CICS コマンドを処理するよう指定します。これは、変換プログラムのデフォルト指定です。また、CICS は、変換プログラム・オプションを指定する XOPTS キーワードの古い名前でもあるため、CICS オプションを XOPTS リストに含めるか、またはリストの名前の指定で XOPTS の代わりに CICS オプションを使用することによって、CICS オプションを明示的に指定することができます。CICS コマンドがないことを指示する唯一の方法は、オプション CICS を使用しないで XOPTS キーワードを使用する方法です。EXEC DLI コマンドを使用しているバッチ DL/I プログラムでは、これを行わなければなりません。例えば、アセンブラー言語で書かれたバッチ DL/I プログラムを変換するためには、次のように指定します。

```
*ASM XOPTS(DLI)
```

COBOL で書かれ、EXEC API コマンドを含むバッチ・プログラムを変換するためには、次のように指定します。

```
CBL XOPTS(EXCI)
```

COBOL2

(COBOL のみ) 省略形: CO2

COBOL2 は、変換プログラムで、変換済み EXEC ステートメントが使用する一時変数を生成するよう指定します。その他のすべての点では、 93 ページの

『COBOL3』 オプションと同じ方法でプログラムが変換されます。COBOL2 と COBOL3 を同時に指定することはできません。COBOL2 は COBOL のデフォルトです。

注: 異なる方法で COBOL2 および COBOL3 を指定した場合、その 2 つのオプションが指定されている場所に関係なく、*常に* COBOL3 オプションが使用されます。この場合、変換プログラムが警告メッセージを出します。

COBOL3

(COBOL のみ) 省略形: CO3

COBOL3 は、変換プログラムで、言語環境プログラムに準拠したプログラムを変換するよう指定します。COBOL3 と COBOL2 を同時に指定することはできません。11 ページの『第 2 章 言語環境プログラム』 で、言語環境プログラムに準拠した、使用可能なコンパイラーについて説明します。

CPP

(C++ のみ) CPP は、変換プログラムで、IBM C/C++ for MVS などのサポートされている C++ コンパイラーによるコンパイル用に C++ プログラムを変換するよう指定します。

CPSM

CPSM は、変換プログラムで EXEC CPSM コマンドを処理することを指定します。代替値は NOCPSM で、これがデフォルトです。

DBCS

(COBOL のみ)

DBCS は、ソース・プログラムが 2 バイト文字を含んでいる*可能性がある* ことを指定します。このオプションによって、変換プログラムは、16 進数コード X'0E' および X'0F' がプログラム中に現れると、それぞれをシフトアウト (SO) およびシフトイン (SI) コードとして処理するようになります。

DBCS を使用した COBOL でのプログラム作成について詳しくは、「*Enterprise COBOL for z/OS: 言語解説書*」にある DBCS 文字ストリングのセクションを参照してください。

DEBUG

(COBOL、C、C++、および PL/I のみ)

DEBUG は、変換プログラムで、実行診断機能 (EDF) によって使用される行番号を CICS に渡すコードを作成するよう指定します。DEBUG はデフォルトです。変更する場合は NODEBUG を指定します。

DLI

DLI は、変換プログラムで EXEC DLI コマンドを処理することを指定します。この指定は、XOPTS オプション、すなわち XOPTS (DLI) で行わなければなりません。

EDF

EDF は、実行診断機能をプログラムに適用することを指定します。EDF はデフォルトです。変更する場合は NOEDF を指定します。

EPILOG

(アセンブラー言語のみ)

EPILOG® は、変換プログラムで、変換対象プログラムの最後にマクロ DFHEIRET を挿入するよう指定します。DFHEIRET は、発行しているプログラムから、そのプログラムを呼び出したプログラムに制御を返します。RETURN コマンドのオプションのいずれかを使用したい場合には、RETURN を使用し、NOEPILOG を指定する必要があります。

EPILOG はデフォルトです。変換プログラムでマクロ DFHEIRET を挿入しないように変更する場合は、NOEPILOG を指定します。(マクロ DFHEIRET に関するプログラミング情報については、 「*CICS* アプリケーション・プログラミング・リファレンス」を参照してください。

EXCI

EXCI は、変換プログラムで外部 CICS インターフェース (EXCI) に対する EXEC API コマンドを処理するよう指定します。これらのコマンドは、バッチ・プログラムにおいてのみ使用されなければなりません。そのため、EXCI 変換プログラム・オプションは、CICS 変換プログラム・オプション、または CICS オプションを暗黙指定するどの変換プログラム・オプションとも互いに排他的です。CICS および EXCI の両方が指定された場合、または EXCI および、CICS を暗黙指定する変換プログラム・オプションが指定された場合、エラー・メッセージが作成されます。

EXCI オプションは、DLI オプションとも互いに排他的です。外部 CICS インターフェースに対する EXEC API コマンドは、EXEC DLI コマンドを使用しているバッチ・プログラムにはコーディングできません。EXCI および DLI の両方の変換プログラム・コマンドが指定された場合、エラー・メッセージが作成されます。

EXCI は、統合変換プログラムによってコンパイルされた COBOL プログラムには使用できませんが、個別の変換プログラム・ステップでは使用することができます。

EXCI 変換プログラム・オプションは XOPTS、すなわち XOPTS(EXCI) によって指定されます。

FEPI

FEPI は、CICS フロントエンド・プログラミング・インターフェース (FEPI) の FEPI API コマンドへのアクセスを許可します。FEPI は、CICS フロントエンド・プログラミング・インターフェースをインストールしている場合にのみ使用可能です。これを変更する場合は NOFEPI を指定します。

FEPI について詳しくは、「*CICS Front End Programming Interface User's Guide*」を参照してください。

FLAG (I、W、E、または S)

(COBOL、C、C++、および PL/I のみ) 省略形: F

FLAG は、変換におけるエラーのうち、メッセージをリストする必要があるエラーの最小の重大度を指定します。

I すべてのメッセージ。

W (デフォルト) 通知メッセージ以外のすべてのメッセージ。

E 警告メッセージおよび通知メッセージ以外のすべてのメッセージ。

S 重大エラー・メッセージおよびリカバリー不能エラー・メッセージのみ。

GDS

(C、C++、およびアセンブラー言語のみ)

GDS は、変換プログラムで CICS GDS (汎用データ・ストリーム) コマンドを処理することを指定します。上記のコマンドのプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

GRAPHIC

(PL/I のみ)

GRAPHIC は、ソース・プログラムが 2 バイト文字を含んでいる*可能性がある* ことを指定します。このオプションによって、変換プログラムは、16 進数コード X'0E' および X'0F' がプログラム中に現れると、それぞれをシフトアウト (SO) およびシフトイン (SI) コードとして処理するようになります。

またこれは、変換プログラムが、16 進形式のシフトアウトおよびシフトインの値を含むパラメーター・リストを、生成しないようにもします。これらの値がどこに現れても、コンパイラーが受け取るデータ・ストリームに予期しない DBCS 区切りが入らないように、変換プログラムはこれらの値を 2 進形式で表します。

使用しているコンパイラーが DBCS をサポートする場合には、2 バイト文字を使用していない場合でも、予期しないシフトアウト・コードおよびシフトイン・コードが入らないようにする必要があります。これは、変換プログラムがこれらのコードを生成しないように、変換プログラムに GRAPHIC オプションを使用するか、あるいはコンパイラーがこれらのコードを DBCS 区切り文字として解釈しないように、コンパイル・ステップに NOGRAPHIC を指定することで可能になります。

DBCS を使用して PL/I でプログラムを作成する場合の詳細については、関係のある言語の解説書を参照してください。

LEASM

(アセンブラーのみ)

LEASM は、変換プログラムで、言語環境プログラムに準拠したアセンブラーのメインプログラムのコードを生成するよう指定します。

LEASM オプションが指定された場合、CICS 環境におけるアセンブラー・サブルーチンで使用されるマクロ展開を形成するのではなく、
DFHEISTG、DFHEIENT、DFHEIRET、および DFHEIEND の各マクロがさまざまに展開され、言語環境プログラムに準拠したアセンブラーのメインプログラムが作成されます。これにより、NOPROLOG および NOEPILOG を使用し、独自の DFHEIENT およびその他のマクロをコーディングしたユーザー・プログラムにおいて、そのプログラム・ソースを変更することなく、言語環境プログラムのサポートを利用できます。例えば、変換プログラムでは複数のコード基底レジスターがサポートされていないため、複数のコード基底レジスターを必要とするすべてのプログラムがこのカテゴリーに分類できます。

LEASM オプションを使用して変換されたアセンブラー・プログラムの例については、110 ページの『LEASM を使用したアセンブラー言語プログラムの例』を参照してください。

LENGTH

(COBOL、アセンブラー、および PL/I のみ)

LENGTH は、アプリケーション・プログラムの CICS コマンドで LENGTH オプションが省略されている場合に、変換プログラムでデフォルトの長さを生成するよう指示します。これを変更する場合は NOLENGTH を指定します。

LINECOUNT(n)

省略形: LC

LINECOUNT は、変換プログラム・リストの各ページに入れる、ヘッダー行とブランク行を含む行数を指定します。「n」の値は 1 ～ 255 の範囲内の整数にしなければなりません。「n」が 5 より小さい場合には、各ページに含まれるのは、ヘッダー行と 1 行のリストのみです。デフォルトは 60 です。

LINKAGE

(COBOL のみ) 省略形: LIN

LINKAGE は、変換プログラムに、既存の規則に従って、最上位のプログラムの中にある LINKAGE SECTION ステートメントおよび PROCEDURE DIVISION ステートメントを修正することを要求します。

つまり変換プログラムは、PROCEDURE DIVISION 内に USING DFHEIBLK DFHCOMMAREA ステートメントがまだない場合には、それを挿入し、LINKAGE SECTION (必要に応じて作成する) に、DFHEIBLK および DFHCOMMAREA の定義が含まれていることを確認します。

LINKAGE はデフォルトです。変更する場合は NOLINKAGE を指定します。

MARGINS(m,n[,c])

(C、C++、および PL/I のみ) 省略形: MAR

MARGINS は、言語ステートメントまたは CICS ステートメントを含む入力の各行またはレコードの桁を指定します。この限界を超えたデータがソース・リストに含まれていても、変換プログラムはそのデータを処理しません。

また、このオプションによって、SOURCE オプションが指定されている場合に生成されるリストの形式設定を行う、米国標準規格の印刷制御文字の位置を指定することもできます。これを指定しない場合には、入力レコードは、間のブランク行なしでリストされます。マージン・パラメーターは次の通りです。

m 左マージンの桁番号。

n 右マージンの桁番号。この値は m より大きくなければなりません。

注: C または C++ のコンパイラー・オプションとして使用する場合には、MARGIN オプションの 2 番目の引数にアスタリスク (*) が使用可能です。しかし、変換プログラムには、1 ～ 100 の範囲内の数値を指定する必要があります。入力データ・セットが固定長レコードを持つ場合、右マージンのために許容される最大値は 80 です。入力データ・セットが可変長レコードを持つ場合、許容最大値は 100 です。

c 米国標準規格の印刷制御文字の桁番号。これは、m および n に指定する値以外の値にする必要があります。c がゼロの場合は、印刷制御文字がないことを意味します。c が非ゼロの場合には、次の印刷制御文字のみをソースに指定することができます。

(ブランク)
1 行スキップしてから印刷する

0 2 行スキップしてから印刷する

- 3 行スキップしてから印刷する

+ スキップせずに印刷する

1 改ページ

C および C++ のデフォルトは、固定長レコードの場合は MARGINS(1,72,0) であり、可変長レコードの場合はレコード長と同じ (1,レコード長,0) です。PL/I のデフォルトは、固定長レコードの場合は MARGINS(2、72、0) で、可変長レコードの場合は MARGINS(10、100、0) です。

NATLANG(EN または KA)

NATLANG は、変換プログラムのメッセージ出力のために、どの言語を使用するかを指定します。

EN (デフォルト) 英語

KA 漢字

(このオプションと NATLANG API オプションとを混同しないように注意してください。)

NOCBLCARD

(COBOL のみ)

NOCBLCARD は、変換プログラムが CBL ステートメントを生成しないことを指定します。CICS で必要なコンパイラー・オプションは、DFHYITVL プロシージャーが指定します。RENT、NODYNAM、および LIB は、必ず指定してください。

NOCPSM

NOCPSM は、変換プログラムが EXEC CPSM コマンドを処理しないことを指定します。これはデフォルトです。変更する場合は CPSM を指定します。

NODEBUG

(COBOL、C、C++、および PL/I のみ)

NODEBUG は、変換プログラムで、実行診断機能 (EDF) によって使用される行番号を CICS に渡すコードを作成しないよう指定します。

NOEDF

NOEDF は、実行診断機能をプログラムに適用しないことを指定します。NOEDF を指定することによるパフォーマンス上の利点はありませんが、このオプションは、十分にデバッグ済みのサブプログラムの中のコマンドが EDF 表示に現れないようにするために有用な場合があります。

NOEPILOG

(アセンブラー言語のみ)

NOEPILOG は、変換プログラムが、変換中のプログラムの最後にマクロ DFHEIRET を挿入しないことを指定します。DFHEIRET は、発行しているプログラムから、そのプログラムを呼び出したプログラムに制御を返します。EXEC CICS RETURN コマンドの任意のオプションを使用する場合には、EXEC CICS RETURN を使用し、NOEPILOG を指定する必要があります。NOEPILOG は、変換プログラムがマクロ DFHEIRET を挿入しないようにします。デフォルトは EPILOG です。

NOFEPI

NOFEPI は、CICS フロントエンド・プログラミング・インターフェース (FEPI) の FEPI API コマンドへのアクセスを許可しません。NOFEPI はデフォルトです。変更する場合は FEPI を指定します。

NOLENGTH

(COBOL、アセンブラー、および PL/I のみ)

NOLENGTH は、アプリケーション・プログラムの CICS コマンドで LENGTH オプションが省略されている場合に、変換プログラムでデフォルトの長さを生成しないよう指示します。デフォルトは LENGTH です。

NOLINKAGE

(COBOL のみ)

NOLINKAGE は、欠落している DFHEIBLK ステートメントおよび DFHCOMMAREA ステートメントを提供したり、あるいは LINKAGE SECTION に EIB 構造の定義を挿入するために変換プログラムが LINKAGE SECTION ステートメント、および PROCEDURE DIVISION ステートメントを修正しないことを要求します。

つまりユーザーは、COMMAREA を定義して EXEC CICS ADDRESS コマンドを使用する COBOL コピーブックを提供できます。

LINKAGE がデフォルトです。

NONUM

(COBOL のみ)

NONUM は、変換プログラムが、プログラムの各行の 1 ～ 6 桁目に入っている行番号を、診断メッセージおよび相互参照リストの中での行番号として使用しないで、独自の行番号を生成することを指定します。NONUM はデフォルトです。変更する場合は NUM を指定します。

NOOPSEQUENCE

(C、C++、および PL/I のみ) 省略形: NOS

NOOPSEQUENCE は、変換プログラムの出力レコード内のシーケンス・フィールドの位置を指定します。C および C++ のデフォルトは、固定長レコードの場合は OPSEQUENCE(73,80) であり、可変長レコードの場合は NOOPSEQUENCE です。PL/I のデフォルトは、両方のタイプのレコードについて OPSEQUENCE(73,80) です。

NOOPTIONS

省略形: NOP

NOOPTIONS は、変換プログラムが、変換時に使用するオプションのリストを出力リストに含めないことを指定します。

NOPROLOG

(アセンブラー言語のみ)

NOPROLOG は、変換プログラムが、アセンブル中のプログラムにマクロ DFHEISTG、DFHEIEND、および DFHEIENT を挿入しないことを指定します。これらのマクロは、ローカル・プログラム・ストレージを定義し、プログラムの入り口で実行します。

NOSEQ

(COBOL のみ)

NOSEQ は、変換プログラムがソース・ステートメントのシーケンス・フィールド 1 ～ 6 桁目を検査しないことを指定します。代替値の SEQ が、デフォルトです。SEQ を指定し、ステートメントがシーケンス通りになっていない場合には、そのステートメントにフラグが付けられます。

NOSEQUENCE

(C、C++、および PL/I のみ) 省略形: NSEQ

NOSEQUENCE は、変換プログラムの入力のステートメントにシーケンス番号が付いていないこと、および変換プログラムが独自の行番号を割り当てなければならないことを指定します。

固定長レコードの場合のデフォルトは SEQUENCE(73,80) です。C および C++ における可変長レコードの場合のデフォルトは NOSEQUENCE であり、PL/I における可変長レコードの場合のデフォルトは SEQUENCE(1,8) です。

NOSOURCE

NOSOURCE は、変換プログラムが、変換されたソース・プログラムのリストを変換プログラム・リストに含めないことを指定します。

NOSPIE

NOSPIE は、変換プログラムがリカバリー不能エラーをトラップしないようにします。その代わりに、ダンプを生成します。NOSPIE を使用する必要があるのは、IBM サポート・センターによる指示があった場合のみです。

NOVBREF

(COBOL、C、C++ および PL/I のみ)

NOVBREF は、変換プログラムが、行番号によるコマンドの相互参照を変換プログラム・リストに含めないことを指定します (NOVBREF は NOXREF と呼ばれていました。互換性のために、NOXREF はまだ受け入れられます)。NOVBREF はデフォルトです。変更する場合は VBREF を指定します。

NUM

(COBOL のみ)

NUM は、変換プログラムが、プログラムの各行の 1 ～ 6 桁目に入っている行番号を、診断メッセージおよび相互参照リストの中での行番号として使用することを指定します。これを変更する場合は NONUM を指定します。NONUM がデフォルトです。

OPMARGINS(m,n[,c])

(C、C++ および PL/I のみ) 省略形: OM

OPMARGINS は、変換プログラム出力マージン、すなわち、その後のコンパイラーへの入力のマージンを指定します。通常、これらのマージンは変換プログラムの入力マージンと同じです。入力マージンの定義、および「m」、「n」、および「c」の意味については、MARGINS を参照してください。C および C++ のデフォルトは OPMARGINS(1,72,0) であり、PL/I の場合のデフォルトは OPMARGINS(2,72,0) です。

OPMARGINS オプションで許容されている最大の「n」値は 80 です。変換プログラムからの出力は常に固定長レコード形式です。

OPMARGINS オプションが、変換プログラムからの出力をある形式に設定するために使用される場合は、使用されるコンパイラーの入力マージンを変更しなければならないことがあります。OPMARGINS 値がデフォルトである場合は、これは必要ではありません。

OPSEQUENCE(m,n)

(C、C++、および PL/I のみ) 省略形: OS

OPSEQUENCE は、変換プログラムの出力レコード内のシーケンス・フィールドの位置を指定します。「m」および「n」の意味については、SEQUENCE を参照してください。C および C++ のデフォルトは、固定長レコードの場合は OPSEQUENCE(73,80) であり、可変長レコードの場合は NOOPSEQUENCE です。PL/I のデフォルトは、両方のタイプのレコードについて OPSEQUENCE(73,80) です。

OPTIONS

省略形: OP

OPTIONS は、変換プログラムが、変換時に使用するオプションのリストを出力リストに含めることを指定します。

PROLOG

(アセンブラー言語のみ)

PROLOG は、変換プログラムが、アセンブル中のプログラムにマクロ DFHEISTG、DFHEIEND、および DFHEIENT を挿入することを指定します。これらのマクロは、ローカル・プログラム・ストレージを定義し、プログラムの入り口で実行します。

QUOTE

(COBOL のみ) 省略形: Q

QUOTE は、リテラルを二重引用符 (") で区切るよう指定します。変換プログラム・ステップとそれに続くコンパイラー・ステップには、同一の値を指定しなければなりません。

CICS 提供の COBOL コピーブックは、単一引用符 (APOST) で生成されています。ユーザー・アプリケーションにおいて CICS コンポーネントとのインターフェース用に CICS 提供の任意のコピーブックを使用する場合は、QUOTE オプションではなく APOST を有効にしてください。

SEQ

(COBOL のみ)

SEQ は、変換プログラムがソース・ステートメントのシーケンス・フィールド 1 ～ 6 桁目を検査することを指定します。SEQ はデフォルトです。変更する場合は NOSEQ を指定します。ステートメントがシーケンスになっていない場合は、フラグが付けられます。

SEQUENCE(m,n)

(C、C++、および PL/I のみ) 省略形: SEQ

SEQUENCE は、変換プログラムの入力のステートメントにシーケンス番号が付けられていることを指定し、シーケンス・フィールドを含む各行またはレコードの桁を指定します。変換プログラムは、エラー・メッセージおよび相互参照リストでの行番号として、この番号を使用します。入力行または入力レコードをシーケンス通りにソートする試みは行いません。シーケンス・フィールドが指定されていない場合には、変換プログラムは独自の行番号を割り当てます。SEQUENCE パラメーターは次の通りです。

m シーケンス番号の左端の桁

n シーケンス番号の右端の桁

シーケンス番号フィールドは 8 文字以下でなければならず、(MARGINS オプションに指定されている) ソース・プログラムにオーバーラップしてはなりません。

固定長レコードの場合のデフォルトは SEQUENCE(73,80) です。C および C++ における可変長レコードの場合のデフォルトは NOSEQUENCE であり、PL/I における可変長レコードの場合のデフォルトは SEQUENCE(1,8) です。

SOURCE

省略形: S

SOURCE は、変換プログラムが、変換されたソース・プログラムのリストを変換プログラム・リストに含めることを指定します。SOURCE はデフォルトです。変更する場合は NOSOURCE を指定します。

SP

SP は、特別 (SP) な CICS コマンドが含まれるアプリケーション・プログラムに対して指定する必要があり、そうしない場合は変換時にプログラムがリジェクトされてしまいます。特別な CICS コマンドとは、ACQUIRE、COLLECT、CREATE、DISABLE、DISCARD、ENABLE、EXTRACT、INQUIRE、PERFORM、 RESYNC、および SET です。これらは通常、システム・プログラマーによって使用されます。上記のコマンドのプログラミング情報については、「*CICS* アプリケーション・プログラミング・リファレンス」のシステム・コマンドを参照してください。

SPACE(1、2、または 3)

(COBOL のみ)

SPACE は、出力リストで使用する行送りのタイプを指定します。SPACE (1) と指定すると 1 行送り、SPACE (2) では 2 行送り、SPACE (3) では 3 行送りとなります。SPACE (3) がデフォルトです。

SPIE

SPIE は、変換プログラムがリカバリー不能エラーをトラップすることを指定します。SPIE はデフォルトです。変更する場合は NOSPIE を指定します。

SYSEIB

SYSEIB は、プログラムが、アプリケーション EIB の代わりにシステム EIB を使用することを指定します。SYSEIB オプションにより、プログラムはアプリケーション EIB を更新することなく、実行の様子をアプリケーションに対して明白にして CICS コマンドを実行できます。しかし、このオプションは、このオプションを使用するプログラムに対して制約事項を課すので、特殊な状態の場合にだけ使用すべきです。SYSEIB オプションを使用して変換したプログラムでは、次のことを行う必要があります。

- AMODE(31) での実行 (システム EIB が「TASKDATALOC(ANY)」ストレージに配置されていることが前提であるため)。
- ADDRESS EIB コマンドを使用したシステム EIB のアドレスの入手 (プログラムが SYSEIB オプションで変換される場合、このコマンドは自動的にシステム EIB のアドレスを返します)。
- SYSEIB オプションの使用は、当該プログラムによって発行されるすべての CICS コマンドで、NOHANDLE オプションの使用を暗黙指定する点に留意する。(コマンドでは、必要に応じて RESP オプションを使用する必要があります)。

VBREF

(COBOL、C、C++、および PL/I のみ)

VBREF は、変換プログラムが、行番号によるコマンドの相互参照を変換リストに含めるかどうかを指定します (VBREF は、XREF と呼ばれていたもので、今でも受け入れられます)。

変換プログラムのオプション・テーブル

表 5. プログラミング言語に適用可能な変換プログラム・オプション

変換プログラム・オプション	**COBOL**	**C**	**C++**	**PL/I**	**アセンブラー**
APOST または QUOTE	X				
CBLCARD または NOCBLCARD	X				
CICS	X	X	X	X	X
COBOL2	X				
COBOL3	X				
CPP			X		
CPSM または **NOCPSM**	X	X	X	X	X
DBCS	X				
DEBUG または NODEBUG	X	X	X	X	
DLI	X	X	X	X	X
EDF または NOEDF	X	X	X	X	X
EPILOG または NOEPILOG					X
EXCI	X	X	X	X	X
FEPI または **NOFEPI**	X	X	X	X	X
FLAG (I または **W** あるいは E または S)	X	X	X	X	
GDS		X	X		X
GRAPHIC				X	
LEASM					X
LENGTH または NOLENGTH	X			X	X
LINECOUNT(n)	X	X	X	X	X
LINKAGE または NOLINKAGE	X				

表 5. プログラミング言語に適用可能な変換プログラム・オプション (続き)

変換プログラム・オプション	COBOL	C	C++	PL/I	アセンブラー
MARGINS(m,n)		X	X	X	
NATLANG	X	X	X	X	X
NUM または **NONUM**	X				
OPMARGINS(m,n[,c])		X	X	X	
OPSEQUENCE(m,n) または NOOPSEQUENCE		X	X	X	
OPTIONS または NOOPTIONS	X	X	X	X	X
PROLOG または NOPROLOG					X
QUOTE または **APOST**	X				
SEQ または NOSEQ	X				
SEQUENCE(m,n) または NOSEQUENCE		X	X	X	
SOURCE または NOSOURCE		X	X	X	
SP	X	X	X	X	X
SPACE(1、2、または **3**)	X				
SPIE または NOSPIE	X	X	X	X	X
SYSEIB	X	X	X	X	X
VBREF または **NOVBREF**	X	X	X	X	

COPY ステートメントの使用

コンパイラー (またはアセンブラー) は、入力として、ユーザー・プログラムの元のソースではなく、変換済みのバージョンを読み取ります。これにより、コンパイラー (アセンブラー) リストに表示される内容が影響を受けます。これはつまり、ユーザーのソース・コードの COPY ステートメントに、未変換の CICS コマンドが含まれていてはならないということでもあります。変換プログラムで未変換の CICS コマンドを変換するには、時間がかかりすぎるためです。

このタスクについて

異なる変換プログラムを使用しており、サンプル集のソースに CICS コマンドが含まれている場合には、そのサンプル集が組み込まれるプログラムを変換およびコンパイルする前に、ソースを別に変換しておく必要があります。統合 CICS 変換プロ

グラムを使用しており、サンプル集のソースに CICS コマンドが含まれている場合は、そのサンプル集が組み込まれるプログラムをコンパイルする前に、サンプル集を別に変換しておく必要はありません。

外部プログラムは、組み込まれるサンプル集内にすべての CICS コマンドがあったとしても、常に CICS 変換プログラムを通じて渡されるか、または統合 CICS 変換プログラムを持つコンパイラーを使用してコンパイルする必要があります。

CICS 提供のインターフェース・モジュール

CICS の下で実行されるユーザー・アプリケーション・プログラムには、以下の機能を使用するために、それぞれ 1 つ以上のインターフェース・モジュール (**スタブ**とも呼ばれる) が必要です。

- EXEC インターフェース
- CPI コミュニケーション機能
- SAA® リソース・リカバリー機能
- CICSPlex® SM アプリケーション・プログラミング・インターフェース (CICSPlex SM スタブについて詳しくは、「*CICSPlex SM アプリケーション・プログラミング・ガイド*」を参照してください)。

EXEC インターフェース・モジュール

CICS アプリケーション・プログラムには、それぞれ CICS へのインターフェースが組み込まれている必要があります。これは、CICS 高水準プログラミング・インターフェースで使用される、EXEC インターフェース・モジュールの形式になっています。 CICSTS41.CICS.SDFHLOAD ライブラリーにインストールされるモジュールは、ユーザー・コードと EXEC インターフェース・プログラム DFHEIP との間の通信を提供するために、ユーザー・アプリケーション・プログラムとリンク・エディットしなければなりません。

CPI コミュニケーション・インターフェース・モジュール

共通プログラミング・インターフェース・コミュニケーション (CPI コミュニケーション) を使用する CICS アプリケーション・プログラムには、それぞれ CPI コミュニケーションへのインターフェースが組み込まれている必要があります。これは、CICS 高水準プログラミング・インターフェースで使用されるインターフェース・モジュールの形式になっていて、すべてのプログラム言語に対して共通です。CICSTS41.CICS.SDFHLOAD ライブラリーにインストールされているモジュール DFHCPLC は、CPI コミュニケーションを使用するそれぞれのアプリケーション・プログラムとリンク・エディットしなければなりません。

SAA リソース・リカバリー・インターフェース・モジュール

SAA リソース・リカバリーを使用する CICS アプリケーション・プログラムには、それぞれ SAA リソース・リカバリーへのインターフェースが組み込まれている必要があります。これは、CICS 高水準プログラミング・インターフェースで使用されるインターフェース・モジュールの形式になっていて、すべてのプログラム言語に

対して共通です。CICSTS41.CICS.SDFHLOAD ライブラリーにインストールされるモジュール DFHCPLRR は、SAA リソース・リカバリー機能を使用するそれぞれのアプリケーション・プログラムにリンク・エディットする必要があります。

EXEC インターフェース・モジュールの使用

言語変換プログラムは、ユーザー・ソース・プログラムを読み取り、新規のソース・プログラムを作成します。通常の言語ステートメントは、変更されずそのままの状態になっていますが、CICS コマンドは、コーディングに使用している言語に必要な形式の CALL ステートメントに変換されます。各呼び出しでは、CICS 提供の「EXEC」インターフェース・モジュール、または CICS の高水準プログラミング・インターフェースによって使用される機能依存部分のコードである**スタブ**が呼び出されます。SDFHLOAD ライブラリーで提供されるこのスタブは、アプリケーション・プログラムにリンク・エディットし、ユーザー・コードと CICS EXEC インターフェース・プログラムである DFHEIP との間の通信を提供する必要があります。これらのスタブは、EXEC CICS コマンドおよび EXEC DLI コマンドの実行時に呼び出されます。

それぞれのプログラム言語ごとにスタブがあります。

表 6. インターフェース・モジュール

言語	インターフェース・モジュール名
アセンブラー	DFHELII および DFHEAI0
LEASM オプションを使用するすべての HLL 言語およびアセンブラーの MAIN プログラム	DFHELII

CICS 提供のスタブ・ルーチンは、内部プログラミング・インターフェース (CICS コマンド・レベルのインターフェース) で動作します。このインターフェースは、互換性のない方法で変更されることはありません。その結果、これらのスタブ・モジュールは上位および下位の両方に互換性があるため、これらの最新レベルのスタブのいずれかを組み込むために CICS アプリケーション・モジュールを再リンクする必要はありません。

DFHEAI0 を除き、すべてのスタブには、EXEC CICS コマンドから必要な CICS サービスへのリンケージを提供するという同じ機能があります。スタブは、変換された EXEC CICS コマンドから呼び出されたさまざまなエントリー・ポイントを提供し、次に CICS の EXEC インターフェース機能に制御を渡す一連の命令を実行することによって、これを実現します。

DFHELII には複数のエントリー・ポイントが含まれており、そのほとんどには、CICS PL/I 変換プログラムの非常に古いバージョンに対する互換性があります。DFHELII には、エントリー DFHEXEC (C および C++ アプリケーション・プログラム用)、DFHEI1 (COBOL およびアセンブラー用)、および DFHEI01 (PL/I 用) が含まれています。

これらのスタブのそれぞれは、8 バイトの DFHYxnnn 形式の目印で始まります。x は、そのスタブがサポートする言語を示します (例えば、A はアセンブラーを示し、I はそのスタブが言語非依存であることを示す)。nnn は、そのスタブの組み込

み元の CICS のリリースを表します。Y は、そのスタブが読み取り専用であることを示します。非常に古いリリースの CICS で提供されていたスタブには、DFHExxxx 形式の目印が付いていました。E の文字は、そのスタブが読み取り専用ではないことを表します。CICS Transaction Server for z/OS バージョン 4 リリース 1 における DFHELII の目印は、DFHYI660 です。

目印は、最後に CICS アプリケーション・ロード・モジュールがリンクされた CICS リリースの判別が必要な場合に便利です。

COBOL

それぞれの EXEC コマンドは、入り口 DFHEI1 を参照する COBOL の CALL ステートメントに変換されます。

以下の例は、変換プログラムが単純な EXEC CICS RETURN コマンドを処理する場合に生成する出力を示しています。

```
*EXEC CICS RETURN END-EXEC
      Call 'DFHEI1' using by content x'0e0800006000001000'
     end-call.
```

DFHEI1 への参照は、DFHYITVL または DFHZITCL などの CICS 提供プロシージャーのリンケージ・エディターのステップに、DFHELII スタブ・ルーチンを組み込むことで解決されます。

PL/I

PL/I プログラムを変換するとき、各 EXEC コマンドは、入り口点 DFHEI01 への呼び出しを生成します。この呼び出しは、入り口 DFHEI01 に関連付けられた可変入り口点 DFHEI0 を使用して実行されます。変換プログラムは、以下のステートメントを、それぞれの変換されるプログラムの始まりの付近に挿入して、呼び出しを使用可能にします。

```
DCL DFHEI0 ENTRY VARIABLE INIT(DFHEI01) AUTO;
DCL DFHEI01 ENTRY OPTIONS(INTER ASSEMBLER);
```

変換プログラムは、正常に変換された EXEC コマンドごとに、固有の入り口名を DFHEI0 に基づいて作成します。以下の例は、変換プログラムが単純な EXEC CICS RETURN コマンドを処理する場合に生成する出力を示しています。

```
/* EXEC CICS RETURN TRANSID(NEXT) */
DO;
DCL DFHENTRY_B62D3C38_296F2687 BASED(ADDR(DFHEI0)) OPTIONS(INTER ASSEM
BLER) ENTRY(*,CHAR(4));
CALL DFHENTRY_B62D3C38_296F2687('xxxxxxxxxxxxxxxxx' /* '0E 08 80 00 03
00 00 10 00 F0 F0 F0 F0 F0 F0 F1 F0 'X */, NEXT);
END;
```

上記の例で、DFHENTRY_B62D3C38_296F2687 は、実際の入り口 DFHEI01 に関連付けられた入り口変数 DFHEI0 に基づいています。この手法により、変換プログラムが、それぞれの可変入り口名ごとに PL/I データ記述子リストを作成することが可能です。それから PL/I コンパイラーは、EXEC コマンドで参照される変数名が、データ記述子リストで変換プログラムが定義した属性と一貫性のある属性を持つよ

うに定義されているかどうかを検査します。この例で、ENTRY(*,CHAR(4)) は、TRANSID オプションに関連した変数 (名前は NEXT) が長さ 4 バイトの文字ストリングになるように指定しています。

DFHEI01 への参照は、DFHYITPL のような CICS 提供プロシージャーの 1 つに対するリンケージ・エディターのステップに、DFHELII スタブ・ルーチンを組み込むことで解決されます。

C および C++

C および C++ プログラムでは、それぞれの EXEC CICS コマンドが、コマンド変換プログラムによって DFHEXEC 関数呼び出しに変換されます。

変換プログラムは、以下のステートメントを、それぞれの変換されるプログラムの始まりの付近に挿入して、呼び出しを使用可能にします。

```
#pragma linkage(DFHEXEC,OS)    /* force OS linkage  */
void DFHEXEC();                /* function to call CICS */
```

以下の例は、変換プログラムが単純な EXEC CICS RETURN コマンドを処理する場合に生成する出力を示しています。

```
/* EXEC CICS RETURN */
{
DFHEXEC( "\x0E\x08\x00\x2F\x00\x00\x10\x00\xF0\xF0\xF0\xF0\xF1\xF8\xF0\xF0");
}
```

DFHEXEC への参照は、DFHYITDL、DFHZITDL、DFHZITEL、DFHZITFL、または DFHZITGL のような CICS 提供プロシージャーの 1 つに対するリンケージ・エディターのステップに、DFHELII スタブ・ルーチンを組み込むことで解決されます。

アセンブラー言語

それぞれの EXEC コマンドは、DFHECALL の呼び出しに変換されます。マクロ

以下の例は、変換プログラムが単純な EXEC CICS RETURN コマンドを処理する場合に生成する出力を示しています。

```
*        EXEC CICS RETURN
         DFHECALL =X'0E0800000800001000'
```

上記の DFHECALL マクロ呼び出しのアセンブリーは、レジスター 1 によってアドレッシングされるパラメーター・リストを作成し、入り口 DFHEI1 のアドレスをレジスター 15 にロードし、さらに BALR 命令を発行してスタブ・ルーチンを呼び出すコードを生成します。

```
         DS   0H
         LA   1,DFHEITPL
         LA   14,=x'0E08000008001000'
         ST   14,0(,1)
         OI   0(1),x'80'
         L    15,=V(DFHEI1)
         BALR 14,15
```

DFHEI1 への参照は、DFHEITAL のような CICS 提供プロシージャーの 1 つに対するリンケージ・エディターのステップに、DFHEAI スタブ・ルーチンを組み込むことで解決されます。CICS Transaction Server for z/OS バージョン 4 リリース 1

の DFHEAI の目印は、DFHYA660 と、このスタブが CICS Transaction Server for z/OS バージョン 4 リリース 1 で提供されたことを示すリリース番号です。

アセンブラー・アプリケーション・プログラム用の DFHEAI0 スタブは、リンケージ・エディターまたはバインド・ユーティリティーの自動呼び出し機能によって組み込まれます。このスタブは、アセンブラー・アプリケーション・プログラムの動的ストレージ域を獲得/解放する都度、DFHEIENT および DFHEIRET マクロが生成したコードによって呼び出されます。このスタブはアセンブラー・アプリケーション・プログラムでのみ必要です。高水準言語で書かれたプログラムの場合は、同等の機能をもつスタブは不要であり、提供されていません。

LEASM を使用したアセンブラー言語プログラムの例

図 11 には、簡単な CICS アセンブラー・プログラムが示されています。

```
*ASM XOPTS(LEASM)
DFHEISTG DSECT
OUTAREA  DS    CL200              DATA OUTPUT AREA
*
EIASM    CSECT ,
         MVC  OUTAREA(40),MSG1
         MVC  OUTAREA(4),EIBTRMID
         EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43) FREEKB ERASE
         EXEC CICS RECEIVE
         MVC  OUTAREA(13),MSG2
         EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
         EXEC CICS RETURN
*
MSG1     DC   C'xxxx: ASM program invoked. ENTER TO END.'
MSG2     DC   C'PROGRAM ENDED'
         END
```

図 11. 簡単な CICS アセンブラー・プログラム。

変換され、アセンブルされると、次の 118 ページの図 12 のように展開されます。

```
ASM XOPTS(LEASM)
         DFHEIGBL ,,,LE            INSERTED BY TRANSLATOR
*,&DFHEIDL; SETB 0  1 MEANS EXEC DLI IN PROGRAM                          01-DFHEI
*,&DFHEIDB; SETB 0  1 MEANS BATCH PROGRAM                                01-DFHEI
*,&DFHEIRS; SETB 0  1 MEANS RSECT                                        01-DFHEI
*,&DFHEILE; SETB 1  1 MEANS LE MAIN                                      01-DFHEI
DFHEISTG DSECT
         DFHEISTG                  INSERTED BY TRANSLATOR
*********************************************************************
*             EXEC INTERFACE DYNAMIC STORAGE                        *
*********************************************************************
DFHEISTG DSECT                     EXEC INTERFACE STORAGE      @BBAC81A 01-DFHEI
         USING *,DFHEIPLR          ESTABLISH ADDRESSABILITY    @BBAC81A 01-DFHEI
*
*********************************************************************
*   D Y N A M I C   S T O R A G E   A R E A    ( D S A )            *
*********************************************************************
*
CEEDSA   DS    0D         Just keep the same label for formulae         02-CEEDS
*
CEEDSAFLAGS DS XL2                 DSA flags                            02-CEEDS
CEEDSALNGC EQU X'1000'               C library DSA                      02-CEEDS
CEEDSALNGP EQU X'0800'               PL/I library DSA                   02-CEEDS
CEEDSAEXIT EQU X'0008'               An Exit DSA                        02-CEEDS
CEEDSAMEMD DS  XL2                 Member defined                       02-CEEDS
CEEDSABKC DS   A                   Addr of DSA of caller                02-CEEDS
CEEDSAFWC DS   A                   Addr of DSA of last called rtn       02-CEEDS
CEEDSAR14 DS   F                   Save area for register 14            02-CEEDS
CEEDSAR15 DS   F                   Save area for register 15            02-CEEDS
CEEDSAR0 DS    F                   Save area for register 0             02-CEEDS
CEEDSAR1 DS    F                   Save area for register 1             02-CEEDS
CEEDSAR2 DS    F                   Save area for register 2             02-CEEDS
CEEDSAR3 DS    F                   Save area for register 3             02-CEEDS
CEEDSAR4 DS    F                   Save area for register 4             02-CEEDS
CEEDSAR5 DS    F                   Save area for register 5             02-CEEDS
CEEDSAR6 DS    F                   Save area for register 6             02-CEEDS
CEEDSAR7 DS    F                   Save area for register 7             02-CEEDS
CEEDSAR8 DS    F                   Save area for register 8             02-CEEDS
CEEDSAR9 DS    F                   Save area for register 9             02-CEEDS
CEEDSAR10 DS   F                   Save area for register 10            02-CEEDS
CEEDSAR11 DS   F                   Save area for register 11            02-CEEDS
CEEDSAR12 DS   F                   Save area for register 12            02-CEEDS
CEEDSALWS DS   A                   Addr of PL/I Language Working Space  02-CEEDS
CEEDSANAB DS   A                   Addr of next available byte          02-CEEDS
CEEDSAPNAB DS  A                   Addr of end-of-prolog NAB            02-CEEDS
          DS   4F                                                       02-CEEDS
CEEDSATRAN DS  0A                  HPL TxArea or                        02-CEEDS
CEEDSARENT DS  A                   Program reentry address-IPAT         02-CEEDS
CEEDSACILC DS  A                   C to Fortran ILC save area           02-CEEDS
CEEDSAMODE DS  A                   Return address of module that        02-CEEDS
*                                  caused the last mode switch
          DS   2F                                                       02-CEEDS
CEEDSARMR  DS  A                   Addr of language specific            02-CEEDS
*                                  exception handler
*
          DS   F                   Reserved                             02-CEEDS
CEEDSAAUTO DS  0D                  Automatic storage starts here        02-CEEDS
CEEDSAEND  DS  0D                  End of DSA                           02-CEEDS
```

```
CEEDSASZ EQU   CEEDSAEND-CEEDSA     Size of DSA                          02-CEEDS
CEEDSA_STDCEEDSA  EQU   X'0000'     flag values of standard CEE DSA      02-CEEDS
*
*
*
DFHEISA  DS    18F                 SAVE AREA R14-R12 AT 12 OFF @BBAC81A 01-DFHEI
DFHEILWS DS    F                   RESERVED                    @BBAC81A 01-DFHEI
DFHEINAB DS    F                   RESERVED                    @BBAC81A 01-DFHEI
DFHEIRS0 DS    F                   RESERVED                    @BBAC81A 01-DFHEI
DFHEIR13 DS    F                   REGISTER 13                 @BBAC81A 01-DFHEI
DFHEIRS1 DS    F                   RESERVED                    @BBAC81A 01-DFHEI
DFHEIBP  DS    F                   EIB POINTER      (NOT USED IF BATCH) 01-DFHEI
DFHEICAP DS    F                   COMMAREA POINTER (NOT USED IF BATCH) 01-DFHEI
DFHEIV00 DS    H                   HALFWORD TEMP USED BY DFHECALL       01-DFHEI
DFHEIRS2 DS    H                   RESERVED                    @BBAC81A 01-DFHEI
DFHEIPL  DS    13F                 PARAMETER LIST                  @05C 01-DFHEI
         DS    51F                 ALLOW 64 PARAMETERS FOR DLI     @L2A 01-DFHEI
*                                AND IN XA2 ON, FOR EXEC CICS ALSO
DFHEIRS3 DS    F                   RESERVED                        @L2A 01-DFHEI
DFHEIRS4 DS    F                   RESERVED                        @L2A 01-DFHEI
DFHEITP1 DS    F                   TEMPORARY POINTER 1             @L2A 01-DFHEI
DFHEITP2 DS    F                   TEMPORARY POINTER 2             @L2A 01-DFHEI
DFHEITP3 DS    F                   TEMPORARY POINTER 3             @L2A 01-DFHEI
DFHEITP4 DS    F                   TEMPORARY POINTER 4             @L2A 01-DFHEI
*********************************************************************
*             START DEFINITION OF USER DYNAMIC STORAGE              *
*********************************************************************
DFHEIUSR DS    0D                  ALIGN USER DYNAMIC STORAGE  @BBAC81A 01-DFHEI
*
OUTAREA  DS    CL200               DATA OUTPUT AREA
*
TESTLE   CSECT ,
         DFHEIENT                  INSERTED BY TRANSLATOR
*********************************************************************
*                                                                   *
* CONTROL BLOCK NAME = DFHEIBLK                                     *
*                                                                   *
* NAME OF MATCHING PL/AS CONTROL BLOCK = None                       *
*                                                                   *
* DESCRIPTIVE NAME = %PRODUCT EXEC Interface Block.                 *
*                                                                   *
*    @BANNER_START                          02                      *
*    Licensed Materials - Property of IBM                           *
*                                                                   *
*    "Restricted Materials of IBM"                                  *
*                                                                   *
*    5697-E93                                                       *
*                                                                   *
*    (C) Copyright IBM Corp. 1990, 1993                             *
*                                                                   *
*                                                                   *
*                                                                   *
*                                                                   *
*    @BANNER_END                                                    *
*                                                                   *
* STATUS = %XA20                                                    *
*                                                                   *
```

```
* FUNCTION = EXEC Interface Block.                                     *
*                                                                      *
*        The exec interface block contains information on the          *
*        transaction identifier, the time and date, and the cursor     *
*        position on a display device. Some of the other fields are    *
*        set indicating the next action that a program should take     *
*        in certain circumstances.                                     *
*        DFHEIBLK also contains information that will be helpful       *
*        when a dump is being used to debug a program.                 *
*        This control block is included automatically by an            *
*        application program using the command-level interface.        *
*        EISEIBA in the EIS addresses the EIB.                         *
*                                                                      *
*                                                                      *
*                                                                      *
* NOTES :                                                              *
*    DEPENDENCIES = S/370                                              *
*    MODULE TYPE = Control block definition                            *
*    PROCESSOR = Assembler                                             *
*                                                                      *
*--------------------------------------------------------------------- *
*                                                                      *
* CHANGE ACTIVITY :                                                    *
*   $SEG(DFHEIBLK),COMP(COMMAND),PROD(%PRODUCT) :                      *
*                                                                      *
*     PN= REASON REL YYMMDD HDXXIII : REMARKS                          *
*    $L1= 550    %0G 900515 HDFSPC  : Add an EIB length equate         *
*    $D1= I05119 %B1 930226 HDDHDMA : Correct comments for date field *
*    $P1= M60581 %B0 900116 HDAEGB  : Change for PLXMAP to data areas *
*                                                                      *
************************************************************************
*              EXEC INTERFACE BLOCK                                    *
************************************************************************
DFHEIBLK DSECT                     EXEC INTERFACE BLOCK        @BBAC81A 01-DFHEI
         USING *,DFHEIBR                                       @BBAC81A 01-DFHEI
EIBTIME  DS    PL4                 TIME IN 0HHMMSS FORMAT      @BBAC81A 01-DFHEI
EIBDATE  DS    PL4                 DATE IN 0CYYDDD+ FORMAT,        @D1C 01-DFHEI
*                                  where C is the century          @D1A
*                                  indicator (0=1900, 1=2000),     @D1A
*                                  YY is the year, DDD is the      @D1A
*                                  day number and '+' is the       @D1A
*                                  sign byte (positive)            @D1A
EIBTRNID DS    CL4                 TRANSACTION IDENTIFIER      @BBAC81A 01-DFHEI
EIBTASKN DS    PL4                 TASK NUMBER                 @BBAC81A 01-DFHEI
EIBTRMID DS    CL4                 TERMINAL IDENTIFIER         @BBAC81A 01-DFHEI
EIBRSVD1 DS    H                   RESERVED                    @BBAC81A 01-DFHEI
EIBCPOSN DS    H                   CURSOR POSITION             @BBAC81A 01-DFHEI
EIBCALEN DS    H                   COMMAREA LENGTH             @BBAC81A 01-DFHEI
EIBAID   DS    CL1                 ATTENTION IDENTIFIER        @BBAC81A 01-DFHEI
EIBFN    DS    CL2                 FUNCTION CODE               @BBAC81A 01-DFHEI
EIBRCODE DS    CL6                 RESPONSE CODE               @BBAC81A 01-DFHEI
EIBDS    DS    CL8                 DATASET NAME                @BBAC81A 01-DFHEI
EIBREQID DS    CL8                 REQUEST IDENTIFIER          @BBAC81A 01-DFHEI
EIBRSRCE DS    CL8                 RESOURCE NAME               @BBDIA0U 01-DFHEI
EIBSYNC  DS    C                   X'FF' SYNCPOINT REQUESTED   @BBDIA0U 01-DFHEI
EIBFREE  DS    C                   X'FF' FREE REQUESTED        @BBDIA0U 01-DFHEI
EIBRECV  DS    C                   X'FF' RECEIVE REQUIRED      @BBDIA0U 01-DFHEI
```

```
EIBSEND  DS    C                   RESERVED                    @BM13417 01-DFHEI
EIBATT   DS    C                   X'FF' ATTACH RECEIVED       @BBDIA0U 01-DFHEI
EIBEOC   DS    C                   X'FF' EOC RECEIVED          @BBDIA0U 01-DFHEI
EIBFMH   DS    C                   X'FF' FMHS RECEIVED         @BBDIA0U 01-DFHEI
EIBCOMPL DS    C                   X'FF' DATA COMPLETE                  01-DFHEI
EIBSIG   DS    C                   X'FF' SIGNAL RECEIVED                01-DFHEI
EIBCONF  DS    C                   X'FF' CONFIRM REQUESTED              01-DFHEI
EIBERR   DS    C                   X'FF' ERROR RECEIVED                 01-DFHEI
EIBERRCD DS    CL4                 ERROR CODE RECEIVED                  01-DFHEI
EIBSYNRB DS    C                   X'FF' SYNC ROLLBACK REQ'D            01-DFHEI
EIBNODAT DS    C                   X'FF' NO APPL DATA RECEIVED          01-DFHEI
EIBRESP  DS    F                   INTERNAL CONDITION NUMBER            01-DFHEI
EIBRESP2 DS    F                   MORE DETAILS ON SOME RESPONSES       01-DFHEI
EIBRLDBK DS    CL1                 ROLLED BACK                          01-DFHEI
*
EIBLENG  EQU   *-EIBTIME           Length of EIB                 @L1A 01-DFHEI
********************************************************************************
*              END OF EXEC INTERFACE BLOCK                                     *
********************************************************************************
DFHEIBR  EQU   11                  EIB REGISTER                @BA02936 01-DFHEI
********************************************************************************
*              PROLOG CODE FOR EXEC INTERFACE                                  *
********************************************************************************
*&DFHEICS; CEEENTRY PPA=DFHPPA,MAIN=YES,PLIST=OS,
*               BASE=&CODEREG;,
*               AUTO=(DFHEIEND-DFHEISTG)
TESTLE  CSECT ,                                                          02-CEEEN
TESTLE  RMODE ANY                                                        02-CEEEN
TESTLE  AMODE ANY                                                        02-CEEEN
           ENTRY TESTLE                                                  02-CEEEN
         PUSH  USING                                                     02-CEEEN
         DROP  ,                                                  @02A  02-CEEEN
         USING *,15                                                      02-CEEEN
         B     CEEZ0007                                                  02-CEEEN
         DC    X'00C3C5C5'                                               02-CEEEN
CEEY0007      DC    A((((DFHEIEND-DFHEISTG)+7)/8)*8)                    X02-CEEEN
                                    . Size of automatic storage.
         DC    A(DFHPPA-TESTLE)   . Address of PPA for this program      02-CEEEN
         B     1(,15)                                                    02-CEEEN
CEEZ0007    EQU *                                                        02-CEEEN
         STM   14,12,CEEDSAR14-CEEDSA(13)                                02-CEEEN
         L     2,CEEINPL0007                                5@01D @01C  02-CEEEN
         L     15,CEEINT0007                                      @01C  02-CEEEN
         DROP  15                                                 @01A  02-CEEEN
         BALR  14,15                                                     02-CEEEN
         LR    2,1                                                       02-CEEEN
         L     14,752(,12)                                               02-CEEEN
         OI    8(14),X'80'                                               02-CEEEN
         BALR  3,0                                                @01A  02-CEEEN
         USING *,3                                                @01A  02-CEEEN
         L     3,CEEOEPV0007                                      @01A  02-CEEEN
         POP   USING                                              @01A  02-CEEEN
         USING TESTLE,3                                           @01A  02-CEEEN
         L     1,CEEDSANAB-CEEDSA(,13)  Get the current NAB               02-CEEEN
         L     0,CEEY0007                                                02-CEEEN
         ALR   0,1                     Compute new value.                02-CEEEN
         CL    0,CEECAAEOS-CEECAA(,12)  Compare with EOS.                 02-CEEEN
```

```
         BNH   CEEX0007                                                  02-CEEEN
         L     15,CEECAAGETS-CEECAA(,12) Get address overflow routine   02-CEEEN
         BALR  14,15                   Get another stack segment.       02-CEEEN
         LR    1,15                                                     02-CEEEN
         B     CEEX0007                Branch around statics      @01A  02-CEEEN
CEEINPL0007    DC    A(CEEINPL)                                        @01A  02-CEEEN
CEEINT0007     DC    V(CEEINT)                                         @01A  02-CEEEN
CEEOEPV0007    DC    A(TESTLE)                                         @01A  02-CEEEN
CEEX0007    EQU *                                                           02-CEEEN
         ST    13,CEEDSABKC-CEEDSA(,1)  Set back chain.                 02-CEEEN
         ST    0,CEEDSANAB-CEEDSA(,1)   Set new NAB value               02-CEEEN
         XC    CEEDSAFLAGS-CEEDSA(,1),CEEDSAFLAGS-CEEDSA(1) . Clear     02-CEEEN
         ST    1,CEEDSAFWC-CEEDSA(,13)  Set forward chain.              02-CEEEN
         LR    13,1                    Set save area address            02-CEEEN
         USING CEEDSA,13               Addresability to SF     V1R2M0   02-CEEEN
         MVC   CEEDSALWS,CEECAALWS-CEECAA(12) Get LWS addr     V1R2M0   02-CEEEN
         LR    1,2                                                      02-CEEEN
         BAL   1,*+8                                               @L2A 01-DFHEI
* The following gives an assembler message if DFHEISTG is too big  @P7A
         DS    0S((DFHEISTG+65264-DFHEIEND-4096)/4096)             @04C 01-DFHEI
         DC    AL2(DFHEIEND-DFHEISTG)  LENGTH OF STORAGE           @L2A 01-DFHEI
         DC    H'0'                Parameter list version number   @P6C 01-DFHEI
*********************************************************************
*              ESTABLISH DATA ADDRESSIBILITY                        *
*********************************************************************
DFHEIPLR EQU   13                  PARAMETER LIST REGISTER     @BBAC81A 01-DFHEI
         LR    DFHEIPLR,15                                     @BBAC81A 01-DFHEI
         USING DFHEISTG,13                                     @BBAC81A 01-DFHEI
         MVC   DFHEIBP(L'DFHEIBP+L'DFHEICAP),0(1)                  @D3AX01-DFHEI
                                     COPY EIB AND CA PTRS          @D3A
*********************************************************************
*              ESTABLISH EIB ADDRESSIBILITY                         *
*********************************************************************
         L     DFHEIBR,DFHEIBP                                 @BBAC81A 01-DFHEI
         USING DFHEIBLK,DFHEIBR                                @BBAC81A 01-DFHEI
*********************************************************************
*              END OF PROLOG CODE FOR EXEC INTERFACE                *
*********************************************************************
         MVC  OUTAREA(40),MSG1
         MVC  OUTAREA(4),EIBTRMID
*        EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(43) FREEKB ERASE
         DFHECALL =X'180660000800C20000082204000020',,(______RF,OUTAREA*
               ),(FB_2,=Y(43))
*********************************************************************
         DS    0H                                                       01-DFHEC
         LA    1,DFHEIPL                                                01-DFHEC
         LA    14,=X'180660000800C20000082204000020'                    01-DFHEC
         SR    15,15                                                    01-DFHEC
         LA    0,OUTAREA                                                01-DFHEC
         STM   14,0,0(1)                                                01-DFHEC
         LA    14,=Y(43)                                                01-DFHEC
         ST    14,12(,1)                                                01-DFHEC
         OI    12(1),X'80'         LAST ARGUMENT                        01-DFHEC
         L     15,=V(DFHEI1)                                            01-DFHEC
         BALR  14,15               INVOKE EXEC INTERFACE                01-DFHEC
*********************************************************************
*        EXEC CICS RECEIVE
```

```
         DFHECALL =X'0402000008000000140000040000000'
*********************************************************************
         DS    0H                                                    01-DFHEC
         LA    1,DFHEIPL                                             01-DFHEC
         LA    14,=X'0402000008000000140000040000000'                01-DFHEC
         ST    14,0(,1)                                              01-DFHEC
         OI    0(1),X'80'          LAST ARGUMENT                     01-DFHEC
         L     15,=V(DFHEI1)                                         01-DFHEC
         BALR  14,15               INVOKE EXEC INTERFACE             01-DFHEC
*********************************************************************
         MVC  OUTAREA(13),MSG2
*        EXEC CICS SEND TEXT FROM(OUTAREA) LENGTH(13) FREEKB ERASE
         DFHECALL =X'180660000800C20000082204000020',,(______RF,OUTAREA*
               ),(FB_2,=Y(13))
*********************************************************************
         DS    0H                                                    01-DFHEC
         LA    1,DFHEIPL                                             01-DFHEC
         LA    14,=X'180660000800C20000082204000020'                 01-DFHEC
         SR    15,15                                                 01-DFHEC
         LA    0,OUTAREA                                             01-DFHEC
         STM   14,0,0(1)                                             01-DFHEC
         LA    14,=Y(13)                                             01-DFHEC
         ST    14,12(,1)                                             01-DFHEC
         OI    12(1),X'80'         LAST ARGUMENT                     01-DFHEC
         L     15,=V(DFHEI1)                                         01-DFHEC
         BALR  14,15               INVOKE EXEC INTERFACE             01-DFHEC
*********************************************************************
*        EXEC CICS RETURN
         DFHECALL =X'0E0800000800001000'
*********************************************************************
         DS    0H                                                    01-DFHEC
         LA    1,DFHEIPL                                             01-DFHEC
         LA    14,=X'0E0800000800001000'                             01-DFHEC
         ST    14,0(,1)                                              01-DFHEC
         OI    0(1),X'80'          LAST ARGUMENT                     01-DFHEC
         L     15,=V(DFHEI1)                                         01-DFHEC
         BALR  14,15               INVOKE EXEC INTERFACE             01-DFHEC
*********************************************************************
*
MSG1     DC   C'xxxx: ASM program invoked. ENTER TO END.'
MSG2     DC   C'PROGRAM ENDED'
         DFHEIRET                  INSERTED BY TRANSLATOR
*********************************************************************
*              EPILOG CODE FOR EXEC INTERFACE                       *
*********************************************************************
         DS    0H                                          @BBAC81A 01-DFHEI
         LA   1,CEET0014               Get address of termination list 02-CEETE
         L    15,=V(CEETREC)           Get address of termination rtn  02-CEETE
         BALR 14,15                    Call termination routine.       02-CEETE

CEET0014 DC   A(*+8)               Parm 1                              02-CEETE
         DC   A(*+8+X'80000000')   Parm 2                              02-CEETE
         DC   A(0)                 Enc_Modifier                        02-CEETE
         DC   A(0)                 Return code.                        02-CEETE
CEEMAIN  CSECT                                                         02-CEETE
CEEMAIN  RMODE ANY                                                     02-CEETE
CEEMAIN  AMODE ANY                                                     02-CEETE
```

```
         DC    A(TESTLE)                                          @04A  02-CEETE
         DC    F'0'                                                     02-CEETE
TESTLE     CSECT                                                        02-CEETE
*************************************************************************
*             END OF EPILOG CODE FOR EXEC INTERFACE                     *
*************************************************************************
         LTORG ,                                              @BBAC81A 01-DFHEI
               =V(DFHEI1)
               =V(CEETREC)
               =Y(43)
               =Y(13)
               =X'180660000800C20000082204000020'
               =X'0402000008000000140000040000000'
               =X'0E0800000800001000'
         DS    0H                                       @F8E1S @L1C 01-DFHEI
         DFHEISTG                  INSERTED BY TRANSLATOR
         DFHEIEND                  INSERTED BY TRANSLATOR
*
*************************************************************************
*        P R O G R A M  P R O L O G  A R E A 1   ( P P A 1)             *
*************************************************************************
*
PPA10018    DS    0F                                                     02-CEEPP
DFHPPA   DS    0F                                                       02-CEEPP
         DC    AL1(PPANL0018-*) Offset to the entry name length         02-CEEPP
         DC    X'CE'            LE/370  Indicator.                      02-CEEPP
         DC    B'10100000'                          . PPA flags        02-CEEPP
*                     Bit 0   0 = Internal Procedure
*                             1 = External Procedure
*                     Bit 1   0 = Primary Entry Point
*                             1 = Secondary Entry Point
*                     Bit 2   0 = Block doesn't have a DSA
*                             1 = Block has a DSA
*                     Bit 3   0 = compiled object
*                             1 = library object
*                     Bit 4   0 = sampling interrupts to library
*                             1 = sampling interrupts to code
*                     Bit 5   0 = not an exit DSA
*                             1 = Exit DSA
*                     Bit 6   0 = own exception model
*                             1 = inherited (callers) exception model
*                     Bit 7     Reserved
         DC    X'00'            Member flags                            02-CEEPP
         DC    A(PPA20018)      Addr of Compile Unit Block (PPA2)       02-CEEPP
         DC    A(0)                                                     02-CEEPP
         DC    A(0)             Data Descriptors for this entry point   02-CEEPP
         DS    0H                                                       02-CEEPP
PPANL0018 DC   AL2(6)        .  Length of Entry Point Name              02-CEEPP
         DC    CL6'TESTLE'    .  Entry Point Name                       02-CEEPP
CEEINPL  DS    0D                                                       02-CEEPP
         DC    A(PPA2M0018)                                             02-CEEPP
         DC    A(CEEINPLSTST-CEEINPL)                                   02-CEEPP
CEEINPLSTST DS 0F                                                       02-CEEPP
         DC    X'00'            Control Level                  @01A  02-CEEPP
         DC    X'00'            ENCLAVE=NO                     @01A  02-CEEPP
         DC    X'00'                                           @01A  02-CEEPP
         DC    X'07'            Number of items.               @01C  02-CEEPP
```

```
         DC    A(PPA2M0018)   . A of A(first entry point in comp unit)  02-CEEPP
         DC    V(CEESTART)    . A(Address of CEESTART)                  02-CEEPP
         DC    V(CEEBETBL)                                              02-CEEPP
         DC    A(15)          . Memeber id                              02-CEEPP
         DC    A(0)                                                     02-CEEPP
         DC    XL4'00070000'          . EXECOPS(ON), PLIST              02-CEEPP
         DS    0H                                                       02-CEEPP
*
*********************************************************************
*        P R O G R A M  P R O L O G  A R E A 2   ( P P A 2)          *
*********************************************************************
*
         EXTRN CEESTART                                                 02-CEEPP
PPA20018    DS    0F                                                   02-CEEPP
         DC    AL1(15)          Member ID                             02-CEEPP
         DC    AL1(0)           Sub ID                                02-CEEPP
         DC    AL1(0)           Member defined                        02-CEEPP
         DC    AL1(1)           Level of PPAx control blocks          02-CEEPP
PPA2S0018   DC    A(CEESTART)      A(CEESTART for this load module)     02-CEEPP
         DC    A(0)             A(Compile Debug Information (CDI) )   02-CEEPP
         DC    A(CEETIMES-PPA20018)   A(Offset to time stamp)         02-CEEPP
PPA2M0018   DC    A(TESTLE)      . A(first entry point in comp. unit)  02-CEEPP
*
*********************************************************************
*                T I M E    S T A M P                               *
*********************************************************************
*
*        Time Stamp
*,Time Stamp = 2004/06/17 08:51:00                                     02-CEEPP
*,Version 1 Release 1 Modification 0                                   02-CEEPP
CEETIMES DS    0F                                                     02-CEEPP
         DC    CL4'2004'          Year                               02-CEEPP
         DC    CL2'06'            Month                              02-CEEPP
         DC    CL2'17'            Day                                02-CEEPP
         DC    CL2'08'            Hours                              02-CEEPP
         DC    CL2'51'            Minutes                            02-CEEPP
         DC    CL2'00'            Seconds                            02-CEEPP
         DC    CL2'1'             Version                            02-CEEPP
         DC    CL2'1'             Release                            02-CEEPP
         DC    CL2'0'             Modification                       02-CEEPP
*********************************************************************
*  C O M M O N   A N C H O R   A R E A   ( C A A )                  *
*********************************************************************
*********************************************************************
*********************************************************************
LEPTRLEN EQU    4                                                      03-CEEDN
*
CEECAA      DSECT ,                CAA mapping                         02-CEECA

    (Definition of LE CAA removed)

*              TERMINATE DEFINITION OF DYNAMIC STORAGE                *
DFHEISTG DSECT                                                @BBAC81A 01-DFHEI
         ORG                                                           01-DFHEI
DFHEIEND DS    0X                 END OF DYNAMIC STORAGE      @BBAC81A 01-DFHEI
         END
```

図 12. 変換されアセンブルされたプログラム

第 8 章 アプリケーション・プログラムのインストール

このセクションでは、CICS の下で実行されるアプリケーション・プログラムをインストールする方法について説明します。CICS アプリケーション・プログラムのインストールでは、ソース・ステートメントを変換してコンパイルし、さらにその結果であるオブジェクト・モジュールをリンク・エディットして CICS ライブラリーに入れることが必要です。

アプリケーション・プログラム は、一般的には、CICS コマンド・レベルのアプリケーション・プログラミング・インターフェース (API) を使用するユーザー・プログラムを意味します。このようなプログラムでは、以下のものを使用することも可能です。

- SQL ステートメント
- DLI 要求
- 共通プログラミング・インターフェース (CPI) ステートメント
- SAA リソース・リカバリー・ステートメント
- 外部 CICS インターフェース・コマンド

注: CICS の動的トランザクション・ルーティング機能を使用するアプリケーション・プログラムを開発する場合、CICS Interdependency Analyzer ツールを使用して、プログラムがトランザクション間の類縁性を生じさせる傾向にあるかどうかを検出してください。トランザクション間の類縁性の説明は、339 ページの『第 22 章 類縁性』を参照してください。

このセクションには、以下の内容が含まれています。

- 120 ページの『プログラムのインストール・ステップ』
- 131 ページの『MVS 常駐モードおよびアドレッシング・モードの定義』
- 134 ページの『RDSA にあるアプリケーション・プログラムの実行』
- 137 ページの『アプリケーション・プログラムにおける BMS マップ・セットの使用』
- 138 ページの『アプリケーション・プログラムをインストールするための CICS 提供プロシージャーの使用』
- 140 ページの『CICS 提供インターフェース・モジュールの組み込み』
- 141 ページの『アセンブラー言語アプリケーション・プログラムのインストール』
- 143 ページの『COBOL アプリケーション・プログラムのインストール』
- 147 ページの『PL/I アプリケーション・プログラムのインストール』
- 149 ページの『C アプリケーション・プログラムのインストール』
- 153 ページの『ユーザー独自のジョブ・ストリームの使用』

© Copyright IBM Corp. 1989, 2009

プログラムのインストール・ステップ

CICS で実行するアプリケーション・プログラムをインストールするには、多くのステップを実行する必要があります。

このタスクについて

ステップを次に示します。

1. コンパイラーで CICS コマンドが変換されない場合は、CICS コマンドを、コンパイラーが理解できる呼び出しに変えるためにプログラムのソース・コードを変換する必要がある。
   a. プログラムが CICS コマンドを使用せず、実行中のトランザクションから呼び出されるだけであれば (しかも、CICS タスク開始によって直接起動されることが決してない)、変換プログラムのステップは必要ない。
   b. DL/I CALL インターフェースまたは EXEC DLI インターフェースのどちらかを通じて DL/I サービスにアクセスする CICS コマンド・レベル・プログラムも、変換が必要となる。EXEC SQL インターフェースを使用して DB2 サービスにアクセスするアプリケーションの場合は、さらにプリコンパイルのステップが必要です。
2. プログラム・ソースをコンパイルしてオブジェクト・コードを生成する。
3. オブジェクト・モジュールをリンク・エディットしてロード・モジュールを生成する。生成したロード・モジュールは、DFHRPL または動的 LIBRARY 連結内のアプリケーション・ロード・ライブラリーに保管します。 EXEC SQL インターフェースを使用して DB2 サービスにアクセスするアプリケーションの場合は、さらに INCLUDE ステートメントが必要です。
4. このプログラムを呼び出すトランザクションのリソース定義項目を CSD で作成し、これらをインストールする。
5. 次のいずれかの方式を使用して、プログラムのリソース定義項目を CSD で作成する。
   - プログラムの自動インストール使用
   - RDO の使用

動的プログラム LIBRARY リソースの使用

実行するアプリケーションの場合、ロード・モジュールは、CICS ロード LIBRARY 連結内のデータ・セット内に常駐する必要があります。CICS には 以下の 2 つのタイプのロード LIBRARY 連結があります。

- 静的ロード LIBRARY 連結: DFHRPL。
- 1 つ以上の動的に定義されたロード LIBRARY 連結。

静的 LIBRARY 連結 - DFHRPL

始動 JCL では、CICS に対して静的ロード LIBRARY 連結 の DFHRPL を定義します。DFHRPL には、CICS の始動および実行に必要なクリティカル・データ・セットのほか、アプリケーション・プログラム・エンティティーが含まれています。CICS が実行されると、CICS を停止して再始動しな

い限り DFHRPL データ・セット名を変更することはできません。このような変更は、通常の場合、現在の連続可用性環境ではオプションではありません。

DFHRPL データ・セット名は、MVS 命名規則に準拠している必要があります。

動的プログラム LIBRARY 連結

プログラム LIBRARY 連結は、CICS に対して動的に定義できます。動的 LIBRARY 連結の使用には、システム・プログラマーや組織にとって、以下のような数多くの利点があります。

- プログラム成果物のロード元となる 1 つ以上のデータ・セットが含まれる。
- デプロイメントする新規アプリケーションを、連続可用性に影響を及ぼすことなく、いつでも開始できる。
- 動的 LIBRARY 連結内の既存のアプリケーションの^サービスを、連続可用性に影響することなく、簡単に終了できる。
- 既存のアプリケーションに対するパッチを、既存の LIBRARY より上位のランキングにある LIBRARY 連結内にインストールすることで、連続可用性に影響を及ぼすことなく非常に簡単にインストールできる。
- 動的 LIBRARY 連結内の既存のデータ・セットを、連続可用性に影響を及ぼすことなく、簡単にオフラインにして圧縮できる。

LIBRARY データ・セット名は MVS データ・セットの命名規則に準拠する必要があり、別名データ・セットを使用することができます。

動的プログラム LIBRARY 連結を使用する必要はありません。DFHRPL を使用することもできます。実際、以下のデータ・セットを DFHRPL で定義する必要があります。

- SDFHLOAD
- 第 1 フェーズの PLT プログラム
- 非 SMS 管理のデータ・セット
- SHR 以外の DISP を持つデータ・セット。

動的 LIBRARY 連結は、使用可能または使用不可に設定してインストールおよび作成することができます。

使用可能

使用可能状況を使用可能に指定して LIBRARY をインストールまたは作成する場合、CICS はデータ・セットの割り振り、連結の順に試行してから、最後にその LIBRARY 連結を開きます。これらの手順のいずれかが失敗すると、既に成功した手順も元に返され、LIBRARY は使用不可としてインストールされます。失敗したステップは、メッセージに示されます。

使用不可

使用可能状況を使用不可に指定して LIBRARY をインストールまたは作成する場合、CICS はデータ・セットの割り振りや連結を試行しません。データ・セットが使用可能で、LIBRARY が作動可能になったら、SET LIBRARY ENABLED コマンドを実行してデータ・セットの割り振りおよび連結を行い、LIBRARY を開きます。

SET LIBRARY ENABLED 操作による使用可能化手順のいずれかが失敗すると、既に成功した手順も元に返され、LIBRARY は使用不可のままになります。失敗したステップは、メッセージに示されます。

動的プログラム LIBRARY 連結の使用について、以下の例を参照してください。

動的 LIBRARY リソースの使用例

動的 LIBRARY リソースを採用するかどうか選択します。これはテスト環境でも、実稼働環境でも、またはその両方でも使用できます。一部のデータ・セットを DFHRPL の外に移動し、動的に定義して、DFHRPL と動的プログラム LIBRARY リソースを組み合わせて使用することができます。

動的 LIBRARY で定義する候補としては、1 つ以上のデータ・セットで提供される、ベンダー・パッケージや企業内アプリケーションなどがあります。

動的プログラム LIBRARY 連結を使用してプログラムを管理する方法を以下の例に示します。

CICS システムへの緊急修正の適用

始める前に

前提

- CICS システムの使用しているバージョンのアプリケーションに、訂正が必要な問題がある。
- そのアプリケーションの更新されたバージョンが既に作成されている。
- その修正を適用するために今すぐ CICS を再始動することができない。
- この操作を実行するユーザーには適切なアクセス権限がある。

このタスクについて

目的

プログラム修正を含む一時 LIBRARY を CICS 領域にインストールすること。

プロセス

1. 修正を提供するプログラムおよびその他の作成物を、PDS または PDSE データ・セット、またはデータ・セットのセットに追加する。
2. CICSPlex SM のコンポーネントの Business Application Services (BAS)、または CEDA、DFHCSDUP、または EXEC CICS CREATE を使用して、修正が入っているデータ・セットを含む LIBRARY リソースを定義する。この LIBRARY リソースは、検索順序において、障害のあるバージョンのプログラムを含む LIBRARY より上位のランキングを保持している必要があります。これは、検索順序において、DFHRPL より前に LIBRARY を置くことになる可能性があります。
3. EXEC CICS CREATE を使用せずに LIBRARY を定義した場合は、新しい LIBRARY リソースを CEDA または CICSPlex SM WUI を使用してインストールする。

4. 問題のプログラムに対して、EXEC CICS SET PROGRAM NEWCOPY または EXEC CICS SET PROGRAM PHASEIN コマンド、または CICSPlex SM か CEMT の相当するコマンドを発行する。

次のタスク

結果

CICS システムは、その稼働を続行しており、検索順序においてアプリケーションの修正バージョンが問題のあるバージョンより前に置かれているので、その修正バージョンが代わりに使用されます。

CICS システムへの新規アプリケーションのインストール

始める前に

前提:

- アプリケーションが 1 つ以上の PDS または PDSE データ・セットで提供されている。アプリケーションは、1 つ以上の PDS または PDSE データ・セット内部のアプリケーション成果物のセットとして提供されるサード・パーティー (ベンダー) の製品か、新規の社内アプリケーションなどが考えられます。
- その修正を適用するために今すぐ CICS を再始動することができない。
- この操作を実行するユーザーには適切なアクセス許可がある。

このタスクについて

目的

1 つ以上のデータ・セットで提供される新規アプリケーションを、連続可用性に影響を及ぼすことなく、稼動中の CICS システムに導入すること。

プロセス

1. CICSPlex SM のコンポーネントの Business Application Services (BAS)、または CEDA、DFHCSDUP、または EXEC CICS CREATE を使用して、新規アプリケーションが入っているデータ・セットを含む LIBRARY リソースを定義する。通常、そのアプリケーションは既存の LIBRARY リソースとの交差を持たないため、デフォルトのランキング値を使用できます。
2. EXEC CICS CREATE を使用せずに LIBRARY を定義した場合は、新しい LIBRARY リソースを CEDA または WUI を使用してインストールする。
3. アプリケーションおよびそのアプリケーションを参照する 1 つ以上のトランザクション定義を構成する、プログラム、マップ・セットを CICS に定義する。
4. プログラムおよびその他の定義をインストールする。

次のタスク

結果

新規アプリケーションが CICS 実動システムにインストールされ、連続可用性も維持されます。

CICS システムのセットへの新規アプリケーションのインストール

始める前に

前提:

- アプリケーションが 1 つ以上の PDS または PDSE データ・セットで提供されている。
- アプリケーションが複数の CICS システムに同時に導入される。
- 新規アプリケーションを追加するために今すぐ CICS 領域を再始動することができないか、CICS の実行に対してそのアプリケーションがクリティカルではない。
- この操作を実行するユーザーには適切なアクセス権限がある。

このタスクについて

目的

1 つ以上のデータ・セットで提供される新規アプリケーションを、CICSplex 内の CICS システムのセットに導入すること。そのようなシステムは、実動中である可能性が高いにもかかわらず、テストまたは開発 CICSplex 内にある可能性もあります。

プロセス

1. CICSPlex SM BAS を使用して、アプリケーション・データ・セットを含む CICSPlex SM LIBRARY 定義 (LIBDEF) を定義する。
2. その CICS 領域内で使用中の他の LIBRARY リソースと関連した順序を反映させた、LIBRARY のランキングを指定する。通常、そのアプリケーションは既存の LIBRARY リソースとの交差を持たないため、デフォルトのランキング値を使用できます。
3. CICS システムのセットが含まれるターゲットの有効範囲を指定して、新しい LIBDEF をインストールする。
4. アプリケーションおよびそのアプリケーションを参照する 1 つ以上のトランザクション定義を構成する、プログラム、マップ・セット、およびその他の成果物を CICS に定義する。
5. CICS システムのセットにプログラムやその他の定義をインストールし、それらの使用を開始する。

次のタスク

結果

新規アプリケーションを使用して CICS 領域が実行されます。

LIBRARY 編成内の CICS アプリケーションの再構築

始める前に

前提:

- 現在すべてのアプリケーションが、DFHRPL 連結内の DD カードとして定義されたデータ・セット内にある。
- この操作を実行するユーザーには適切なアクセス許可がある。

このタスクについて

目的

アプリケーションの編成を LIBRARY リソースに再構築すること (データ・セット名を、操作の適合性ではなく、そのデータ・セットに含まれているアプリケーションに関係したものにするなど)。

プロセス

1. LIBRARY データ・セットへのアプリケーションの新しい割り振りを決定し、各アプリケーションに 1 つの LIBRARY にするか、各 LIBRARY に複数のアプリケーションにするかを決定する。LIBRARY ごとに 1 つのアプリケーションを選択して、簡潔なシステム構成と保守の容易さを維持することができます。また、1 つに連結された複数データ・セットを LIBRARY 内に必要とするアプリケーションや、単一データ・セットを必要とするアプリケーションも決定します。
2. DFHRPL 内に残すアプリケーションと、動的リソースにするアプリケーションを決定する。
3. 動的 LIBRARY で定義されるアプリケーションで CICS 始動にクリティカルなアプリケーションとクリティカルではないアプリケーションを決定する。
4. CICSPLEX SM、CEDA、DFHCSDUP、または EXEC CICS CREATE のコンポーネントの Business Application Services LIBRARY コマンドを使用して、各アプリケーション (またはアプリケーションがグループ化される場合はアプリケーション・セット) の動的リソースになる LIBRARY リソースを定義する。
   a. その CICS 領域内で使用中の他の LIBRARY と関連した順序を反映させた、各 LIBRARY のランキングを指定する。通常、そのアプリケーションは既存の LIBRARY との交差を持たないため、デフォルトのランキング値を使用できます。
   b. CICS の実行に対してクリティカルな各 LIBRARY に CRITICAL 状況を指定し、そうではない LIBRARY はデフォルトの状況のまま残す。
   c. LIBRARY 内のデータ・セットの名前を指定する。
5. CEDA INSTALL LIBRARY コマンド または CICSPlex SM WUI を使用して、新しい LIBRARY リソースをインストールする。
6. 動的 LIBRARY 連結内にあるアプリケーションを収容するデータ・セットを、次回の CICS 再始動時に DFHRPL 連結から除去する。
7. オプションで、テスト目的で LIBRARY を DFHRPL の前に置く値にランキングを設定した場合は、各 LIBRARY のランキングを永続を意図された値にリセットする。
8. GRPLIST からの CICS の再始動中か、BAS のインストール中、または CICS の再始動後のいずれかのタイミングで新規 LIBRARY リソースをインストールする。この段階で、システムはプログラムを (既に DFHRPL 連結内には存在しないため) 新規 LIBRARY リソースからロードします。

次のタスク

結果

CICS は以前と同じように稼動しますが、アプリケーションのセットの編成は改善され、アプリケーションの場所およびアプリケーションがインストールされた CICS システムの追跡が容易になります。

CICS システムからの LIBRARY のオフライン化、またはアプリケーションの除去

始める前に

前提:

- アプリケーションが、動的 LIBRARY リソース内の既知のデータ・セットまたはデータ・セットのセット内にある。
- この操作を実行するユーザーには適切なアクセス許可がある。

このタスクについて

目的

例えば、PDS の圧縮や、稼動中の CICS システムからアプリケーションを除去するために、LIBRARY をオフラインにすること。

プロセス

1. EXEC CICS SET LIBRARY コマンド、CEMT または WUI を使用して、LIBRARY を使用不可にする。
2. そのアプリケーションの使用がすべて完了したら、ロードされたプログラムのコピーが除去される操作、例えば、SET PROGRAM NEWCOPY を実行するか、次の再ロードでプログラムが失敗するのを許可する。
3. LIBRARY を使用不可に設定する 1 つの理由は、データ・セットを圧縮してから、LIBRARY を使用可能に設定するか、LIBRARY 定義を再インストールして、アプリケーションの使用を再開することです。 LIBRARY を再び使用可能にした後で、PROGRAM NEWCOPY または PHASEIN を発行してプログラムの使用を再開します。

次のタスク

結果

LIBRARY が使用不可に設定されている間、新規ユーザーはそのアプリケーションを使用できなくなります。ただし、検索順序において使用不可に設定された LIBRARY より後にある別の LIBRARY にコピーが存在する場合は、そこからロードされます。

2 つの LIBRARY 連結間の切り替え

始める前に

前提:

- プログラム、またはアプリケーションを構成する複数のプログラム成果物を収容している LIBRARY が、現在 CICS にインストールされている。
- 1 つ以上の PDS または PDSE データ・セット内にあるプログラムまたはアプリケーションの新バージョンが使用可能である。
- この操作を実行するユーザーには適切なアクセス権限がある。

このタスクについて

目的

一方の LIBRARY を CICS に導入し、もう一方の LIBRARY をオフラインにすることで、新 LIBRARY 内のプログラムをロードして、旧 LIBRARY 内のプログラムを置換すること。

プロセス

1. CICSPlex SM BAS を使用して、新しい アプリケーション・データ・セットを含む CICSPlex SM LIBRARY 定義 (LIBDEF) を定義する。
2. 新しい LIBRARY をインストールする。
3. PROGRAM NEWCOPY または PHASEIN コマンドを発行して、1 つ以上のプログラムの新規コピーの使用を開始する。
4. 古い LIBRARY リソースを使用不可に設定するか、再使用する可能性がない場合は廃棄する。
5. オプションで、新しい LIBRARY のランキングを CEMT SET LIBRARY、WUI、SPI プログラム、または CICSPlex SM API を使用して古い LIBRARY のランキングに戻す。

次のタスク

結果

CICS は、新しい LIBRARY とアプリケーションの新バージョンを使用して実行されます。

CICS システム内の LIBRARY リソースに関する情報の検出

始める前に

前提:

- CICS が稼働中である。

このタスクについて

目的

以下のような LIBRARY リソースに関する情報を検出すること。

- CICS にインストールされている LIBRARY リソース。
- CICS 内のアクティブな LIBRARY リソース (例えばインストール済みの LIBRARY リソースや使用可能な LIBRARY リソースなど) の現在の検索順序。
- 2 つの LIBRARY 連結の、検索順序における相対位置。

- クリティカルな LIBRARY リソース。
- LIBRARY 連結に対して定義されているデータ・セット。

プロセス

CICSPlex SM WUI または CEMT INQUIRE LIBRARY コマンドを使用して、ライブラリーに対する照会を行う。特定の LIBRARY またはプロパティーを指定しない場合、この照会では、インストール済みのすべての LIBRARY リソースが、現在の検索順序で表示されます。LIBRARY リソースの一部のプロパティーを指定する場合は、インストール済みの LIBRARY リソースのサブセットが表示されます。特定の LIBRARY が指定されている場合は、その LIBRARY の詳細情報が表示されます。

次のタスク

結果

この照会によって、LIBRARY リソースのクリティカル状況および使用可能化状況、ランキング、検索順序全体における絶対位置が表示されます。使用不可に設定されている LIBRARY リソースは、リストには表示されますが、検索順序には入りません。LIBRARY の詳細を表示すると、その連結内のデータ・セットが表示されます。CICSPlex SM WUI を使用する場合は、「**Number of DSNAMEs (DSNAME 数)**」フィールド名をクリックして LIBRARY のデータ・セット名レコードを表示します。照会要求またはコマンドを発行して、2 つの LIBRARY 連結の検索位置番号を比較し、検索順序全体においてどちらが前にあるかを判別することができます。

CICS システム内のプログラムに関する LIBRARY 情報の検出

始める前に

前提:

- CICS が稼働中である。
- プログラムが CICS システム内で使用されている。

このタスクについて

目的

(例えば、プログラムが意図された場所からロードされたかどうかを確認するために) CICS システム内のプログラムに対する照会を行って、プログラムのロード元の LIBRARY (およびその LIBRARY 内のデータ・セット) を調べること。

プロセス

WUI または CEMT INQUIRE PROGRAM コマンドを使用して、プログラムに対する照会を行う。

次のタスク

結果

そのプログラムのロード元の LIBRARY やデータ・セットなどの情報を含む、プログラム情報が返されます。

- インストール済み LIBRARY からプログラムがロードされた場合は、LIBRARY と LIBRARYDSN 名が返される。
- 使用不可に設定された LIBRARY からプログラムがロードされた場合は、LIBRARY 名は返されるが LIBRARYDSN は空白になる。
- 廃棄された LIBRARY からプログラムがロードされた場合は、LIBRARY と LIBRARYDSN の両方が空白になる。
- プログラムがロードされていない場合は、LIBRARY と LIBRARYDSN が両方とも空白になる。
- LPA からプログラムがロードされた場合は、LIBRARY と LIBRARYDSN の両方が空白になる。

LIBRARY リソースの CRITICAL プロパティーの修正

始める前に

前提:

- CICS が稼働中である。
- CICS システムで最低 1 つの動的 LIBRARY がアクティブである。

このタスクについて

目的

CICS 始動に対して、1 つ以上の CICS 領域内の LIBRARY をクリティカルとして指定すること。

プロセス

1. CICSPlex SM WUI、CEMT、または SPI を使用して、インストール済みの LIBRARY のクリティカル状況を変更する。この変更は、次のウォーム始動または緊急始動時に発効します。そのときのクリティカル状況によって、LIBRARY 連結内のデータ・セットのいずれかが使用不可な場合、またはその他の問題によって LIBRARY が有効として回復することが妨げられる場合に CICS の始動を中断せずに実行するかどうかが決まります。
2. 永久に CRITICAL ステータスに変更するには、CICSPlex SM BAS または CEDA を使用して LIBRARY 定義を必要なクリティカル状況に定義し、CICS コールド・スタートまたは初期始動のたびに CICSPlex SM BAS インストールまたは GRPLIST インストールを使用して定義をインストールする。

次のタスク

結果

再始動時の CICS の振る舞いは、LIBRARY のクリティカル設定や、LIBRARY 内のデータ・セットが使用可能であるかどうかによって異なります。

LIBRARY 構成への変更の追跡の継続

始める前に

前提:

- CICS が稼働中である。

このタスクについて

目的

監査ログを使用して、CICS システムの LIBRARY 構成に対する以下のような変更を判別すること。

- 新規 LIBRARY がインストールされる。
- CICS から LIBRARY が除去 (廃棄) される。
- LIBRARY のランキング、クリティカル状況、または使用可能化状況が変更される。
- 全体的な LIBRARY 検索順序が変更される。

プロセス

1. CSLB 一時データ・キューに書き込まれる監査ログを調べて、CICS システム内の LIBRARY 構成に対する変更と、その結果的な LIBRARY 検索順序を参照する。
2. オプションで、社内またはベンダーで開発されたユーティリティーを使用して、このシステムまたは複数のシステムの監査ログを分析および解釈する。

次のタスク

結果

CICS は実行を継続します。

LIBRARY 構成のタイディアップ

始める前に

前提:

- CICS が稼働中である。

このタスクについて

目的

一時的な修正の適用に使用された、または既に使用されていないアプリケーションを含む LIBRARY 連結のタイディアップ。

プロセス

1. CICS にインストールされている LIBRARY リソースの名前および LIBRARY の変更に関する監査ログを調べ、一時的な修正の適用に使用された、既に必要のない LIBRARY リソースや、既に使用されていないアプリケーションの LIBRARY リソースを発見する。
2. 必要ではなくなった LIBRARY リソースを廃棄する。
3. 今後必要になるという確信がない限り、これらの LIBRARY リソースの定義を削除して、操作手順によって LIBRARY 定義を再利用できるようにする。

次のタスク

結果

CICS は従来どおりに実行を継続します。CICS システムにインストールされている LIBRARY リソースのセットは、システムで現在使用されているアプリケーションに必要なものだけになります。

MVS 常駐モードおよびアドレッシング・モードの定義

このセクションでは、MVS 常駐モードおよびアドレッシング・モードがアプリケーション・プログラムに与える効果と、モードの変更方法、アプリケーション・プログラムを永続的に常駐させる方法について説明します。

このタスクについて

MVS/370 で実行されるように作成されたアプリケーションは、AMODE(24) および RMODE(24) オプションを指定してリンク・エディットすれば、どの MVS システムにおいても実行できます。

コマンド・レベル・プログラムは、16MB より高位、すなわち 16MB より高位のアドレス域に常駐することが可能です。このプログラムには、EXEC CICS コマンド、EXEC DLI コマンド、および CALL DL/I コマンドを組み込むことができます。

プログラムのアドレッシング・モードの設定

MVS で実行されるすべてのプログラムには、2 つの属性、アドレッシング・モード (AMODE) と常駐モード (RMODE) が割り当てられます。 AMODE は、プログラムが制御を受けるように設計されたアドレッシング・モードを指定します。

このタスクについて

プログラムのモードは切り替えが可能ですが、一般的にユーザーのプログラムはこのモードで実行されるように設計され、ロード・モジュール内の異なる入り口点に対して異なる AMODE 属性を持ちます。RMODE 属性は、ユーザー・プログラムの仮想記憶域における常駐可能な場所を示します。AMODE および RMODE の有効な指定値は以下のとおりです。

AMODE(24)　24 ビット・アドレッシング・モードを指定します。
AMODE(31)　31 ビット・アドレッシング・モードを指定します。
AMODE(ANY)　24 ビットまたは 31 ビット・アドレッシング・モードを指定します。

RMODE(24)　モジュールが 16MB より低位の仮想記憶域に常駐しなければならないことを示します。24 ビット依存性を持つ 31 ビット・プログラムの場合は、RMODE(24) を指定することができます。

RMODE(ANY)　モジュールが仮想記憶域のどこにでも常駐できることを示します。

注: C または C++ 言語のプログラムは、AMODE(31) を指定してリンク・エディットする必要があります。

プログラムに AMODE 属性 や RMODE 属性を指定しないと、MVS がシステム・デフォルトの AMODE(24) および RMODE(24) を割り当てます。 AMODE と RMODE を 以下の場所のいずれかで指定すると、これらのデフォルトを指定変更することができます。このリストで示す割り当ては、このリストの割り当てを後で上書きします。

1. リンク・エディットの MODE 制御ステートメント。

   ```
   MODE AMODE(31),RMODE(ANY)
   ```

2. 以下のうちのどちらか。
   - リンク・エディット・ジョブ・ステップの EXEC ステートメント内 PARM ストリング。

     ```
     //LKED    EXEC  PGM=IEWL,PARM='AMODE(31),RMODE(ANY),..'
     ```

   - リンク・エディット・ステップで EXEC ステートメントの処理と同等な処理を行う LINK TSO コマンド。
3. アセンブラー・プログラムのソース・コード内の AMODE ステートメントまたは RMODE ステートメント。 これらのモードは COBOL でも、コンパイラー・オプションを使用して設定できます。COBOL コンパイラー・オプションの詳細については、ご使用の COBOL コンパイラーに関係のあるアプリケーション・プログラミング・ガイドを参照してください。

CICS アドレス・スペースに関する考慮事項

表 7 は、AMODE 属性と RMODE 属性の有効な組み合わせとその効果を示したものです。

表 7. AMODE および RMODE の有効な指定とその効果

AMODE	RMODE	常駐	アドレッシング
24	24	16 MB より低位	24 ビット・モード
31	24	16 MB より低位	31 ビット・モード
任意	24	16 MB より低位	31 ビット・モード
31	任意	16 MB より高位	31 ビット・モード

以下の例は、31 ビット規格でコーディングされたプログラムのリンク・エディット制御ステートメントを示しています。

```
//LKED.SYSIN DD *
   MODE AMODE(31),RMODE(ANY)
   NAME    anyname(R)    ("anyname" is your load module name)
/*
//
```

永続的に常駐するプログラムの作成

このタスクについて

プログラムを常駐属性 RESIDENT(YES) を指定して CSD で定義すると、プログラムは最初の参照時にロードされます。このことは、RMODE(ANY) または RMODE(24) のいずれかを指定してリンク・エディットされたプログラムにも適用されます。ただし、CICS が使用するストレージ圧縮アルゴリズムが、常駐プログラムを除去しないことに注意してください。

タスクがプログラムをロードするストレージが十分でない場合は、そのタスクは十分なストレージが使用できるようになるまで中断されます。いずれかの DSA がストレージ不足になりそうな場合、CICS は使用中でないプログラムが占有するストレージを解放します。CICS の動的ストレージ域について詳しくは、「*CICS System Definition Guide*」の 動的ストレージ域を参照してください。

RMODE(24) のプログラムを常駐にしないで、非常駐にし、ライブラリー・ルックアサイド機能 (LLA) を使用することができます。このようなプログラムが占有するスペースは、その使用回数がゼロになったときに解放され、使用可能な仮想記憶域が増えます。LLA はストレージおよびステージ (場所) 内にライブラリー・ディレクトリーを保持し、LLA 管理ライブラリー・モジュールのコピーを、仮想ルックアサイド機能 (VLF) によって管理されるデータ・スペースに入れます。CICS は、DASD 上のプログラム・ディレクトリーを検索せず、プログラム・モジュールを LLA ディレクトリーからストレージに配置します。CICS が配置されたモジュールを要求すると、LLA は入出力なしでストレージからモジュールを取得します。

リンク・パック域にあるアプリケーションの実行

プログラムは、特定の要件を満たすとリンク・パック域 (LPA) に常駐することができます。

アセンブラー言語、C、COBOL、または PL/I プログラムは、読み取り専用で、次の要件を順守する必要があります。

アセンブラー
RENT アセンブラー・オプションを使用します。

C RENT コンパイラー・オプションを使用します。

COBOL
WORKING STORAGE を上書きしないでください (CICS 変換プログラムは、(変換プログラム・オプション NOCBLCARD を指定しない限り) 必須の RENT コンパイラー・オプションを持つ CBL ステートメントを生成します)。

PL/I
STATIC ストレージを上書きしないでください (CICS 変換プログラムは必須の REENTRANT オプションを PROCEDURE ステートメントに挿入します)。

すべてのプログラムは、RENT と REFR オプションを指定してリンク・エディットする必要があります。

これらの規格に合わせて作成し、LPA にインストール済みのモジュールを CICS で使用したい場合は、CSD のプログラム・リソース定義で USELPACOPY(YES) を指定します。

RDSA にあるアプリケーション・プログラムの実行

16MB より上のアドレスに常駐させることが可能で、読み取り専用のプログラムは、CICS 拡張読み取り専用 DSA (ERDSA) に常駐させることができます。16MB より上のアドレスに常駐するのに適格ではない が、読み取り専用であるプログラムの場合は、16MB より低位の CICS 読み取り専用 DSA (RDSA) に常駐することができます。

プログラムが ERDSA での常駐に適格であるためには、以下のことが必要です。

- 読み取り専用規格に対して適正に作成されている
- 31 ビット・アドレッシング規格で作成されている
- RENT 属性および RMODE(ANY) 常駐属性を指定してリンク・エディットされている

プログラムが、ERDSA には的確ではないが RDSA に的確である場合は、次のことが必要です。

- 読み取り専用規格に対して適正に作成されている
- RENT 属性を指定してリンク・エディットされている

注: システム初期化パラメーターとして `RENTPGM=PROTECT` を指定して CICS を実行している場合、RDSA はキー 0 の読み取り専用ストレージから割り振られます。

RENT および RMODE(ANY) を指定してリンク・エディットされたプログラムは、CICS によって、自動的に ERDSA にロードされます。

特定の言語に対する ERDSA の要件は、以下で説明するとおりです。

アセンブラー

CICS でアセンブラー・プログラムを ERDSA にロードしたい場合は、以下のオプションを指定して、プログラムをアセンブルおよびリンク・エディットします。

1. RENT アセンブラー・オプション
2. リンク・エディットの RENT 属性
3. RMODE(ANY) 常駐モード

注: これらのオプションを指定する場合は、プログラムが間違いなく読み取り専用 (すなわち、例えば静的ストレージへの書き込みなどのいかなる方法でも、それ自身が変更されないこと) であることを確認します。読み取り専用でないと、ストレージ例外が発生します。さらにプログラムは、31 ビット・アドレッシング規格で作成されなければなりません。ERDSA に常駐するプログラムのストレージ保護例外の考えられる原因については、「*CICS Problem Determination Guide*」の保護例外の原因を参照してください。

CICS 提供プロシージャー DFHEITAL には、XREF および LIST オプションのみを指定するパラメーター LNKPARM があります。 ERDSA に対して適格なプログラムをリンク・エディットするには、呼び出しジョブから LNKPARM を指定変更して、他の必要なオプションだけでなく、RENT および RMODE(ANY) オプションも指定します。

例:

```
//ASMPROG  JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//EITAL    EXEC DFHEITAL,
           .
           (other parameters as necessary)
           .
//         LNKPARM='LIST,XREF,RMODE(ANY),RENT'
```

注: アセンブラー・プログラム (DFHEAI) の CICS EXEC インターフェース・モジュールは、AMODE(ANY) および RMODE(ANY) を指定します。ただし、アセンブラーはアプリケーションを AMODE(24) および RMODE(24) にデフォルト指定するため、その結果であるロード・モジュールも AMODE(24) および RMODE(24) になります。

アプリケーション・プログラムを AMODE(31) および RMODE(ANY) としてリンク・エディットしたい場合は、アセンブラー・プログラムに適切なステートメントを使用することをお勧めします。例:

```
MYPROG    CSECT
MYPROG    AMODE 31
MYPROG    RMODE ANY
```

AMODE および RMODE を設定するには、2 つの方法があります。

- JCL の PARM キーワードでリンク・エディット (またはバインダー) 制御情報を使用すると、必要な AMODE および RMODE を指定することができる。例:

  ```
  //EITAL  EXEC DFHEITAL,
           LNKPARM='LIST,XREF,RENT,AMODE(31),RMODE(ANY)'
  ```

- 代わりに、JCL のリンク・エディットまたはバインダー・ステップで、SYSIN データ・セットの MODE 制御ステートメントを使用することも可能。

バインダーを使用すると、AMODE と RMODE の指定の矛盾に関して、予期しない警告メッセージが表示されることがあります。

C および C++

CICS で C プログラムおよび C++ プログラムを ERDSA にロードする場合は、以下のオプションを指定してプログラムをコンパイルおよびリンク・エディットします。

1. RENT コンパイラー・オプション

CICS 提供プロシージャー DFHYITDL または DFHYITFL (C の場合) および DFHYITEL または DFHYITGL (C++ の場合) には、多数のリンク・エディット・オプションを指定する LNKPARM パラメーターがあります。 ERDSA に対して適格なプログラムをリンク・エディットするには、呼び出しジョブからこのパラメーターを指定変更して、必要な他のオプションに RENT を追加します。 C 用の CICS EXEC インターフェース・モジュール (DFHELII) は、AMODE(31) および

RMODE(ANY) を指定してリンク・エディットされるため、RMODE(ANY) オプションを追加する必要はありません。したがって、CICS EXEC インターフェース・スタブ (106 ページの『CICS 提供のインターフェース・モジュール』を参照してください) を組み込むと、自動的に AMODE(31) および RMODE(ANY) としてプログラムがリンク・エディットされます。

以下のジョブ・ステートメントの例は、RENT オプションが追加された LNKPARM パラメーターを示しています。

```
//CPROG    JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITDL    EXEC DFHYITDL,
           .
           (other parameters as necessary)
           .
//         LNKPARM='LIST,MAP,LET,XREF,RENT'
```

COBOL

統合 CICS 変換プログラムを使用する場合、コンパイルでは、RENT コンパイラー・オプションが必要で、CBL カードを変換中に追加する必要はありません。独立した変換ステップを使用する COBOL プログラムは、自動的に ERDSA に対して適格です。その理由は、以下のとおりです。

- 変換プログラム・オプション CBLCARD (デフォルト) は、必須のコンパイラー・オプション RENT が、CICS 変換プログラムによって生成された CBL ステートメントに自動的に組み込むため。変換プログラム・オプション NOCBLCARD を使用する場合、RENT オプションは、コンパイル・ジョブ・ステップの PARM ステートメントでも指定できますし、インストール・システムで定義されたオプションを設定する COBOL マクロ IGYCOPT を使用しても指定できます。
- COBOL コンパイラーは、読み取り専用と 31 ビット・アドレッシングの規格に合致するコードを自動的に生成するため。
- COBOL 用の CICS EXEC インターフェース・モジュール (DFHELII) は、AMODE(31) および RMODE(ANY) を指定してリンク・エディットされるため。したがって、CICS EXEC インターフェース・スタブ (106 ページの『CICS 提供のインターフェース・モジュール』を参照してください) を組み込むと、自動的に AMODE(31) および RMODE(ANY) としてプログラムがリンク・エディットされます。

リンク・エディットに再入可能属性を指定する必要もあります。 CICS 提供プロシージャー DFHYITVL には、多数のリンク・エディット・オプションを指定する LNKPARM パラメーターがあります。ERDSA に対して適格なプログラムをリンク・エディットするには、呼び出しジョブからこのパラメーターを指定変更して、必要な他のオプションに RENT を追加します。例:

```
//COBPROG  JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITVL    EXEC DFHYITVL,
           .
           (other parameters as necessary)
           .
//         LNKPARM='LIST,XREF,RENT'
```

PL/I

CICS PL/I プログラムは、静的ストレージを変更しないという条件で、通常 ERDSA に対して適格です。以下の要件は、CICS または PL/I のいずれかに課されます。

- 必須の REENTRANT オプションは、CICS 変換プログラムによって、PL/I の PROCEDURE ステートメントに自動的に組み込まれる。
- PL/I コンパイラーは、31 ビット・アドレッシング規格に合致するコードを自動的に生成する。
- PL/I 用 CICS EXEC インターフェース・モジュール (DFHELII) は、AMODE(31) および RMODE(ANY) を指定してリンク・エディットされる。したがって、CICS EXEC インターフェース・スタブ (106 ページの『CICS 提供のインターフェース・モジュール』を参照してください) を組み込むと、自動的に AMODE(31) および RMODE(ANY) としてプログラムがリンク・エディットされます。

リンク・エディットに再入可能属性を指定する必要もあります。CICS 提供プロシージャー DFHYITPL には、多数のリンク・エディット・オプションを指定する LNKPARM パラメーターがあります。ERDSA に対して適格なプログラムをリンク・エディットするには、呼び出しジョブからこのパラメーターを指定変更して、必要な他のオプションに RENT を追加します。例:

```
//PLIPROG  JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//YITPL    EXEC DFHYITPL,
           .
           (other parameters as necessary)
           .
//         LNKPARM='LIST,XREF,RENT'
```

注: プログラムが間違いなく読み取り専用 (すなわち、例えば静的ストレージに書き込まない) であることを確認しない限り、リンク・エディット・ステップで RENT 属性を指定しないでください。読み取り専用でないと、ストレージ例外が発生します。ERDSA に常駐するプログラムのストレージ保護例外の考えられる原因については、「*CICS Problem Determination Guide*」の保護例外の原因を参照してください。

アプリケーション・プログラムにおける BMS マップ・セットの使用

このセクションでは、アプリケーション・プログラムにおいて BMS マップ・セットを使用する方法について説明します。

このタスクについて

CICS のもとで実行されるアプリケーション・プログラムをインストールする前に、以下を実行してください。

- 157 ページの『第 9 章 マップ・セットおよび区分セットのインストール』で説明するように、プログラムが使用するすべての BMS マップ・セットを作成する。
- この物理マップ・セット (BMS が形式設定アクティビティーで使用する) を、DFHRPL または動的 LIBRARY 連結にあるデータ・セットに組み込む。
- シンボリック・マップ・セットをユーザー・コピー・ライブラリーに組み込むか (アプリケーション・プログラムにコピーされる)、あるいは直接アプリケーション・プログラム・ソースに組み込む。

DFHMAPS プロシージャーは、シンボリック・マップ・セットの出力を、DSCTLIB パラメーターで指定したライブラリー (デフォルトでは CICSTS41.CICS.SDFHMAC ライブラリー) に書き込みます。シンボリック・マップ・セットをユーザー・コピー・ライブラリーに組み込みたい場合、以下を実行します。

- 物理マップ・セットおよびシンボリック・マップ・セットを一緒にインストールするために使用される DFHMAPS プロシージャーで、EXEC ステートメントの *DSCTLIB=name* オペランドによって、ライブラリー名を指定する。
- ユーザー・コピー・ライブラリーの DD ステートメントを、アプリケーション・プログラムをアセンブルおよびコンパイルするのに使用されるジョブ・ストリームの SYSLIB 連結に組み込む。

  DFHMAPS プロシージャーでシンボリック・マップ・セットを CICSTS41.CICS.SDFHMAC ライブラリー (デフォルト) に書き込むように選択する場合は、CICSTS41.CICS.SDFHMAC ライブラリーの DD ステートメントを、アプリケーション・プログラムのコンパイルに使用するジョブ・ストリームの SYSLIB 連結に組み込みます。これは、アセンブラー言語プログラムのアセンブルに使用する DFHEITAL プロシージャーの場合は必要ありません。それは、このプロシージャーのジョブでは、CICSTS41.CICS.SDFHMAC ライブラリーの DD ステートメントを既に SYSLIB 連結に組み込んでいるためです。
- PL/I では、ブロック・サイズが 32760 バイトのライブラリーを指定する。これは、PL/I コンパイラーのブロック・サイズ制限に対処するために必要です。

マップ・セットのインストールの詳細については、157 ページの『第 9 章 マップ・セットおよび区分セットのインストール』を参照してください。BMS サービスを使用するプログラムの作成の詳細については、775 ページの『第 60 章 基本マッピング・サポート』を参照してください。

アプリケーション・プログラムをインストールするための CICS 提供プロシージャーの使用

CICS は、変換 (必要であれば)、コンパイル、およびリンク・エディットのステップのためのジョブ制御言語 (JCL) を、サポートするプログラム言語ごとに、個別のカタログ式プロシージャーで提供します。CICS をインストールした後に、CICSTS41.CICS.SDFHPROC ライブラリーにインストールされたプロシージャーを、プロシージャー・ライブラリーにコピーする必要があります。

各プロシージャー名は、DFH*wx*TyL という形式をとります。ここで、変数 w、x、および y は、プログラムのタイプ (EXCI バッチまたは CICS オンライン)、コンパイラーのタイプ、およびプログラム言語に応じて決まります。上記の命名規則を使用して、プロシージャー名が 139 ページの表 8 で示すように指定されます。

表 8. アプリケーション・プログラムをインストールするためのプロシージャー

言語	言語環境プログラム準拠のコンパイラー			非言語環境プログラム準拠のコンパイラー	
	スタンドアロンの変換プログラム	EXCI	統合変換プログラム	スタンドアロンの変換プログラム	EXCI
アセンブラー	-	-	-	DFHEITAL	DFHEXTAL
C	DFHYITDL (注 1 を参照)	DFHYXTDL	DFHZITDL (注 1 を参照)	-	-
XPLINK コンパイラー・オプションを使用した C	DFHYITFL (注 2 を参照)	-	DFHZITFL (注 1 を参照)	-	-
C++	DFHYITEL (注 1 を参照)	DFHYXTEL	DFHZITEL (注 1 を参照)	-	-
XPLINK コンパイラー・オプションを使用した C++	DFHYITGL (注 2 を参照)	-	DFHZITGL (注 1 を参照)	-	-
COBOL (注 4 を参照)	DFHYITVL	DFHYXTVL (注 3 を参照)	DFHZITCL (注 2 を参照)	-	-
PL/I (注 5 を参照)	DFHYITPL (注 2 を参照)	DFHYXTPL	DFHZITPL (注 2 を参照)	-	-

注:

1. **COMPILER** パラメーターで C コンパイラーの正しい名前を指定していれば、C でも DFHYITEL を使用できます。
2. 生成されたモジュールの出力ライブラリーは、(PDS ではなく) PDSE です。
3. 統合 CICS 変換プログラムの使用時には変換プログラム・オプションが無視されるため、EXCI COBOL プログラムには個別の変換プログラム・ステップを使用する必要があります。
4. DFHZITCL は、統合 CICS 変換プログラムを含むバージョンの Enterprise COBOL コンパイラーを使用するため、COBOL モジュールのコンパイルで推奨されるプロシージャーです。ただし、COBOL プログラムが EXCI オプションを使用するバッチ処理用である場合は、統合変換プログラムを使用することはできません。
5. DFHZITPL は、統合 CICS 変換プログラムを含むバージョンの Enterprise PL/I コンパイラーを使用するため、PL/I モジュールのコンパイルで推奨されるプロシージャーです。ただし、PL/I プログラムが EXCI オプションを使用するバッチ処理用である場合は、統合変換プログラムを使用することはできません。
6. Language Environment (IMS™ ルーチン) の元のバッチ環境で EXEC DLI コマンドを実行するプログラムでは、以下の特殊なプロシージャーを使用します。

DFHYBTPL
PL/I アプリケーション・プログラム

DFHYBTVL
COBOL アプリケーション・プログラム

ロード・ライブラリーの 2 次エクステントへのプログラムのインストール

CICS は、CICS の実行時に作成されるロード・ライブラリーの 2 次エクステントをサポートしています。1 次および 2 次エクステントを持つ DFHRPL または動的 LIBRARY 連結でライブラリーを定義し、CICS の実行中に、2 次エクステントがリンク・エディットの結果としてロード・ライブラリーに追加される場合、CICS ローダーは、そのライブラリーのオカレンスを検出し、クローズ、再オープンを行います。つまり、プログラムを新しくコピーしたことによって新しいライブラリー・エクステントが発生した場合でも、CEMT NEWCOPY コマンドを使用して新しいバージョンを導入できるという意味です。

ただし、これによって、モジュールを 2 次エクステントからロードする際の検索時間が増加する可能性があります。可能であれば、2 次エクステントの使用は避けてください。

注: DFHXITPL を使用している場合、バインダー・ステップでの SYSLMOD DD ステートメントは、(古いバージョンの PL/I コンパイラー用の PDS ではなく) PDSE を参照している必要があります。

CICS 提供インターフェース・モジュールの組み込み

アプリケーション・プログラムで、CPI コミュニケーションまたは SAA® リソース・リカバリーを使用する場合は、該当するインターフェース・モジュールをそのプログラムで使用できるようにします。

ユーザーのオンライン・アプリケーション・プログラムを CICS ライブラリーにインストールするための CICS 提供プロシージャーは、該当する言語の EXEC インターフェース・モジュールの INCLUDE ステートメントを含む CICS ライブラリー・メンバーを指定します。例えば、DFHYITVL プロシージャーは以下のステートメントを使用します。

```
//COPYLINK EXEC PGM=IEBGENER,COND=(7,LT,COB)
//SYSUT1   DD DSN=&INDEX..SDFHSAMP(&STUB),DISP=SHR
//SYSUT2   DD DSN=&&COPYLINK,DISP=(NEW,PASS),
//            DCB=(LRECL=80,BLKSIZE=400,RECFM=FB),
//            UNIT=&WORK,SPACE=(400,(20,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN    DD DUMMY

//SYSLIN   DD DSN=&&COPYLINK,DISP=(OLD,DELETE)
//         DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
```

この COBOL の例で、シンボリック・パラメーター STUB のデフォルトは DFHEILID です。 DFHEILID メンバーには、INCLUDE SYSLIB(DFHELII) というステートメントが含まれています。

PL/I および C 用に提供されたプロシージャーも、DFHEILID を参照します。すなわち、DFHELII スタブが使用されます。

アプリケーション・プログラムで CPI コミュニケーションまたは SAA リソース・リカバリー機能を使用する場合は、以下のいずれかを実行してください。

- アプリケーション・プログラムをインストールする CICS 提供プロシージャーの呼び出しに使用するジョブで、LKED.SYSIN 指定変更に、適切な INCLUDE ステートメントを追加する。次の INCLUDE ステートメントを追加します。
  - INCLUDE SYSLIB(DFHCPLC) (プログラムで CPI コミュニケーションを使用する場合)
  - INCLUDE SYSLIB(DFHCPLRR) (プログラムで SAA リソース・リカバリーを使用する場合)

リンク・エディット・ステップ中に、DFHEI1 項目の重複定義を示す警告メッセージが表示されることがあります。これらのメッセージは無視してかまいません。

リンク・エディット要件の詳細については、 153 ページの『ユーザー独自のジョブ・ストリームの使用』を参照してください。

アセンブラー言語アプリケーション・プログラムのインストール

アセンブラー言語で作成されたアプリケーション・プログラムを変換、アセンブル、およびリンク・エディットするには、DFHEITAL または DFHEXTAL プロシージャーを使用することができます。

このタスクについて

図 13 で示されるジョブ制御ステートメントの例を使用すると、アセンブラー言語で作成されたアプリケーション・プログラムを処理することができます。プロシージャー名の中の「x」は、プログラムが CICS アプリケーション・プログラムと EXCI バッチ・プログラムのいずれであるかによって決まります。CICS 提供プロシージャーの名前については、 139 ページの表 8を参照してください。

```
//jobname      JOB     accounting info,name,MSGLEVEL=1
//             EXEC    PROC=DFHExTAL                        1
//TRN.SYSIN    DD      *
*ASM      XOPTS(translator options . . .)                  2
               .
               assembler-language source statements
               .
/*
//LKED.SYSIN   DD      *
               NAME    anyname(R)
/*
//
```

ここで、anyname はユーザーのロード・モジュール名です。

図 13. DFHExTAL プロシージャーを呼び出すジョブ制御ステートメントの例

注:

1 どちらかの読み取り専用 DSA にプログラムをインストールする場合の詳細については、 134 ページの『RDSA にあるアプリケーション・プログラムの実行』を参照してください。

LPA から使用されるプログラムをインストールする場合は、次のオプションを追加してください。

- DFHEITAL プロシージャーの ASM ステップで、EXEC ステートメントの PARM オプションに RENT を追加する
- DFHEITAL プロシージャー呼び出しの LNKPARM パラメーターに、RENT オプションおよび REFR オプションを追加する

(133 ページの『リンク・パック域にあるアプリケーションの実行』を参照してください。)

2 XOPTS ステートメントに組み込むことのできる変換プログラム・オプションの詳細については、 90 ページの『変換プログラムのオプションの定義』を参照してください。

143 ページの図 14 は、コマンド・レベル変換プログラムによってアセンブラー・ソース・プログラムが処理され、変換プログラム・リストと出力ファイルが生成されるしくみを示しています。この出力ファイルは、続いてアセンブラーが CICS.SDFHMAC への参照を使用して処理し、アセンブラー・リストともう 1 つの出力ファイルが生成されます。その後、この出力ファイルを、リンケージ・エディターが CICS.SDFHLOAD への参照を使用して処理し、リンケージ・エディター・リストと CICS.SDFHLOAD に保管されるロード・モジュールが生成されます。

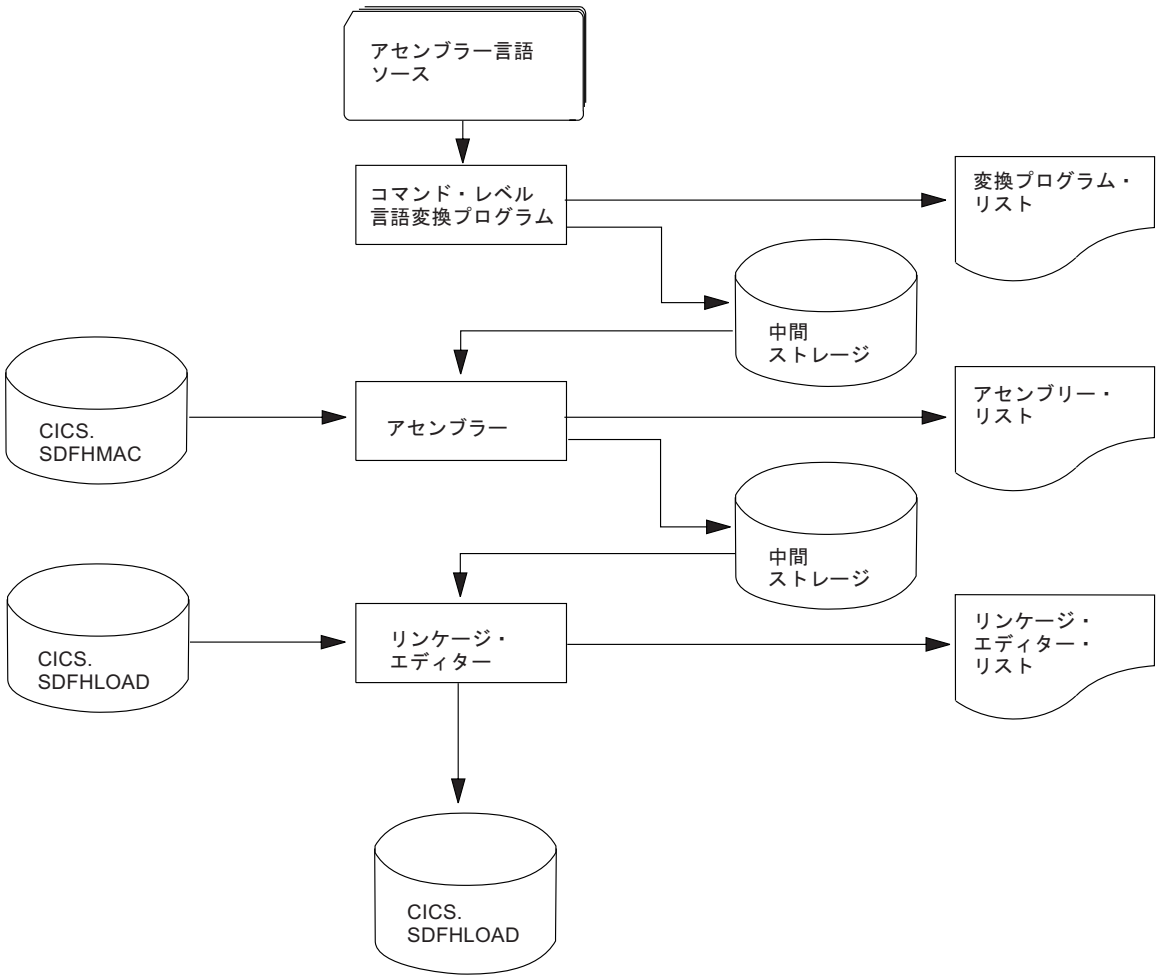


図 14. DFHEITAL プロシージャーを使用したアセンブラー言語プログラムのインストール

COBOL アプリケーション・プログラムのインストール

144 ページの図 15 は、独立した変換プログラム・ステップを必要とする COBOL プログラムおよび PL/I プログラム用の、カタログ式プロシージャーの制御のフローについて説明しています。統合変換プログラムを使用する場合は、個別の変換プログラムのステップはありません。 高水準言語ソースおよび CICS.SDFHLOAD の両方がコンパイラーに入力され、変換プログラムとコンパイラーの結合されたリストが生成されます。

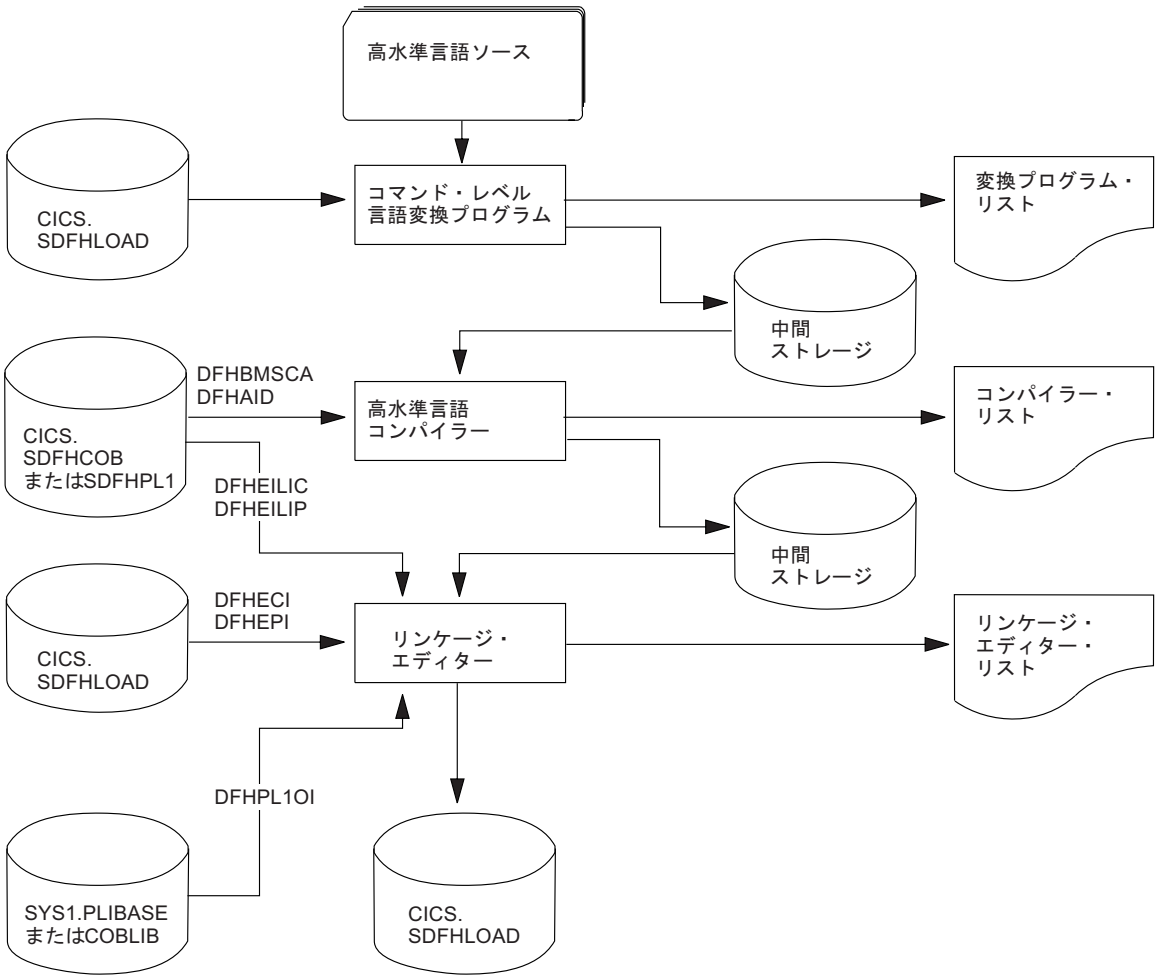


図 15. COBOL および PL/I プログラムのインストール

COBOL アプリケーション・プログラムをインストールする JCL の例

145 ページの図 16 に示されているジョブ制御ステートメントを使用して、単独の変換プログラムで COBOL アプリケーション・プログラムを処理することができます。プロシージャー名は、それが CICS アプリケーション・プログラムであるか、または EXCI バッチ・プログラムであるかによって決まります。CICS 提供 COBOL プロシージャーの名前については、 139 ページの表 8を参照してください。

```
//jobname    JOB  accounting info,name,MSGLEVEL=1
//           EXEC PROC=procname                                    1
//TRN.SYSIN  DD      *                                          2
 CBL   XOPTS(Translator options . . .)                          3
              .
             COBOL source statements
              .
/*
//LKED.SYSIN DD      *                                          4
               NAME    anyname(R)
/*
//
```

ここで、*procname* はプロシージャーの名前であり、anyname はユーザーのロード・モジュール名です。

図 16. DFHYITVL または DFHYXTVL プロシージャーを呼び出すジョブ制御ステートメントの例

プロシージャー DFHZITCL を使用して統合変換プログラムを呼び出すには、以下の 図 17 に示されるようなジョブ制御ステートメントを使用できます。

```
 //jobname    JOB  accounting info,name,MSGLEVEL=1
 //           EXEC DFHZITCL,PROGLIB=dsnname                  1
 //COBOL.SYSIN  DD *
    .
    . COBOL source statements
    .
 /*
 //LKED.SYSIN DD *
             NAME anyname(R)
 /*
 //
```

ここで、anyname はユーザーのロード・モジュール名です。

図 17. DFHZITCL プロシージャーを呼び出すジョブ制御ステートメントの例

COBOL プログラムのインストールについての注意

1 **変換プログラムのオプション:**

コンパイル・ステップで必要な COBOL 機能のバージョンに従い、COBOL3 または COBOL2 の変換プログラム・オプションを指定します。

s

コンパイラー・オプション:

COBOL プログラムをコンパイルするには、コンパイラー・オプション RENT、NODYNAM、および LIB が必要です。

変換プログラム・オプション CBLCARD (デフォルト) を使用する場合は、CICS 変換プログラムが、これらのオプションを含む CBL ステートメントを自動的に生成します。変換プログラム・オプション NOCBLCARD を指定すると、CBL または PROCESS カードの生成を避けることができます。

CICS 提供の COBOL プロシージャーにある COB ステップの PARM ステートメントにより、コンパイラー・オプションの各値が指定されます。例えば、次のようにします。

```
//COB    EXEC PGM=IGYCRCTL,REGION=&REG,
//       PARM='NODYNAM,LIB,OBJECT,RENT,APOST,MAP,XREF'
```

統合変換プログラムを備えたコンパイラーで COBOL プログラムをコンパイルするには、CICS コンパイラー・オプションを使用し、コンパイラーが変換プログラムを呼び出すよう指示する必要があります。DFHZITCL プロシージャーには、以下のようにこのコンパイラー・オプションが含まれています。

```
CBLPARM='NODYNAM,LIB,MAP,CICS(''COBOL3'')'
```

注: 統合変換プログラムに対して PARM ストリング中で CICS 変換プログラム・オプションを指定する場合は、この例が示すように二重のアポストロフィを付ける必要があります。ただし、ソース・プログラム内でこれらのオプションを指定する場合は、単一アポストロフィを使用します (例えば、ソース・プログラム内の CBL ステートメントを、`CBL CICS('COBOL3,SP') APOST` のように指定します)。

CICS 提供の COBOL プロシージャーでは、SIZE オプションおよび BUF オプションの値を指定しません。このデフォルトは、SIZE=MAX と BUF=4K です。SIZE は、コンパイラーが使用可能な仮想記憶域の量を定義し、BUF は、それぞれのコンパイラーのバッファー作業ファイルに割り振られる動的ストレージの量を定義します。これらのオプションは、プロシージャーを呼び出す EXEC ステートメントの PARM.COB パラメーターを使用すれば変更できます。例:

```
EXEC  PROC=procname,PARM.COB='SIZE=512K,BUF=16K,.,.,.'
```

以下の方法のいずれかを使用すれば、コンパイラー・オプションを変更することができます。

- CICS 提供の COBOL プロシージャーの COB ステップで定義された PARM ステートメントを指定変更する。

  このプロシージャーを呼び出すジョブの PARM ステートメントを指定すると、それによって、このプロシージャーの JCL で指定された**すべて**のオプションが指定変更されます。必要なオプションすべてが、指定変更または CBL ステートメントで指定されていることを確認してください。
- CICS 提供の COBOL プロシージャーの呼び出しに使用されるジョブ・ストリームのソース・ステートメントの先頭で、CBL ステートメントを指定する。
- インストールされている COBOL で、マクロ IGYCOPT をデフォルト指定する。これが必要なのは、CBL ステートメントを使用していない場合 (すなわち、変換プログラム・オプション 98 ページの『NOCBLCARD』 を指定した場合) です。

変換プログラム・オプション CBLCARD|NOCBLCARD について詳しくは、90 ページの『変換プログラムのオプションの定義』を参照してください。 98 ページの『NOCBLCARD』 オプションの使用を選択する場合は、COBOL コンパイラー・オプション ALOWCBL=NO も指定して、エラー・メッセージ IGYOS4006-E が発行されるのを防いでください。ALOWCBL コンパイラー・オプションについて詳しくは、ご使用の COBOL のバージョンに対応する「*インストールおよびカスタマイズ*」のマニュアルを参照してください。

2 変換プログラムに対する入力がない場合は、DD * ではなく DD DUMMY を指定することができます。ただし、DD DUMMY を指定する場合は、適切な DCB オペランドもコーディングしてください。(変換プログラムは、SYSIN データ・セットのすべてのデータ制御ブロック情報を提供するわけではありません。)

3 CICS TS で提供される独立型の変換プログラムを使用する場合は、XOPTS ステートメントにある変換プログラム・オプションにより、CICS 提供の COBOL プロシージャー内の類似のオプションが指定変更されます。

XOPTS ステートメントに組み込み可能な変換プログラム・オプションの詳細については、90 ページの『変換プログラムのオプションの定義』を参照してください。

統合 CICS 変換プログラムを使用する場合、COBOL コンパイラーでは、XOPTS ではなく、変換プログラム・オプションを定義するキーワード CICS のみが認識されます。

4 リンク・エディットで解決されない弱い外部参照は無視できます。

リンク・エディット・ジョブ・ステップは、CICS の環境特有のモジュールが入ったライブラリーにアクセスする必要があり、さらに必要に応じて、言語環境プログラムのリンク・エディット・モジュールの入ったライブラリーにもアクセスします。モジュールおよびライブラリー・サブルーチンが異なる名前を使用してライブラリーにインストール済みの場合、これらのライブラリー名を指定変更、または変更します。

読み取り専用 DSA のいずれかにプログラムをインストールする場合について詳しくは、134 ページの『RDSA にあるアプリケーション・プログラムの実行』を参照してください。

LPA から使用されるプログラムをインストールする場合は、CICS 提供の COBOL プロシージャーの呼び出しにおいて、LNKPARM パラメーターに RENT オプションおよび REFR オプションを追加します。(133 ページの『リンク・パック域にあるアプリケーションの実行』を参照してください。)

PL/I アプリケーション・プログラムのインストール

144 ページの図 15 は、PL/I プログラム用の、カタログ式プロシージャーの制御のフローについて説明しています。

PL/I プログラムの作成について詳しくは、「*PL/I Programming Guide*」を参照してください。

PL/I アプリケーション・プログラムをインストールする JCL の例

148 ページの図 18 に示されているジョブ制御ステートメントを使用して、単独の変換プログラムで PL/I アプリケーション・プログラムを処理することができます。

プロシージャー名内の値「x」は、それが CICS アプリケーション・プログラムであるか、または EXCI バッチ・プログラムであるかによって決まります。CICS 提供

プロシージャーの名前については、 139 ページの表 8を参照してください。

```
//jobname    JOB  accounting info,name,MSGLEVEL=1
//           EXEC PROC=DFHYxTPL                                    1
//TRN.SYSIN  DD      *                                            2
*PROCESS    XOPTS(translator options...)PL/I compiler options...; 3
             .
            PL/I source statements                                4
              .
/*
//LKED.SYSIN DD      *                                            5
              NAME    anyname(R)
/*
//
```

ここで、anyname はユーザーのロード・モジュール名です。

図 18. DFHYxTPL プロシージャーを呼び出すジョブ制御ステートメントの例

PL/I プログラムのインストール時の注意

1. PL/I COUNT ランタイム・オプションは、言語環境ではサポートされていません。REPORT オプションは、言語環境プログラム・オプション RPTSTG および RPTUPTS に置き換えられます。「*z/OS Language Environment Migration Guide*」を参照してください。
2. 変換プログラムに対する入力がない場合は、DD * の代わりに DD DUMMY を指定できます。ただし、DD DUMMY を指定する場合は、適切な DCB オペランドもコーディングしてください。(変換プログラムは、SYSIN データ・セットのすべてのデータ制御ブロック情報を提供するわけではありません。)
3. **変換プログラムおよびコンパイラー・オプション:** XOPTS ステートメントに組み込み可能な変換プログラム・オプションについて詳しくは、 90 ページの『変換プログラムのオプションの定義』を参照してください。

   PL/I コンパイラーからのメッセージ「IEL0548I PARAMETER TO MAIN PROCEDURE NOT VARYING CHARACTER STRINGは無視してください。

   PL/I コンパイラーから、引数とパラメーターがプロシージャー DFHxxxx の呼び出しで一致しないことを述べる警告メッセージが表示されることがあります。これらのメッセージは、オペランドで指定した CICS コマンドへの引数のデータ型が正しくないことを示します。これらのメッセージで記載されたすべてのフィールド、特に **receiver** フィールドを注意深くチェックしてください。
4. CALL PLIDUMP ステートメントをアプリケーション・プログラムに組み込む場合、出力は CESE 一時データ宛先に送信されます。CSD 内の CICS 提供リソース定義グループ DFHDCTG には、CESE の項目が含まれます。
5. **リンク・エディットの考慮事項:** リンク・エディットでは未解決の弱い外部参照を無視できます。

   読み取り専用 DSA のいずれかにプログラムをインストールする場合について詳しくは、 134 ページの『RDSA にあるアプリケーション・プログラムの実行』を参照してください。

   LPA から使用されるプログラムをインストールする場合は、RENT および REFR オプションを、DFHYxTPL プロシージャー呼び出しの LNKPARM パラ

メーターに追加します。(詳しくは、 133 ページの『リンク・パック域にあるアプリケーションの実行』を参照してください。)

PL/I プロシージャーと統合変換プログラム

新規プロシージャー DFHZITPL を使用して統合変換プログラムを呼び出すために、以下のサンプル JCL を使用することができます。

```
//jobname    JOB  accounting info,name,MSGLEVEL=1
//           EXEC DFHZITPL,PROGLIB=dsnname                    1
//PLI.SYSIN  DD *
   .
   . PLI source statements
   .
/*
//LKED.SYSIN DD *
            NAME anyname(R)
/*
//
```

ここで、anyname はユーザーのロード・モジュール名です。

図 19. DFHZITPL プロシージャーを呼び出すジョブ制御ステートメントの例

統合変換プログラムを使用して PLI プログラムのインストールする場合の注意

1. DFHZITPL プロシージャーは、コンパイラーに変換プログラムを呼び出させることを指示する以下のコンパイラー・オプションを組み込みます。

   ```
   PLIPARM=('SOURCE,OPTIONS,SYSTEM(CICS),PP(CICS)')
   ```

   注: このプロシージャーで、LKED ステップでの SYSLMOD DD ステートメントは、(古いバージョンの PL/I コンパイラー用の PDS ではなく) PDSE を参照している必要があります。

C アプリケーション・プログラムのインストール

150 ページの図 20 は、独立した変換プログラム・ステップを必要とする C コマンド・レベル・プログラム用の、DFHYxTzL カタログ式プロシージャーの制御のフローについて説明しています。統合変換プログラムを使用する場合は、個別の変換プログラムのステップはありません。高水準言語ソースおよび CICS.SDFHLOAD の両方がコンパイラーに入力され、変換プログラムとコンパイラーの結合されたリストが生成されます。

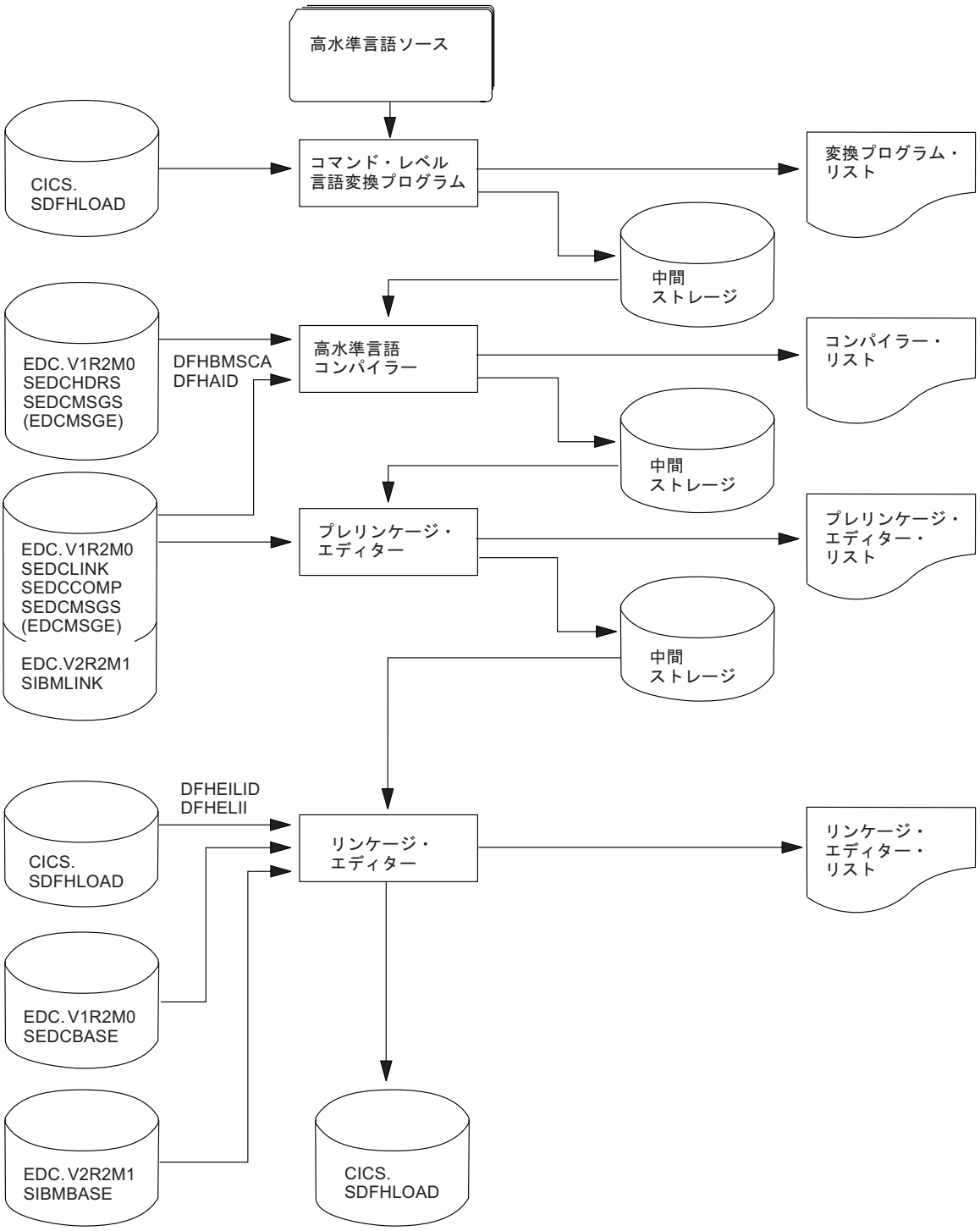


図 20. DFHYxTzL プロシージャーを使用した C プログラムのインストール

変換プログラム、コンパイラー、プレリンケージ・エディター、およびリンケージ・エディターの各ステップがあり、それぞれで、リストと次のステップに渡される中間ファイルを生成します。C ライブラリーは、コンパイラー、プレリンケージ・エディター、およびリンケージ・エディターの各ステップで参照されます。

注: XPLINK コンパイラー・オプションを選択する場合は、上の図のプリリンクのステップがありません。

C プログラムのインストールは、あらかじめ C ライブラリーおよびコンパイラーがインストール済みで、さらに C の CICS サポートが生成済みの場合に可能です (「*CICS Transaction Server for z/OS インストール・ガイド*」を参照してください)。

C アプリケーション・プログラムをインストールする JCL の例

C アプリケーション・プログラムを処理するには、図 21 に示されているジョブ制御ステートメントを使用できます。プロシージャー名の中の x は、プログラムが CICS アプリケーション・プログラムか、または EXCI バッチ・プログラムかによって決まります。CICS 提供プロシージャーの名前については、 139 ページの表 8を参照してください。

```
//jobname    JOB  accounting info,name,MSGLEVEL=1
//           EXEC PROC=DFHYxTzL                                  1
//TRN.SYSIN  DD      *                                          2
#pragma XOPTS(Translator options . . .)                        3
              .
             C source statements
              .
/*
//LKED.SYSIN DD      *                                          4
               NAME    anyname(R)
/*
//
```

ここで、anyname はユーザーのロード・モジュール名です。

図 21. DFHYxTzL プロシージャーを呼び出す JCL の例

C プログラムのインストール時の注意

1. **コンパイラー・オプション:** プロシージャーを呼び出す EXEC ステートメントのパラメーター指定変更 (PARM.C)、または ~pragma オプション・ディレクティブを使用すると、コンパイラー・オプションをコーディングすることができます。
2. 変換プログラムに対する入力がない場合は、DD * の代わりに DD DUMMY を指定できます。ただし、DD DUMMY を指定する場合は、適切な DCB オペランドもコーディングしてください (変換プログラムは、SYSIN データ・セットのすべてのデータ制御ブロック情報を提供するわけではありません)。
3. **変換プログラム・オプション:** XOPTS ステートメントに組み込み可能な変換プログラム・オプションについては、 90 ページの『変換プログラムのオプションの定義』を参照してください。
4. 読み取り専用 DSA のいずれかにプログラムをインストールする場合について詳しくは、 134 ページの『RDSA にあるアプリケーション・プログラムの実行』を参照してください。

   LPA から使用されるプログラムをインストールする場合は、RENT および REFR オプションを、DFHYxTzL プロシージャー呼び出しの LNKPARM パラメーターに追加します (詳しくは、 133 ページの『リンク・パック域にあるアプリケーションの実行』を参照してください)。

C 言語のプログラムは AMODE(31) を指定してリンク・エディットし、デフォルトで DFHYxTzL プロシージャーが AMODE(31) を指定するようにする必要があります。

XL C に対する 統合 CICS 変換プログラムの呼び出し

XL C のためにプロシージャーを使用して統合変換プログラムを呼び出すには、以下の 図 22 に示されるようなジョブ制御ステートメントを使用できます。

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC DFHZITxL,PROGLIB=dsnname                  1
//C.SYSIN DD *
   .
   . C source statements
   .

//LKED.SYSIN DD *
           NAME anyname(R)

//
```

ここで、anyname はユーザーのロード・モジュール名です。

図 22. XL C の統合変換プログラムのための JCL サンプル

1. **変換プログラム名:**XPLINK のない C プログラムには DFHZITDL を指定し、XPLINK のある C プログラムには、DFHZITFL を指定します。

XL C に対する統合 CICS 変換プログラムの呼び出し

XL C++ のためにプロシージャーを使用して統合変換プログラムを呼び出すには、以下の 図 23 に示されるようなジョブ制御ステートメントを使用できます。

```
//jobname JOB accounting info,name,MSGLEVEL=1
// EXEC DFHZITxL,PROGLIB=dsnname                  1
//CPP.SYSIN DD *
   .
   . C++ source statements
   .

//LKED.SYSIN DD *
           NAME anyname(R)

//
```

ここで、anyname はユーザーのロード・モジュール名です。

図 23. XL C++ の統合変換プログラムのための JCL サンプル

1. **変換プログラム名:**XPLINK のない C++ プログラムには DFHZITEL を指定し、XPLINK のある C++ プログラムには、DFHZITGL を指定します。

C ソース・コードへの事前変換済みコードの組み込み

このタスクについて

変換プログラムは、dfhexec または DFHEXEC を生成することがあります。両方のバージョンがプログラムに存在すると、エラー・メッセージ IEW2456E が表示されます。このエラーを防ぐ方法は 2 つあります。

1. dfhexec を含む旧コードを再コンパイルする。
2. 以下で示すように、プリリンカーの RENAME 制御ステートメントをジョブで使用する。

```
//jobname    JOB  accounting info,name,MSGLEVEL=1
//           EXEC PROC=DFHYxTzL
//TRN.SYSIN  DD      *
#pragma XOPTS(Translator options . . .)
              .
             C source statements
              .
/*
//PLKED.SYSLIN DD      *
         RENAME dfhexec DFHEI1
//LKED.SYSLIN DD      *
         NAME anyname(R)
/*
//
```

ここで、anyname はユーザーのロード・モジュール名です。

図 24. dfhexec を名前変更する JCL の例

ユーザー独自のジョブ・ストリームの使用

アプリケーション・プログラムを変換、アセンブル (またはコンパイル)、およびリンク・エディットするユーザー独自の JCL を作成したい場合、提供されるカタログ式プロシージャーをモデルとして使用することができます。これらのカタログ式プロシージャーは、CICSTS41.CICS.SDFHPROC ライブラリーにインストールされています。

このセクションの残りの部分では、変換プログラムの重要な点、およびプログラムの主要なカテゴリーそれぞれについて要約します。わかりやすくするため、以下の説明では、プログラムを CICSTS41.CICS.SDFHLOAD または IMS.PGMLIB にロードすることにします。実際には、ライブラリーが CICS ジョブ・ストリームの DFHRPL または動的 LIBRARY 連結に組み込まれているか、バッチ・ジョブ・ストリーム (独立型 IMS バッチ・プログラムの場合) の STEPLIB ライブラリー連結に組み込まれている場合にのみ、すべてのライブラリーを使用することができます。

注: ジョブ・ストリームで参照される IMS ライブラリーは、IMS.libnam (例えば、IMS.PGMLIB) によって識別されます。IMS ライブラリーにユーザー独自の命名規則を使用する場合は、それに従って IMS ライブラリーを名前変更してください。

変換プログラムの要件

CICS 変換プログラムは、256KB の仮想記憶が最低限必要です。変換プログラム・オプション CICS および DLI の使用が必要になる場合があります。

EXEC CICS または EXEC DLI コマンドを使用するオンライン・プログラム

1. 常に、変換プログラム・オプション CICS を使用する。プログラムが EXEC DLI コマンドを実行する場合は、変換プログラム・オプション DLI を使用します。
2. リンク・エディット入力 (SYSLIN DD ステートメントにより定義される) は、オブジェクト・デックの**前に**、正しいインターフェース・モジュールを組み込む必要があります。したがって、インターフェース・モジュールの INCLUDE ステートメントは、オブジェクト・デックの前に置きます。さらに、ORDER ステートメントを INCLUDE ステートメントの前に、また ENTRY ステートメントはすべての INCLUDE ステートメントの後に書き込みます。

   インターフェース・モジュールは、以下のとおりです。

   DFHEAI
   アセンブラー

   DFHELII
   すべての HLL 言語

   CICS 提供プロシージャーでは、リンク・エディット・ステップへの入力 (SYSLIN DD ステートメントで定義されます) によって、オブジェクト・デックを持つライブラリー・メンバーを連結します。このメンバーには、必要なインターフェース・モジュールに対する INCLUDE ステートメントが含まれています。例えば、DFHYITVL プロシージャーは、以下の INCLUDE ステートメントを含むライブラリー・メンバー DFHEILID を連結します。

   ```
   INCLUDE SYSLIB(DFHELII)
   ```
3. リンク・エディットからの出力であるロード・モジュール (SYSLMOD DD ステートメントで定義される) を、CICSTS41.CICS.SDFHLOAD またはユーザー独自のプログラム・ライブラリーに入れる。

155 ページの図 25 は、COBOL アプリケーション・プログラムのインストールに使用可能な、CICS 提供プロシージャー DFHYITVL を基にした JCL 例とインライン・プロシージャーを示しています。このプロシージャーは、COPYLINK ステップと、必要なインターフェース・モジュールに対する INCLUDE ステートメントを含んだライブラリー・メンバー DFHEILID の連結を、(DFHYITVL プロシージャーが組み込んでいるように) 組み込んでいません。代わりに、JCL が以下の INCLUDE ステートメントを提供します。

```
INCLUDE SYSLIB(DFHELII)
```

このステートメントが提供されなかった場合、リンク・エディットは未解決の外部参照に対するエラー・メッセージを返し、プログラムの出力は実行可能でないとしてマークされます。

```
//*      The following JCL could be used to execute this procedure
//*
//APPLPROG EXEC MYYITVL,
//       INDEX='CICSTS41.CICS
//       PROGLIB='CICSTS41.CICS.SDFHLOAD',
//       DSCTLIB='CICSTS41.CICS.SDFHCOB',
//       INDEX2='user.qualif'
//       OUTC=A,                      Class for print output
//       REG=4M,                      Region size for all steps
//       LNKPARM='LIST,XREF',         Link edit parameters
//       WORK=SYSDA                   Unit for work datasets

//TRN.SYSIN  DD *
//*         .
//*         . Application program
//*         .
//*
//LKED.SYSIN DD *
        INCLUDE SYSLIB(DFHELII)
        NAME anyname(R)
//*
//MYYITVL PROC SUFFIX=1$,           Suffix for translator module
//       INDEX='CICSTS41.CICS', Qualifier(s) for CICS libraries
//       PROGLIB='CICSTS41.CICS.SDFHLOAD', Name of o/p library
//       DSCTLIB='CICSTS41.CICS.SDFHCOB',  Private macro/dsect
//       AD370HLQ='SYS1',             Qualifier(s) for AD/Cycle compiler
//       LE370HLQ='SYS1',             Qualifier(s) for Language Environment libraries
//       OUTC=A,                      Class for print output
//       REG=4M,                      Region size for all steps
//       LNKPARM='LIST,XREF',         Link edit parameters
//       WORK=SYSDA                   Unit for work datasets
//*
//*      This procedure contains 3 steps
//*      1.   Exec the COBOL translator (using the supplied suffix 1$)
//*      2.   Exec the COBOL compiler
//*      3.   Linkedit the output into dataset &PROGLIB

//TRN    EXEC PGM=DFHECP &SUFFIX,,
//            PARM='COBOL3',
//            REGION=&REG

//STEPLIB  DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPUNCH DD DSN=&&SYSCIN,
//            DISP=(,PASS),UNIT=&WORK,
//            DCB=BLKSIZE=400,
//            SPACE=(400,(400,100))
//*
//COB    EXEC PGM=IGYCRCTL,REGION=&REG,
//       PARM='NODYNAM,LIB,OBJECT,RENT,APOST,MAP,XREF'
//STEPLIB  DD DSN=&AD370HLQ..SIGYCOMP,DISP=SHR
//SYSLIB   DD DSN=&DSCTLIB,DISP=SHR
//         DD DSN=&INDEX..SDFHCOB,DISP=SHR
//         DD DSN=&INDEX..SDFHMAC,DISP=SHR
//         DD DSN=&INDEX..SDFHSAMP,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN    DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),
//            UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT4   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6   DD UNIT=&WORK,SPACE=(460,(350,100))
//*
//LKED   EXEC PGM=IEWL,REGION=&REG,
//            PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB   DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//         DD DSN=&LE370HLQ..SCEELKED,DISP=SHR
//SYSLMOD  DD DSN=&PROGLIB,DISP=SHR
//SYSUT1   DD UNIT=&WORK,DCB=BLKSIZE=1024,
//            SPACE=(1024,(200,20))
```

CALL DLI インターフェースを使用するオンライン・プログラム

1. 変換プログラム・オプション CICS を指定するが、変換プログラム・オプション DLI は指定しない。

   注: CICS コマンドを使用せず、実行中のトランザクションからのみ呼び出される (しかも、CICS タスク開始によって直接起動されることが決してない) プログラムの場合は、変換プログラム・ステップは必要ありません。
2. インターフェース・モジュール DFHDLIAI は、自動的にリンク・エディットによって組み込まれる。リンク・エディットの入力で INCLUDE ステートメントを使用する場合は、それを、オブジェクト・デックの**後**に置きます。
3. コピーブック DLIUIB をプログラムに組み込む。
4. リンク・エディットからの出力であるロード・モジュール (SYSLMOD DD ステートメントで定義される) を、CICSTS41.CICS.SDFHLOAD またはユーザー定義のアプリケーション・プログラム・ライブラリーに入れる。

EXEC DLI コマンドを使用するバッチまたは BMP プログラム

1. 変換プログラム・オプション DLI が必須。変換プログラム・オプション CICS は指定しないでください。
2. インターフェース・モジュールに対する INCLUDE ステートメントは、リンク・エディットへの入力 (SYSLIN DD ステートメントで定義される) のオブジェクト・デックの**後に続ける**必要がある。IMS.RESLIB に常駐するインターフェース・モジュール DFSLI000 は、すべてのプログラム言語に対して同一です。CICSTS41.CICS.SDFHLOAD をリンク・エディットの入力 (SYSLIB DD ステートメントで定義されます) に組み込む場合は、IMS.RESLIB の**後**に連結します。
3. リンク・エディットからの出力であるロード・モジュール (SYSLMOD DD ステートメントで定義される) を、IMS.PGMLIB、またはバッチ・ジョブ・ストリームの STEPLIB DD ステートメントに連結されたライブラリーに入れる。

DL/I CALL コマンドを使用するバッチまたは BMP プログラム

DL/I CALL インターフェースを使用するアセンブラー、COBOL、または PL/I のプログラムを準備する場合は、CICS 提供プロシージャーを使用しないでください。CALL ASMTDLI、CALL CBLTDLI、または CALL PLITDLI を含むプログラムは、IMS アプリケーションとしてアセンブルまたはコンパイル、そしてリンク・エディットする必要があり、CICS 要件の対象ではありません。DL/I CALL インターフェースを使用するアプリケーション・プログラムの準備方法の詳細については、関連する IMS のマニュアルを参照してください。

第 9 章 マップ・セットおよび区分セットのインストール

このセクションでは、CICS の基本マッピング (BMS) 機能で使用するために、マップ・セットおよび区分セットのアセンブルとリンク・エディットを行う方法について説明します。また、BMS マップから生成された HTML テンプレートのインストール方法についても説明します。

ユーザー・プログラムが BMS マップを使用する場合には、そのマップを作成する必要があります。これを行う従来方式は、BMS マクロでこのマップをコーディングし、それをアセンブルすることです。異なる出力オプションを使用して二度アセンブルします。

- 一度目のアセンブルでは、一組の定義が作成される。適切な言語ステートメントを使用して、この定義をユーザー・プログラムの中にコピーし、その定義によって、マップ内のフィールドを名前で参照することができます。
- 二度目のアセンブルでは、ユーザー・プログラムの実行時に使用するオブジェクト・モジュールが作成される。

このプロセスについて、以下の図で説明します。

© Copyright IBM Corp. 1989, 2009

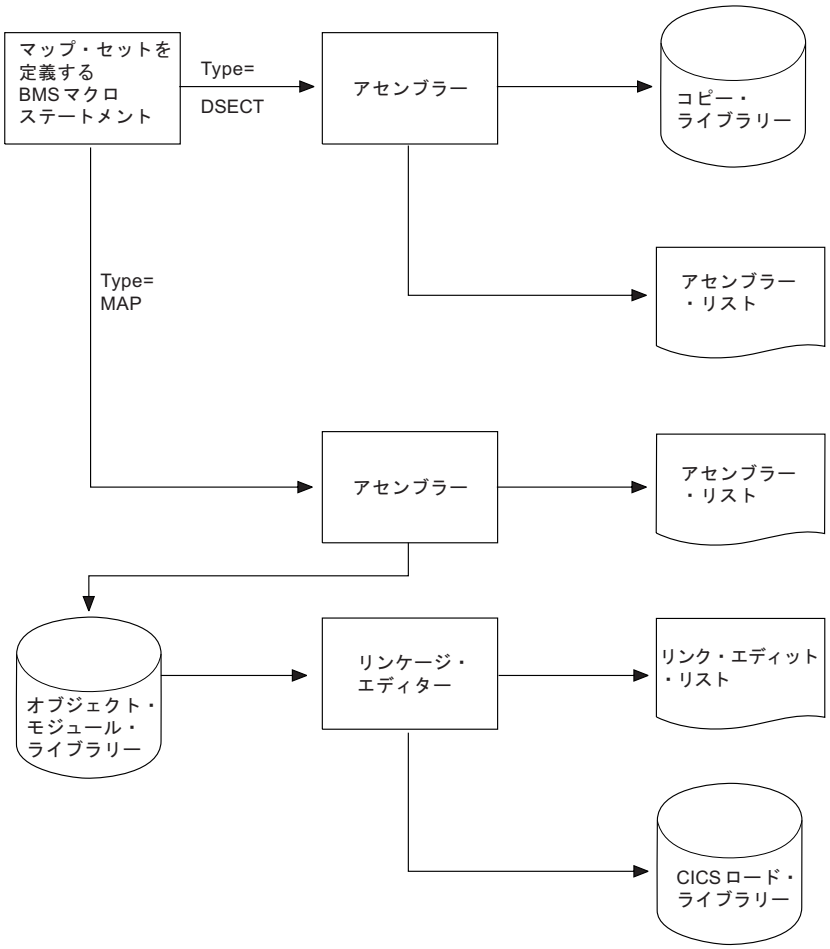


図 26. マップの準備

どの方法でマップを作成するとしても、そのマップを使用するプログラムのいずれかをコンパイル (アセンブル) する前に、当のマップを作成する必要があります。さらに、マップを変更した場合には、通常は、そのマップを使用するすべてのプログラムを再コンパイル (再アセンブル) する必要があります。変更内容によっては、プログラムが使用する物理マップにしか影響せず、対応したシンボリック・マップに反映されないものもあります。このような変更の 1 つは、フィールドの順序を変更しないフィールド位置の変更です。しかし、データ・タイプ、フィールド長、フィールド・シーケンス、およびその他の変更はシンボリック・マップに影響するので、常に再コンパイル (再アセンブル) するのが最も安全です。

CICS は、IBM 画面定義機能 II (SDF II) (プログラム番号 5665-366) のようなライセンス・プログラムを使用して対話式に行う、BMS マップ・セットと区分セットの定義をサポートします。SDF II について詳しくは、「*Screen Definition Facility II Primer for CICS/BMS Programs*」および「*Screen Definition Facility II General Information*」を参照してください。

マップ・セットまたは区分セットの常駐モードを、リンク・エディット・ステップで RMODE(ANY) と指定すると、CICS は、BMS マップ・セットおよび区分セットを 16MB 境界より上のアドレスにロードします。CICS の以前のリリースからのマップ・セットまたは区分セットのいずれかを使用している場合は、それらを

RMODE(ANY) を指定して再度リンク・エディットすると、16MB 境界より上のアドレスにロードすることができます。RMODE(ANY) を指定するリンク・エディット・ステップの例については、このセクションのサンプル・ジョブ・ストリームを参照してください

このセクションには、以下の内容が含まれています。

- 『マップ・セットのインストール』
- 168 ページの『区分セットのインストール』
- 169 ページの『CICS へのプログラム、マップ・セット、および区分セットの定義』

関連概念

775 ページの『第 60 章 基本マッピング・サポート』

関連タスク

166 ページの『DFHMAPT プロシージャーを使用した BMS マップからの HTML テンプレートのインストール』

マップ・セットのインストール

このセクションでは、まずマップ・セットのタイプ、その定義方法、そして CICS がそれを認識する方法について説明します。それに続いて、物理マップ・セットおよびシンボリック記述マップ・セットを別々に準備する方法について説明します。最後に、物理マップ・セットおよびシンボリック記述マップ・セットの両方を 1 つのジョブで準備する方法について説明します。これらの説明では、SYSPARM パラメーターを、マップ・セットの 2 つのタイプを識別するために使用することを前提とします。

以下を参照してください。

- 『マップ・セットのタイプ』
- 161 ページの『物理マップ・セットのインストール』
- 163 ページの『シンボリック記述マップ・セットのインストール』
- 165 ページの『物理マップおよびシンボリック記述マップの同時インストール』

マップ・セットのタイプ

1 つのマップ・セットをインストールするには、2 つのタイプのマップ・セットを準備しなければなりません。

- **物理**マップ・セット。BMS が、アプリケーション・プログラムが使用する標準的な装置独立の形式のデータから、端末で必要な装置依存の形式のデータに変換するために使用します。
- **シンボリック記述**マップ・セット。ユーザー・データの標準的な装置独立の形式を定義するために、アプリケーション・プログラムで使用されます。これは、アセンブラー言語における DSECT、COBOL におけるデータ定義、PL/I における BASED または AUTOMATIC 構造、および C/370 における「struct」です。

物理マップ・セットは、CICS ロード・ライブラリーのカタログに入れられていなければなりません。シンボリック記述マップ・セットは、ユーザー・コピー・ライブラリーのカタログに入れるか、あるいは直接アプリケーション・プログラム自体に挿入することができます。

マップ・セット定義マクロは、二度アセンブルされます。一度は、BMS の形式設定アクティビティーで使用される物理マップ・セットを生成するためで、もう一度は、アプリケーション・プログラムにコピーされるシンボリック記述マップ・セットを生成するためです。

必要なマップ・セットのタイプの定義

マップ・セットの 2 つのタイプは、以下のいずれかによって識別できます。

- DFHMSD マクロの TYPE オペランド
- マップ・セットをアセンブルするのに使用するジョブの EXEC ステートメントにおける SYSPARM オペランドの使用

この目的で SYSPARM オペランドを使用すると、DFHMSD マクロの TYPE オペランドは無視されます。SYSPARM を使用すると、物理マップ・セットとシンボリック記述マップ・セットの両方を、同一の変更されていない BMS マップ・セット定義マクロから生成することが可能です。

マップ・セットは、**位置合わせされない**マップ・セット、または**位置合わせされる**マップ・セットのいずれかとしてアセンブルされます (位置合わせされるマップは、その長さフィールドがハーフワード境界に調整されます)。アプリケーション・パッケージが位置合わせされるマップを必要とする場合を除いて、位置合わせされないマップを使用します。

マップ・セットが位置合わせされるかされないかを決定できるのは SYSPARM の値のみで、これは EXEC PROC=DFHMAPS ステートメントで指定します。SYSPARM オペランドは、物理マップ・セットまたはシンボリック記述マップ・セット (DSECT) のどちらをアセンブルするかを指定するのにも使用されます。この場合、SYSPARM オペランドは TYPE オペランドを指定変更します。どちらのオペランドも指定しない場合は、位置合わせされない DSECT が生成されます。

DFHMSD マクロの TYPE オペランドは、物理マップ・セットまたはシンボリック記述マップ・セットのどちらが必要かの定義のみ可能です。

各種のマップ・セットを生成するためのオペランドの組み合わせで可能なものについては、表 9を参照してください。

表 9. マップのアセンブルのための SYSPARM と DFHMSD オペランドの組み合わせ

マップ・セットのタイプ	EXEC DFHMAPS ステートメントの SYSPARM オペランド	DFHMSD マクロの TYPE オペランド
位置合わせされる記号記述マップ・セット (DSECT)	A A ADSECT	指定されない DSECT 任意 (SYSPARM が使用される)

表 9. マップのアセンブルのための *SYSPARM* と *DFHMSD* オペランドの組み合わせ *(続き)*

マップ・セットのタイプ	EXEC DFHMAPS ステートメントの SYSPARM オペランド	DFHMSD マクロの TYPE オペランド
位置合わせされる 物理マップ・セット	A AMAP	MAP 任意 (SYSPARM が 使用される)
位置合わせされない 記号 記述マップ・ セット (DSECT)	指定されない 指定されない DSECT	指定されない DSECT 任意 (SYSPARM が 使用される)
位置合わせされない 物理マップ・セット	指定されない MAP	MAP 任意 (SYSPARM が 使用される)

物理マップ・セットは、それ自体がアセンブルされて位置合わせされるマップになっているか、位置合わせされないマップになっているかを示しています。この情報は実行時に検査され、適切なマップ位置合わせが使用されます。したがって、位置合わせされるマップ・セットと位置合わせされないマップ・セットを混合しておくことができます。

拡張データ・ストリーム端末の使用

3270 情報表示システム用に設計されたアプリケーションとマップは、変更しなくても、カラー、拡張強調表示、プログラム式シンボル、妥当性検査のような 3270 データ・ストリームに対する拡張機能をサポートする装置で実行されます。カラーのような固定拡張属性を使用するには、物理マップ・セットの再アセンブルのみが必要です。アプリケーション・プログラムによる動的属性変更が必要な場合は、物理マップ・セットとシンボリック記述マップ・セットの両方を再アセンブルしなければならず、アプリケーション・プログラムも再アセンブルまたは再コンパイルしなければなりません。

物理マップ・セットのインストール

162 ページの図 27 は、物理マップ・セットをインストールするための、アセンブラーおよびリンケージ・エディターのステップを示しています。

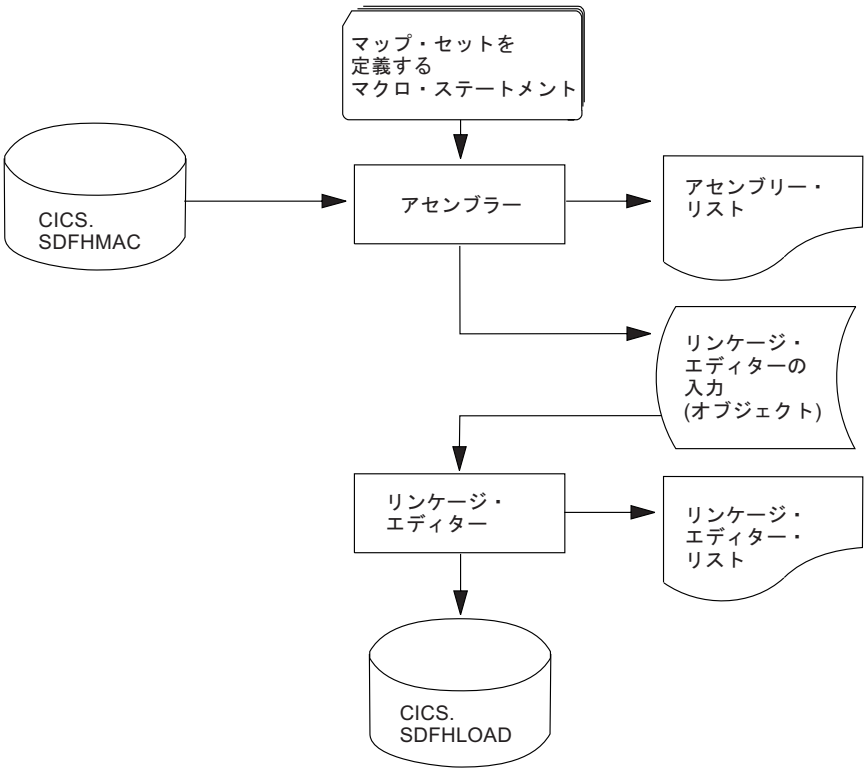


図 27. 物理マップ・セットのインストール

図 28 は、物理マップ・セットのアセンブルおよびリンク・エディットのジョブ・ストリーム例を示しています。

```
//PREP     JOB  'accounting information',CLASS=A,MSGLEVEL=1
//STEP1    EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(MAP)'          1
//SYSPUNCH DD   DSN=&&TEMP,DCB=(RECFM=FB,BLKSIZE=2960),
//         SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN    DD   *
```

マップ・セットを定義するマクロ・ステートメント

```
/*
//STEP2    EXEC PROC=DFHLNKVS,PARM='LIST,LET,XREF'               2
//SYSLIN   DD   DSN=&&TEMP,DISP=(OLD,DELETE)
//         DD   *
         MODE RMODE(ANY|24)                                    3
         NAME mapsetname(R)                                    4
/*
//
```

図 28. 物理マップ・セットのアセンブルおよびリンク・エディット

Notes

1. ハーフワードで位置合わせされる長さフィールドの場合、SYSPARM(MAP) オプションではなく SYSPARM(AMAP) オプションを指定します。
2. 物理マップ・セットは、RMODE(ANY) および RENT オプションを指定してリンク・エディットしない限り、CICS キー・ストレージにロードされます。物理マップ・セットをこれらのオプションを指定してリンク・エディットする場合は、RENTPGM 初期設定パラメーターで RENTPGM=PROTECT が指定されると

いう条件で、キー 0 保護ストレージにロードされます。ただし、マップ・セット (3270 または LU1 装置のみに送信されるものは除きます) を、RENT または REFR オプションを指定してリンク・エディットすることはお勧めしません。その理由は、場合によっては CICS がマップ・セットを変更することがあるためです。一般的には、3270 または LU1 装置のみに送信されるマップ・セットに対して、RENT または REFR オプションを使用してください。CICS で使用可能なストレージ保護機能の詳細については、「*CICS System Definition Guide*」のストレージ保護を参照してください。

3. MODE ステートメントは、マップ・セットが 16MB 境界より上のアドレス (RMODE(ANY)) または下のアドレス (RMODE(24)) のいずれにロードされるかを指定します。 RMODE(ANY) は、CICS がマップ・セットを仮想記憶域のどこにでもロードできることを指示しますが、可能であれば 16MB 境界より上のアドレスにロードしようとします。
4. BMS がストレージにロードする物理マップ・セットの名前を指定するには、NAME ステートメントを使用します。マップ・セットが装置依存の場合は、アプリケーション・プログラムで使用する 1 文字から 7 文字の元のマップ・セット名に装置接尾部を追加して、マップ・セット名を派生させます。CICS BMS がサポートする各種端末の追加される接尾部は、マップ・セットの定義に使用される DFHMSD マクロの TERM または SUFFIX オペランドで指定されるパラメーターに依存します。

物理マップ・セットを使用するには、そのリソース定義を定義してインストールしなければなりません。これは、 169 ページの『CICS へのプログラム、マップ・セット、および区分セットの定義』で説明するように、プログラム自動インストール機能を使用するか、CEDA DEFINE MAPSET コマンドと INSTALL コマンドを使用することにより実行できます。

シンボリック記述マップ・セットのインストール

シンボリック記述マップ・セットにより、アプリケーション・プログラマーは、物理マップ・セットのフィールドへの記号による参照を作成できます。図 29 には、BMS 用のシンボリック記述マップ・セットの準備が示されています。

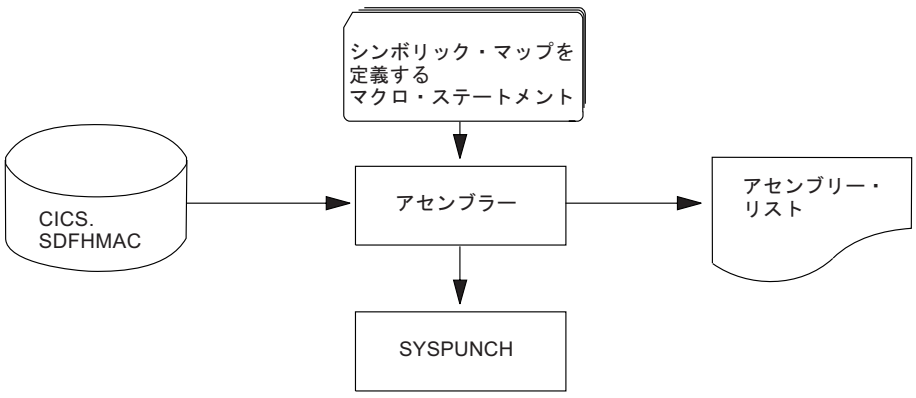


図 29. DFHASMVS プロシージャーを使用したシンボリック記述マップ・セットのインストール

シンボリック記述マップ・セットをプログラムで使用するには、そのマップ・セットのソース・ステートメントをアセンブルし、SYSPUNCH を経由して、ストレージ

定義のパンチ・コピーを手に入れなければなりません。これを最初に実行するときに、SYSPUNCH 出力を SYSOUT=A に送信すると、シンボリック記述マップ・セットのリストを入手することができます。多数のマップ・セットがご使用のシステムによって使用される場合、または共通のマップ・セットに複数ユーザーがいる場合は、使用するそれぞれの言語ごとに、ご自分専用のコピー・ライブラリーを設置してください。

記号記述が複数のプログラム言語に対して同一の名前で準備されているときは、シンボリック記述マップ・セットの別々のコピーを、それぞれのユーザー・コピー・ライブラリーに入れなければなりません。ユーザー・コピー・ライブラリーが SYSLIB に正しく連結されていることを確認してください。

異なる接尾部を持つ物理マップ・セットのバージョンすべてに対応するシンボリック記述マップ・セットは、1 つしか必要ありません。例えば、画面サイズの異なる端末で同じアプリケーションを実行するには、以下のようにします。

1. それぞれ同一のフィールドを持つが、画面サイズに合うように位置が決められた 2 つのマップ・セットを定義する。それぞれのマップ・セットは同じ名前と、異なる接尾部を持っています。この接尾部は、端末特定の接尾部に一致します。
2. 異なる物理マップ・セットを別々にアセンブルして、リンク・エディットする。ただし、作成するシンボリック記述マップ・セットは 1 つだけです。その理由は、シンボリック記述マップ・セットは、すべての物理マップ・セットの場合で同じになるためです。

図 30 のジョブ・ストリーム例を使用すれば、シンボリック記述マップ・セットのリストを手に入れることができます。これは、CICS がサポートするすべてのプログラム言語に適用されます。

```
//DSECT    JOB  'accounting information',CLASS=A,MSGLEVEL=1
//ASM      EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(DSECT)'
//SYSPUNCH DD   SYSOUT=A
//SYSIN    DD   *
```

マップ・セットを定義するマクロ・ステートメント

```
/*
//
```

図 30. シンボリック記述マップ・セットのリスト作成

長さフィールドがハーフワードで位置合わせされるシンボリック記述マップ・セットをアセンブルする場合は、図 30 のジョブ例における EXEC ステートメントを以下のように変更します。

```
//ASSEM  EXEC PROC=DFHASMVS,PARM.ASSEM='SYSPARM(ADSECT)'
```

シンボリック記述マップ・セットのパンチ・コピーを手に入れたい場合、上記の例の //SYSPUNCH ステートメントを、パンチ・データ・ストリームに出力を送信するようにコーディングします。例:

```
//SYSPUNCH DD   SYSOUT=B
```

シンボリック記述マップ・セットを専用コピー・ライブラリーに保管するには、以下と同じようなジョブ制御ステートメントを使用します。

```
//SYSPUNCH DD   DSN=USER.MAPLIB.ASM(map set name),DISP=OLD
//SYSPUNCH DD   DSN=USER.MAPLIB.COB(map set name),DISP=OLD
//SYSPUNCH DD   DSN=USER.MAPLIB.PLI(map set name),DISP=OLD
```

物理マップおよびシンボリック記述マップの同時インストール

166 ページの図 31 は、物理マップとシンボリック記述マップを一緒にインストールするための DFHMAPS プロシージャーを示しています。DFHMAPS プロシージャーは、 166 ページの図 31 で示すように、以下の 4 つのステップから構成されています。

1. マップ・セット用にコーディングした BMS マクロが、一時順次データ・セットに追加される。
2. マクロがアセンブルされ、物理マップ・セットが作成される。MAP オプションは、EXEC ステートメントの SYSPARM グローバル変数にコーディングされます (PARM='SYSPARM(MAP)')。
3. 物理マップ・セットがリンク・エディットされ、CICS ロード・ライブラリーに入れられる。
4. 最後にマクロが再びアセンブルされ、このときはシンボリック記述マップ・セットが生成される。このステップでは、DSECT を EXEC ステートメントの SYSPARM グローバル変数にコーディングします (PARM='SYSPARM (DSECT)'。出力は、//SYSPUNCH DD ステートメントで指定されたあて先に送信されます。 DFHMAPS プロシージャーでのこの宛先は CICSTS41.CICS.SDFHMAC ライブラリーです。

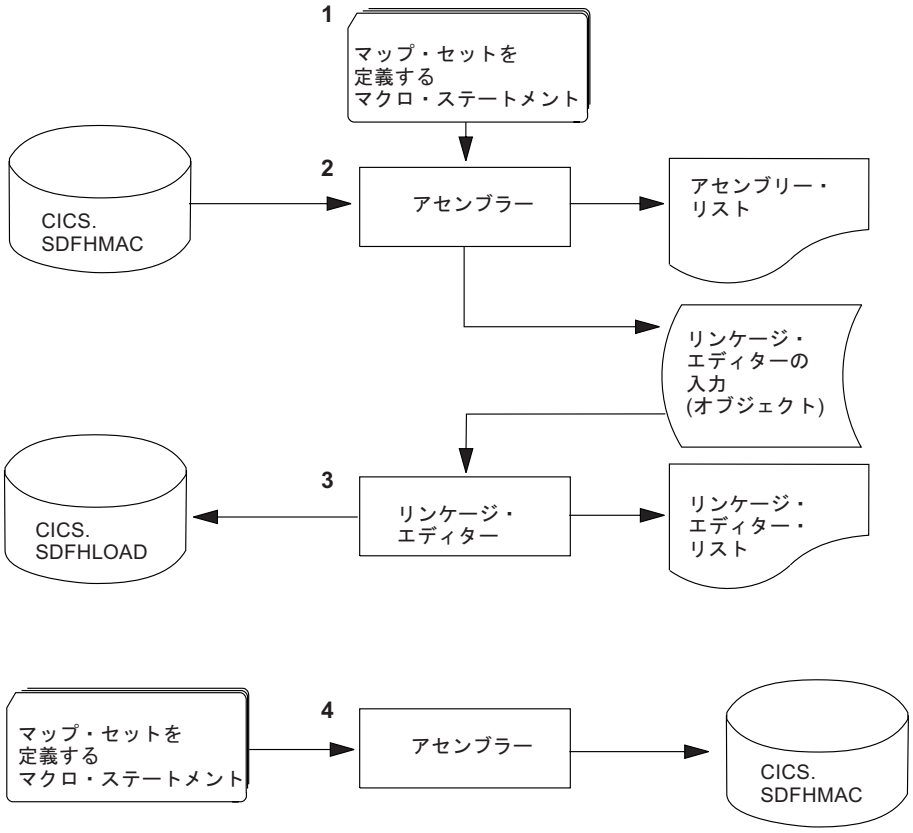


図 31. 物理マップ・セットおよびシンボリック記述マップ・セットの同時インストール

DFHMAPT プロシージャーを使用した BMS マップからの HTML テンプレートのインストール

このタスクについて

DFHMAPT プロシージャーは DFHMAPS と似ていますが、BMS マップから生成される HTML テンプレートをインストールするステップが追加されています。このステップにおいて、TEMPLATE が EXEC ステートメントの SYSPARM グローバル変数にコーディングされます (PARM='SYSPARM(TEMPLATE)')。DFHMAPT プロシージャーでは、出力は CICSTS41.CICS.SDFHHTML に送信されます。

独自のマクロを使用して HTML テンプレートをカスタマイズしたい場合で、BMS ソースにご自分のマクロを追加したくない場合は、ステップ ASMTEMPL を以下のように変更します。

1. EXEC ステートメントの **PARM** パラメーターを次のように変更する。

   ```
   PARM='SYSPARM(TEMPLATE,macro_name),DECK,NOOBJECT'
   ```

2. ご自分のマクロが入っているライブラリーを SYSLIB 連結に追加する。

物理マップおよびシンボリック記述マップをインストールするための JCL

物理マップ・セットとシンボリック記述マップ・セットのソース・ステートメントとをアセンブルしてできるロード・モジュールは、図 32 のジョブ・ストリーム例を使用して、同一のジョブで生成することができます。

```
//PREPARE  JOB  'accounting information',CLASS=A,MSGLEVEL=1
//ASSEM    EXEC PROC=DFHMAPS,MAPNAME=mapsetname,RMODE=ANY|24   (see note)
//SYSUT1   DD   *

      マップ・セットを定義するマクロ・ステートメント

/*
//
```

図 32. 物理マップおよびシンボリック記述マップの同時インストール

注: RMODE ステートメントは、マップ・セットが 16MB 境界より上のアドレスにロードされるか (RMODE=ANY)、または下のアドレスにロードされるか (RMODE=24) を指定します。 RMODE=ANY は、CICS がマップ・セットを仮想記憶域のどこにでもロードできることを指示しますが、可能であれば 16MB 境界より上のアドレスにロードしようとします。

DFHMAPS プロシージャーは、ハーフワードで位置合わせされないマップ・セットを生成します。入力マップの長さフィールドをハーフワードで位置合わせしたい場合は、EXEC ステートメントで A=A とコーディングしなければなりません。 図 32 のジョブ例では、EXEC ステートメントを次のように変更します。

```
//ASSEM  EXEC  PROC=DFHMAPS,MAPNAME=mapsetname,A=A
```

この変更により、アセンブリー・ステップの SYSPARM オペランドは、それぞれ、SYSPARM(AMAP) および SYSPARM(ADSECT) に変更されることになります。

DFHMAPS プロシージャーでは、シンボリック記述マップ・セットの出力 (SYSPUNCH) を、CICSTS41.CICS.SDFHMAC ライブラリーに送信します。 EXEC ステートメントで DSCTLIB=name を指定して (ここで「name」は選択されたユーザー・コピー・ライブラリーです)、これを指定変更します。

マップのアセンブルへの CSECT の追加

CSECT を使用して BMS マップを生成することが必要になる可能性があります。例えば、マップが必ず 16MB より上のアドレスに常駐するように AMODE および RMODE オプションを指定する必要がある場合、あるいは、DFSMS バインダーの IDENTIFY ステートメントを、変更管理の理由から使用する必要がある場合です。この場合、適切な CSECT を BMS マクロ・ステートメントの 1 番最初に組み込むだけでなく、条件付きアセンブラー・ステートメントもいくつか追加して、CSECT ステートメントがシンボリック記述マップに組み込まれないことを確実にする必要があります。以下の例は、CSECT 名および AMODE と RMODE ステートメントの両方を追加する方法を示しています。

```
//PREPARE  JOB  'accounting information',CLASS=A,MSGLEVEL=1
//ASSEM    EXEC PROC=DFHMAPS,MAPNAME=mapsetname,RMODE=ANY|24
//SYSUT1   DD   *
.
AIF   ('&SYSPARM' EQ 'DSECT').SKIPSD
AIF   ('&SYSPARM' EQ 'ADSECT').SKIPSD
ANYNAME  CSECT           Binder IDENTIFY requires CSECT name
ANYNAME  AMODE 31
ANYNAME  RMODE ANY
.SKIPSD  ANOP ,
DFH0STM  DFHMSD TYPE=DSECT,MODE=INOUT,CTRL=FREEKB,LANG=COBOL,          C
               TIOAPFX=YES,TERM=3270-2,MAPATTS=(COLOR,HILIGHT),        C
               DSATTS=(COLOR,HILIGHT)
         SPACE
DFH0STM  DFHMDI SIZE=(24,80)
.
.
.
SPACE
DFHMSD TYPE=FINAL
END
.
 /*
//
```

図 33. マップのアセンブルへの CSECT の追加

区分セットのインストール

区分セットは、物理マップ・セットと同じ方法でインストールします (162 ページの図 27 に説明されています)。記号記述区分セットの概念について説明しません。図 34 のジョブ・ストリームは、区分セットのアセンブルおよびリンク・エディットの例です。

```
//PREP     JOB  'accounting information',CLASS=A,MSGLEVEL=1
//STEP1    EXEC PROC=DFHASMVS
//SYSPUNCH DD   DSN=&&TEMP,DCB=(RECFM=FB,BLKSIZE=2960),
//         SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN    DD   *
.
      Macro statements defining the partition set
.
/*
//STEP2    EXEC PROC=DFHLNKVS,PARM='LIST,LET,XREF'              1
//SYSLIN   DD   DSN=&&TEMP,DISP=(OLD,DELETE)
//         DD   *
           MODE RMODE(ANY|24)                                   2
           NAME partitionsetname(R)                             3
/*
//
```

図 34. 区分セットのアセンブルおよびリンク・エディット

Notes

1. 区分セットは、RMODE(ANY) および RENT オプションを指定してリンク・エディットしない限り、CICS キー・ストレージにロードされます。区分セットをこれらのオプションを指定してリンク・エディットする場合は、**RENTPGM** シ

ステム初期設定パラメーターで RENTPGM=PROTECT が指定されるという条件で、キー 0 保護ストレージにロードされます。

CICS で使用可能なストレージ保護機能の詳細については、「*CICS System Definition Guide*」を参照してください。

2. MODE ステートメントは、区分セットが 16MB 境界より上のアドレス (RMODE(ANY)) または下のアドレス (RMODE(24)) のいずれにロードされるかを指定します。RMODE(ANY) は、CICS が区分セットを仮想記憶域のどこにでもロードできることを指示しますが、可能であれば 16MB 境界より上のアドレスにロードしようとします。
3. BMS がストレージにロードする区分セットの名前を指定するには、NAME ステートメントを使用します。区分セットが装置依存の場合は、アプリケーション・プログラムで使用する 1 文字から 7 文字の元の区分セット名に装置接尾部を追加して、区分セット名を派生させます。BMS が付加する各種端末の接尾部は、区分セットを定義した DFHPSD マクロ命令の SUFFIX オペランドで指定されたパラメーターによって異なります。

   区分セットの接尾部の全リストを提供しているプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

区分セットを使用するには、そのリソース定義を定義してインストールしなければなりません。これは、「*CICS Resource Definition Guide*」で説明するように、プログラム自動インストール機能を使用するか、CEDA DEFINE PARTITIONSET コマンドと INSTALL コマンドを使用することにより実行できます。

CICS へのプログラム、マップ・セット、および区分セットの定義

このタスクについて

CICS 始動 JCL で指定されるロード・ライブラリーのいずれかにインストールされたプログラムが使用可能になるためには、プログラム、およびそのプログラムが使用するすべてのマップ・セットとパーティションが CICS に対して定義されなければなりません。このために、CICS はリソース定義 MAPSET (マップ・セットの場合)、PARTITIONSET (区分セットの場合)、および PROGRAM (プログラムの場合) を使用します。このようなリソース定義の作成とインストールは、以下のいずれの方法を使用しても可能です。

- CICS は、最初にロードされたときに、プログラムの自動インストール機能を使用して、プログラム、マップ・セット、または区分セットの定義を動的に作成、インストールして、カタログに入れる。
- プログラム、マップ・セット、または区分セットの特定のリソース定義を作成して、それを CICS 領域にインストールすることができる。

  リソース定義は、以下のどちらかの方法を使用すればインストールすることができます。

  – CICS 初期設定時に、GRPLIST システム初期設定パラメーターで指定されるグループ・リストにリソース定義を組み込む。
  – CICS の実行中に、CEDA INSTALL コマンドを使用する。

CICS へのプログラムの定義について詳しくは、 「*CICS Resource Definition Guide*」のを参照してください。

第 10 章 アプリケーションのテスト

このガイダンスは、Java アプリケーションのテストには関係していません。

以下の方式を使用して CICS アプリケーション・プログラムをテストすることができます。

単一スレッド・テスト

『空』の CICS システムを除き、**単一スレッド**テストでは一度に 1 つのアプリケーション・トランザクションを取り上げて、トランザクションの振る舞いを調べます。これにより、プログラム・ロジックをテストできるのと同時に、基本的な CICS 情報 (リソース定義など) が正しいかどうか知ることもできます。別のシステムで通常のオンライン実動 CICS システムがアクティブであるときに、1 つの CICS 領域でこの単一のアプリケーションをテストすることができます。

マルチスレッド・テスト

マルチスレッド・テストは、いくつかのトランザクションを同時にアクティブにして行います。普通はすべてのトランザクションが同じ CICS 領域にあるので、新しいトランザクションが他のトランザクションと共存できるか、テストできます。

単一スレッド・テストでは完全に機能したトランザクションでも、マルチスレッド・テストでは失敗する場合があります。また、他のトランザクションを失敗させる原因になったり、CICS を終了させてしまうことさえあります。

レグレッション・テスト

レグレッション・テストは、システムを変更した時にシステムのすべてのトランザクションが変更前と変更後の両方に、同じ方法で処理を続けることを確認するために使用されます。これは、1 つの問題を解決するために適用した修正によって、さらに問題が起こらないようにするためのものです。変更内容については小さいデータ・ファイルを調べる方がはるかに簡単なので、小さいファイルの組を 1 つ作成して、テストを実行するというのはいい考えです。

いいレグレッション・テストは、プログラムごとにすべてのコードを試してみる、すなわちすべてのテスト項目および起こりうる条件を調べるものです。システムの発展につれて、トランザクションまたは起こりうる条件などは増加していくので、それに合わせてテスト・システムにもこれらを追加してください。各テストの結果は、前回のテスト結果と一致していなければなりません。矛盾がある場合には、疑ってみてください。端末出力、ファイル変更、およびログ項目を比較すれば、妥当性について調べることができます。

レグレッション・テストについては、順次端末サポート (611 ページの『第 40 章 順次端末サポートの使用』に説明があります) が有用です。時によって機能するモジュールがあり、現在修正中である場合には、その機能がまだ機能していることを確認するために、古いテストを再実行する必要があります。順次端末サポートによって、古いテスト・ケースの「ライブラリー」を保持し、必要な時点でそれらに戻すことが、簡単にできるようになります。

© Copyright IBM Corp. 1989, 2009

順次端末サポートによって、遠隔通信装置を使用せずにプログラムをテストすることができます。システム・プログラマーは、(端末管理テーブル (TCT) を使用して) 順次装置を端末として使用することを指定することができます。これらの順次装置は、カード読み取り装置、ライン・プリンター、ディスク装置、または磁気テープ装置とすることができます。以下のような順次装置の組み合わせも可能です。

- カード読み取り装置とライン・プリンター (CRLP)
- 入力用の 1 つ以上のディスクまたはテープ・データ・セット
- 出力用の 1 つ以上のディスクまたはテープ・データ・セット

プログラム・モジュールの基本テストを実行するためのトランザクション・テスト・ケースのストリームを作成することができます。テストの進行に伴い、トランザクション・ストリームを追加生成すれば、プログラムのマルチプログラミング機能の妥当性検査や、トランザクション・テスト・ケースの並行処理も可能になります。

アプリケーション・プログラムをテストして、デバッグすることができるようにするためには、以下の 2 つの主な作業を実行しなければなりません。

1. 『テストに対するアプリケーションの準備』
2. 173 ページの『テストに対するシステムの準備』

テストに対するアプリケーションの準備

アプリケーション・テーブル項目およびシステム・テーブル項目を準備するためには、以下のことを行う必要があります。

1. 各プログラムを変換、アセンブルまたはコンパイル、およびリンク・エディットします。テストを始める前に、プログラムのこれら 3 つのステップのいずれにおいてもエラー・メッセージがないことを確認しておいてください。
2. 変換ステップで DEBUG および EDF オプションを使用して、実行診断機能 (EDF) 表示で変換プログラムのステートメント番号を使用できるようにしておきます。
3. COBOL コンパイラー・オプションの CLIST および DMAP を使用して、ダンプおよび EDF 表示のストレージの場所と元の COBOL ソース・ステートメントとを関連付けて、作業用ストレージから変数を見付けられるようにしておきます。
4. RDO DEFINE PROFILE コマンドを使用して、トランザクションで使用するプロファイルを生成し、定義が INSTALL されていることを確認します。
5. アプリケーションの各トランザクションごとに RDO DEFINE TRANSACTION コマンドを使用して、定義が INSTALL されていることを確認します。
6. システムでプログラム自動インストールを使用しない場合には、アプリケーションで使用する各プログラムごとに RDO DEFINE PROGRAM コマンドを使用し、定義が INSTALL されていることを確認します。
7. システムでプログラム自動インストールを使用しない場合には、アプリケーションの各マップ・セットごとに RDO DEFINE MAPSET コマンドを使用し、各定義が INSTALL されていることを確認します。

8. 使用する各ファイルごとに RDO DEFINE FILE コマンドを使用するか、または、FCT に項目を入れます。RDO を使用する場合は、定義が INSTALL されていることを確認してください。
9. 必要とされるファイルごとに少なくとも 1 つのテスト・バージョンを作成します。
10. 各一時データ宛先をアプリケーション・プログラムで使用するように定義します。
11. アプリケーション・プログラムで使用する各ファイルのために、ジョブ制御 DD カードを始動ジョブ・ストリームに入れます。
12. テスト・データを準備します。

テストに対するシステムの準備

システムのデバッグの準備をするためには、以下のことを行う必要があります。

1. GRPLIST システム初期設定で指定するリストに、グループ DFHEDF を組み込むことによって、ユーザー・システムで EDF が使用できるようにします。
2. アプリケーション・プログラムに適したトレース・オプションを設定します。トレース・オプションの設定について詳しくは、「*CICS Problem Determination Guide*」のを参照してください。
3. すべてのトランザクション・ダンプ・コードについてトランザクション・ダンプが使用可能であること、ならびに、すべてのシステム・ダンプ・コードについてシステム・ダンプが使用可能であることを確認してください。これらにはデフォルトの設定値があります。ダンプ・オプションの設定について詳しくは、「*CICS Problem Determination Guide*」のを参照してください。
4. ダンプを印刷できるようにします。DFHDU660 ジョブ・ストリームまたはプロシージャーを準備し、CICS ダンプ・データ・セットを始動プロシージャーに定義してください。
5. ユーザー・システムで使用可能な SDUMP データ・セットについてシステム・プログラマーに確認して、それらを処理するための JCL を準備してください。
6. SIT の **ICVR** パラメーターをゼロより大きな数に設定して、CICS でループを検出できるようにします。通常は、5 ～ 10 秒 (ICVR=5000 ～ ICVR=10000) が、妥当な値といえます。
7. 統計を出します。統計の使用方法について詳しくは、「*CICS パフォーマンス・ガイド*」のを参照してください。

第 11 章 実行診断機能 (EDF)

実行診断機能 (EDF) を使用して、アプリケーション・プログラムまたはプログラム準備プロシージャーを変更せずに、アプリケーション・プログラムをオンラインでテストすることができます。CICS 実行診断機能は CICS 提供のトランザクション CEDF によってサポートされています。これは DFHEDFP プログラムを呼び出します。

注: 別の CICS 提供のトランザクション CEDX を介して CEDF を間接的に呼び出すこともできます。このトランザクションを使用すると、デバッグしたいトランザクションの名前を指定することができます。このセクションで CEDF トランザクションについて言及する場合 (例えば、下記のように、CICS による新規 CEDF タスクの開始について説明している場合) には、そのトランザクションが CEDX コマンドによって呼び出されている可能性があることに注意してください。

ユーザー・プログラムの名前は、文字「DFH」で始めてはいけません。この接頭部は、CICS システム・モジュールおよびサンプル用に使用されているためです。CICS 提供のトランザクションで EDF を使用しようとしても何の効果もありません。しかし、EDF は CICS サンプル・プログラムおよびある種のユーザー置換可能モジュールと一緒に使用することができます。(例えば、EDF を使用して、DFHPEP をデバッグすることができます。)

EDF は、いろいろな時点でアプリケーション・プログラムの CICS コマンドの実行を代行受信し、そこで行われていることを表示します。各コマンドは実行前に表示され、その大部分は実行の完了後にも表示されます。アプリケーション・プログラムによって送信された画面は保持されるので、まさにユーザーが実動システム上にいるかのように、テスト中にアプリケーション・プログラムと会話することができます。

EDF 制御のもとでトランザクションを実行すると、EDF は以下の時点でトランザクションを代行受信するので、トランザクションと対話することができます。

- **プログラム開始**時点。つまり、EXEC インターフェース・ブロック (EIB) が更新された後で、プログラムに制御権が与えられる前。
- **各 CICS コマンドの実行の開始**時点。この割り込みは、初期トレース項目を作成した後で、コマンドが実行される前に起こります。標準 CICS コマンドおよびフロントエンド・プログラミング・インターフェース (FEPI) コマンドの両方が代行受信されます。EXEC DLI コマンドおよび EXEC SQL コマンド、ならびにリソース・マネージャー・インターフェースを通じて処理されるすべての要求も、この時点で代行受信されます。
- ABEND、XCTL、および RETURN コマンド (これらのコマンドは、EDF が表示するエラー条件を引き起こすことがある) 以外の**すべてのコマンドの実行の終了**時点。EDF がトランザクションを代行受信するのは、コマンドの処理を終了した時点で、HANDLE CONDITION メカニズムを呼び出す前で、さらに応答トレース項目が作成される前です。
- **プログラム終了**時点。

© Copyright IBM Corp. 1989, 2009

- **タスク正常終了**時点。
- **ABEND** が起きて**タスク異常終了**の後。

EDF の実行例については、「*Designing and Programming CICS Applications*」を参照してください。この資料では、EDF セッションのサンプルを使用して解説しています。

注: オプション NOEDF を使用して変換されたプログラムの場合には、各コマンドの実行の前後以外は、上記の時点がなおも適用されます。リソース定義で、またはプログラム自動インストール出口によって、CEDF が NO と定義されたプログラムの場合、プログラム開始画面も終了画面も同様に抑制されます。

EDF がアプリケーション・プログラムの実行に割り込むたびに、新しい CEDF タスクが開始されます。各 CEDF タスクは短命で、適切な表示を処理するのに十分なだけの長さを存続します。

EDF 対話用に使用する端末は送受信 (ATI/TTI) 状況になっている必要があり、データの送信と受信ができなければなりません。これは、ディスプレイ端末の場合は最も一般的な状況ですが、システム・プログラマーにその状況を検査してもらって判別するか、あるいは CEMT を使用することができます。

端末で開始するトランザクションの場合、EDF は、テストしているトランザクションと同じ端末でも、異なる端末でも使用することができます。同じ端末から開始する場合は、画面を消去してトランザクション・コード CEDF を入力することによって開始し**なければなりません**。そうでない場合には、予測しない結果になることがあります。空の画面の最上部にメッセージ **THIS TERMINAL: EDF MODE ON** が表示されます。再び画面をクリアして、トランザクションを通常の方法で実行します。

EDF を使用している場合は、ユーザー・タスクを直接パージすることはできません。タスクを終了する必要がある場合は、まず CEDF タスクを強制的にパージし、EDF 画面が表示されている間に Enter キーを押します。Enter キーを押しても応答がない場合は、CEDF タスクの 2 度目の強制パージを行います。CEDF が終了し、ユーザー・トランザクションは AED3 異常終了を受け取ります。

この章では、以下のことについて説明します。

- 『EDF を使用する場合の制約事項』
- 179 ページの『EDF 表示画面での実行内容』
- 186 ページの『EDF の使用』
- 193 ページの『重ね書きによる変更』
- 195 ページの『EDF メニュー機能の使用』

EDF を使用する場合の制約事項

EDF を使用してアプリケーション・プログラムのデバッグを行うときは、数々の制限に注意する必要があります。

EDF を使用してデバッグするユーザー・アプリケーション・プログラムは、変換プログラム・オプションのデフォルト EDF を使用してアセンブル (コンパイル) しな

ければなりません。NOEDF を指定した場合には、プログラムは EDF を使用してデバッグすることができません。NOEDF を指定することによるパフォーマンス上の利点はありませんが、このオプションは、既にデバッグ済みのサブプログラム内のコマンドが EDF 表示画面に表示されないようにする場合に役立つことがあります。

EDF を使用してデバッグする対象のアプリケーション・プログラムでは、そのリソース定義に属性 CEDF(YES) を使用する必要もあります (デフォルト設定)。あるプログラムが CEDF(YES) を使用して定義され、変換プログラム・オプション EDF を使用してコンパイルされた場合、そのプログラムでは EDF 診断画面が表示されます。プログラムが CEDF(YES) を使用して定義されているが、変換プログラム・オプション NOEDF を使用してコンパイルされている場合は、プログラムの開始および終了の各画面のみが表示されます。CEDF(NO) が指定されている場合、EDF 画面は表示されません。

属性 CEDF(NO) を設定したプログラムが属性 CEDF(YES) を設定したプログラムにリンクしている場合、そのトランザクションでは EDF を使用できません。例えば、CICSPlex SM 動的トランザクション・ルーティング・プログラム EYU9XLOP が属性 CEDF(NO) を使用して定義され、ユーザー置換可能プログラム EYU9WRAM (ワークロード管理処理用) が属性 CEDF(YES) を使用して定義されている場合、EYU9WRAM のデバッグには EDF を使用できません。あるトランザクション内の複数のプログラムをデバッグするには、そのすべてのプログラムが CEDF(YES) を使用して定義されるようにする必要があります。

EDF を使用するにあたってはいくつかの制約事項があるので、いずれか一方の画面モードを使用した方がいい場合や使用することが必要になる場合があります。

- リモート・トランザクションを実行する場合、EDF は単一画面モードでしか使用できません。
- 単一画面モードでテストする場合、EDF では VM PASSTHRU はサポートされません。
- 単一画面モードでは、メッセージが EDF 表示画面を妨害するので、ユーザー・トランザクションおよび CEDF のどちらもメッセージ・ジャーナリングを指定すべきではありません。メッセージ・ジャーナリングは各トランザクションのプロファイル定義によって制御されます。
- 単一画面モードでは、CEDF トランザクションにはプロファイル定義に PROTECT=YES を指定すべきではありません。このオプションを指定すると、CEDF トランザクションのメッセージ保護は無効になります。ユーザー・トランザクションには、CEDF のもとで実行する場合でも、PROTECT=YES オプションを指定することができます。この制約事項は二重画面モードには適用されません。
- SEND LAST コマンドが発行された場合に、単一画面モードを使用していると、EDF が終わってから、コマンドが処理されます。
- 区分画面を使用するアプリケーション・プログラム、またはアプリケーション・プログラム自身が要求単位 (RU) チェーニングを行うプログラムの場合は、二重画面モードでテストしなければなりません。
- 単一画面モードで、ユーザー・トランザクションのプロファイルに INBFMH=ALL または INBFMH=DIP が指定されている場合には、CEDF のプロ

ファイルは同一の INBFMH 値をもっていなければなりません。そうでない場合には、ユーザー・トランザクションは ADIR で異常終了します。その点、二重画面モードは一致するプロファイルを必要としません。

- インバウンド応答モードを、属性設定キーを使用可能にする「文字」に設定している場合、EDF は、単一画面モードではそれらの属性設定キーを使用禁止にします。
- 二重画面モードにおいて EDF のもとで CECI を使用する場合には、ある種のコマンド (例えば、ASSIGN および ADDRESS) はトランザクション端末にではなく、EDF 端末に対して発行されるのだということを理解しておいてください。CEDF から CECI を呼び出す方法について詳しくは、INVOKE CECIを参照してください。
- TCAM 端末が EDF によりサポートされるのは、二重画面モードの場合だけで、端末がプールされていない場合です。

  注: CICS Transaction Server for z/OS バージョン 4 リリース 1 では、ローカル TCAM 端末はサポートされません。サポートされている TCAM 端末は、TCAM の DCB インターフェース (ACB ではない) によって CICS TS 3.1 以前の端末専有領域に接続されているリモート端末のみです。
- 二重画面モードで EDF を使用する場合には、例えば、START コマンドを発行して、EDF 端末で 2 番目のタスクが開始されることは避ける必要があります。EDF は疑似会話型トランザクションなので、使用している端末で 2 番目のタスクが開始されるのを妨げません。これにより、環境によってはデッドロックになることがあります。
- 二重画面モードで EDF 画面抑止を使用する場合には、DELAY、WAIT など、長い待機の原因となるコマンド、または 2 番目の RECEIVE によって、EDF はそれが終了したかのように見えることがあります。タスクが ABEND した場合には、EDF はモニター端末で再活動化されます。

以下の制約事項は、両方の画面モードに適用されます。

- トランザクションが FREE コマンドを出した場合、警告は出ないまま、EDF はオフに切り替えられます。
- EDF は、CPI 通信インターフェース (CPI-C) または SAA リソース・リカバリー・インターフェース (CPI-RR) に対する呼び出しを代行受信しません。EDF のもとで CPI 呼び出しを使用するトランザクションをテストすることはできますが、呼び出し点での EDF 表示画面を見ることはできません。
- SIGNON コマンドの処理時に、CEDF はパスワード値の表示を抑制し、パスワードが偶然見られてしまうリスクを減らします。

OPEN TCB および EDF

ユーザー・プログラムが OPEN TCB (L8、L9、X8、または X9) を使用して正常に実行している場合であっても、CEDF はそのプログラムが QR TCB で実行されるよう強制します。これは、CEDF 自身がスレッド・セーフではないためです。

パラメーター・リストのスタッキング

CEDF のみが、EXEC CICS パラメーター・リストのコピー用のスタッキング・レベルを 1 つ持っています。これは、アプリケーションが EXEC 可能グローバル・

ユーザー出口、またはユーザー置換可能モジュール (URM) を呼び出す場合に、グローバル・ユーザー出口または URM によって発行される EXEC CICS コマンドのパラメーター・リストが、メインプログラムによって発行される EXEC CICS コマンドのパラメーター・リストをオーバーレイすることがあることを意味します。

セキュリティーに関する考慮事項

EDF は、強力なツールであるため、ご使用のシステムでは、その接続時間セキュリティーによって使用が制限されている場合があります。(ご使用のシステムで使用される外部セキュリティー・マネージャーが、EDF トランザクションに対するセキュリティー属性を定義します。) このような制限が行われているときに、CEDF の使用が認可されていない場合には、トランザクションを開始することができません。

セキュリティーの使用における概要については、システム・プログラマーに確認するか、または「*CICS RACF Security Guide*」を参照してください。

EDF 表示画面での実行内容

すべての EDF 表示画面は同じ一般形式をもっていますが、内容は、タスクに割り込みが起こった時点によって異なります。表示画面には到達した代行受信点が示され、その代行受信点に関連する情報も表示されます。図 35 は代表的な表示画面の例です。これは、SEND MAP コマンドの実行後に表示されたものです。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID: 1234567 DISPLAY:00
STATUS:  COMMAND EXECUTION COMPLETE                                    1
 EXEC CICS SEND MAP
  MAP ('T1     ')
  FROM ('.....................................................'...)
  LENGTH (154)
  MAPSET ('DFH0T1 ')
  CURSOR                                                               2
  TERMINAL
  ERASE
  NOFLUSH
  NOHANDLE




 OFFSET:X'002522'    LINE:00673          EIBFN=X'1804'
 RESPONSE: NORMAL                              EIBRESP=0              3
ENTER:  CONTINUE                                                       4
PF1 : UNDEFINED          PF2 : SWITCH HEX/CHAR  PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE  PF6 : USER DISPLAY
PF7 : SCROLL BACK        PF8 : SCROLL FORWARD   PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY   PF11: EIB DISPLAY      PF12: ABEND USER TASK
```

図 35. 代表的な EDF 表示画面

注: 1ヘッダー 2本文 3メッセージ行 4機能のメニュー

表示画面は、ヘッダー、本体 (基本表示域)、メッセージ行、およびこの時点で選択できる機能のメニューから構成されています。本体が 1 画面に収まらない場合には、PF7 と PF8 を使用してスクロールできる複数の画面を、EDF が作成します。ヘッダー、メニュー、およびメッセージ領域は各画面で繰り返し表示されます。

ヘッダー

ヘッダーには、以下のものが表示されます。

- 実行中のトランザクションの ID
- 実行中のプログラム名
- CICS がトランザクションに割り当てた内部タスク番号
- トランザクションを実行中の CICS 領域のアプリケーション ID
- 表示番号
- STATUS (状況)、すなわち、EDF による代行受信の理由

本体

本体、すなわち表示画面の主要部分に含まれる情報は、代行受信した時点によって変わります。以下の画面は、本体の内容を示しています。

- 『プログラムの開始時』
- 181 ページの『CICS コマンドの実行開始時』
- 182 ページの『コマンドの実行の終了時』
- 184 ページの『プログラムおよびタスクの終了時』
- 185 ページの『異常終了時』

プログラムの開始時

プログラム開始時点では、EDF は、図 36 に示すように、COMMAREA (ある場合)、および EIB 内の主要フィールドの内容を示します。上記の EIB フィールドのプログラミング情報については、 *CICS アプリケーション・プログラミング・リファレンス*を参照してください。COMMAREA 値が指定されていない場合、画面上の行 4 はブランクのままで、EIBCALEN はゼロの値をもちます。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID: 1234567 DISPLAY:00
STATUS:  PROGRAM INITIATION

    COMMAREA     = '3476559873'
    EIBTIME      = 92920
    EIBDATE      = 91163
    EIBTRNID     = 'AC20'
    EIBTASKN     = 32
    EIBTRMID     = 'S246'

    EIBCPOSN     = 4
    EIBCALEN     = 10
    EIBAID       = X'7D'                                AT X'032F059A'
    EIBFN        = X'0000'                              AT X'032F059B'
    EIBRCODE     = X'000000000000'                      AT X'032F059D'
    EIBDS        = '........'
+   EIBREQID     = '........'


ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY         PF12: UNDEFINED
```

図 36. プログラム開始時の代表的な EDF 表示画面

CICS コマンドの実行開始時

CICS コマンドの実行開始時点では、EDF は、図 37 に示すように、キーワード、オプション、引数値を含むコマンドを表示します。PF2 を押すことによって、16 進形式と文字形式のいずれかで情報を表示する (交互に切り替える) ことができます。文字形式を要求すると、数値引数は符号付きの数字形式で表示されます。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00032 APPLID: 1234567 DISPLAY:00
STATUS:  ABOUT TO EXECUTE COMMAND
EXEC CICS SEND MAP
 MAP ('T1     ')
 FROM ('.....................................................'..)
 LENGTH (154)
 MAPSET ('DFH0T1 ')
 CURSOR
 TERMINAL
 ERASE
 NOFLUSH
 NOHANDLE



OFFSET:X'002522'   LINE:00673   EIBFN=X'1804'

ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : SWITCH HEX/CHAR     PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY         PF12: ABEND USER TASK
```

図 37. CICS コマンドの実行開始時の代表的な EDF 表示画面

図 38 には、DB2 バージョン 2.3 で実行されている EXEC SQL コマンドの**実行開始**用の類似した画面が示されています。

```
TRANSACTION: LOKO PROGRAM: TLOKO TASK: 00082 APPLID: 1234567 DISPLAY:00
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL UPDATE
 DBRM=TLOK0,    STMT=00242, SECT=00001
 IVAR 001: TYPE=CHAR,                LEN=00010        AT X'001E5A99'
           DATA=X'F0F0F0F0F0F1F0F0F0F0'









OFFSET:X'000298'   LINE: UNKNOWN EIBFN= X'0A02'
ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : UNDEFINED           PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY         PF12: ABEND USER TASK
```

図 38. SQL コマンドの実行開始時の代表的な SQL 表示画面

オプションおよび値に加えて、コマンドは、プログラム内の 16 進オフセットによって識別されます。DEBUG 変換プログラム・オプションを使用してプログラムを

変換した場合には、 181 ページの図 37 に示されているように、行番号も現れます。(このオプションの詳細については、 90 ページの『変換プログラムのオプションの定義』を参照してください。)

EXEC SQL コマンドまたは EXEC DLI コマンドの開始時に、EDF 表示画面の本体には、コマンドが変換する先の CALL のパラメーター・リストが示されます。DLI コマンドが複数の CALL ステートメントを生成する場合には、最後の CALL ステートメントしか見えません。

コマンドの実行の終了時

コマンドの実行の終了時点では、EDF は、コマンドの開始時と同じ形式の表示画面を提供します。この時点で、返されたか変更された変数の値、および応答コードによって、コマンドの実行の効果がわかります。ABEND、XCTL、および RETURN コマンドの場合、(これらのコマンドが、EDF が表示するエラー条件を引き起こすことがあるにもかかわらず)、EDF はこの表示画面を提供しません。 181 ページの図 37 における**実行直前**画面に対応する完了画面は、図 39 に示されています。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00054 APPLID: 1234567 DISPLAY:00
STATUS:  COMMAND EXECUTION COMPLETE
EXEC CICS SEND MAP
 MAP ('T1     ')
 FROM ('.....................................................'...)
 LENGTH (154)
 MAPSET ('DFH0T1 ')
 CURSOR
 TERMINAL
 ERASE
 NOFLUSH
 NOHANDLE



OFFSET:X'002522'   LINE:00673          EIBFN=X'1804'
RESPONSE: NORMAL                       EIBRESP=0

ENTER:  CONTINUE
PF1 : UNDEFINED          PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK        PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY   PF11: EIB DISPLAY         PF12: ABEND USER TASK
```

図 39. CICS コマンド完了時の代表的な EDF 表示画面

CICS コマンドの場合には、応答コードは名前 (例えば、NORMAL または NOTFND) および対応する 10 進形式の EIBRESP 値の両方によって記述されます。DL/I の場合には、応答コードは 2 文字の DL/I 状況コードで、EIBRESP 値はありません。EIBRESP のコードのリストを含む、プログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」に記載され、DL/I のコードについては「*Application Programming: EXEC DLI Commands*」に記載されています。

183 ページの図 40 および 183 ページの図 41 には、EXEC DLI コマンドの代表的な画面が示されています。

```
TRANSACTION: XDLI PROGRAM: UPDATE TASK: 00111 APPLID: 1234567 DISPLAY: 00
STATUS:  COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
 USING PCB (+00003)
 FIRST
 SEGMENT ('A     ')
 INTO ('             ')
 SEGLENGTH (+00012)
 FIRST
 VARIABLE
+SEGMENT ('B      ')




 OFFSET:X'000246'   LINE:  00000510          EIBFN:X'000C'
 RESPONSE:  'AD'

ENTER:  CONTINUE
PF1 : UNDEFINED            PF2 : SWITCH HEX/CHAR   PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS    PF5 : WORKING STORAGE   PF6 : USER DISPLAY
PF7 : SCROLL BACK          PF8 : SCROLL FORWARD    PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY     PF11: EIB DISPLAY       PF12: ABEND USER TASK
```

図 40. DLI コマンド完了時の代表的な EDF 表示画面 (画面 1)

```
TRANSACTION: XDLI PROGRAM: UPDATE TASK: 00111 APPLID: 1234567 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
+
 FIRST
 SEGMENT ('C        ')
 SEGLENGTH (+00010)
 LOCKED
 INTO ('SMITH    ')
 WHERE (ACCOUNT = '12345')
 FIELDLENGTH (+00005)




 OFFSET:X'000246'  LINE: 00000510          EIBFN:X'000C'
 RESPONSE: 'AD'

ENTER: CONTINUE
PF1 : UNDEFINED            PF2 : SWITCH HEX/CHAR    PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS    PF5 : WORKING STORAGE    PF6 : USER DISPLAY
PF7 : SCROLL BACK          PF8 : SCROLL FORWARD     PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY     PF11: EIB DISPLAY        PF12: ABEND USER TASK
```

図 41. DLI コマンド完了時の代表的な EDF 表示画面 (画面 2)

```
TRANSACTION: LOKO PROGRAM: TLOKO TASK: 00111 APPLID: 1234567 DISPLAY: 00
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER DSNCSQL
EXEC SQL UPDATE
 PLAN=TLOK0,    DBRM=TLOK0,    STMT=00242, SECT=00001
  SQL COMMUNICATION AREA:
  SQLCABC      = 136                                      AT X'001E5A18'
  SQLCODE      = 000                                      AT X'001E5A1C'
  SQLERRML     = 000                                      AT X'001E5A20'
  SQLERRMC     = ''                                       AT X'001E5A22'
  SQLERRP      = 'DSN'                                    AT X'001E5A68'
  SQLERRD(1-6) =  000,  000, 00001, -1, 00000,  000       AT X'001E5A70'
  SQLWARN(0-A) = '_ _ _ _ _ _ _ _ _ _ _'                  AT X'001E5A88'
  SQLSTATE     = 00000                                    AT X'001E5A93'

OFFSET:X'000298'   LINE: UNKNOWN EIBFN= X'0A02'
RESPONSE:

ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : UNDEFINED           PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY         PF12: ABEND USER TASK
```

図 42. SQL コマンド完了時の代表的な SQL 表示画面

プログラムおよびタスクの終了時

プログラム終了時および**正常なタスク終了時**には、本体情報はありません。関連情報はすべてヘッダーに入ります。図 43 および 185 ページの図 44 には、プログラムおよびタスクの終了における画面の要約が示されています。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00054 APPLID: 1234567 DISPLAY:00
STATUS:  PROGRAM TERMINATION









ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : SWITCH HEX/CHAR     PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY         PF12: ABEND USER TASK
```

図 43. プログラム終了時の代表的な EDF 表示画面

```
TRANSACTION: AC20                TASK: 00054 APPLID: 1234567 DISPLAY: 00
STATUS:  TASK TERMINATION









CONTINUE EDF? (ENTER YES OR NO)                                 REPLY: YES
ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY         PF12: UNDEFINED
```

図 44. タスク終了時の代表的な EDF 表示画面

異常終了時

異常終了または**タスク異常終了**が発生した場合、EDF は、図 45 および 186 ページの図 46 に示されている画面を表示します。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK:00054 APPLID: 1234567 DISPLAY: 00
STATUS:  AN ABEND HAS OCCURRED
   COMMAREA     = '1287656678'
   EIBTIME      = 135510
   EIBDATE      = 91163
   EIBTRNID     = 'AC20'
   EIBTASKN     = 76
   EIBTRMID     = 'S232'
   EIBCPOSN     = 4
   EIBCALEN     = 10
   EIBAID       = X'7D'                          AT X'032F059A'
   EIBFN        = X'1804'  SEND                  AT X'032F059B'
   EIBRCODE     = X'000000000000'                AT X'032F059D'
   EIBDS        = '........'
+  EIBREQID     = '........'


 ABEND :   ABCD

ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: EIB DISPLAY         PF12: UNDEFINED
```

図 45. 異常終了時の代表的な EDF 表示画面

```
TRANSACTION: AC20                TASK: 00054 APPLID: 1234567 DISPLAY: 00
STATUS:  ABNORMAL TASK TERMINATION
   COMMAREA     = '2934564671'
   EIBTIME      = 135510
   EIBDATE      = 91163
   EIBTRNID     = 'AC20'
   EIBTASKN     = 76
   EIBTRMID     = 'S232'
   EIBCPOSN     = 4
   EIBCALEN     = 10
   EIBAID       = X'7D'                          AT X'032F059A'
   EIBFN        = X'1804'  SEND                  AT X'032F059B'
   EIBRCODE     = X'000000000000'                AT X'032F059D'
   EIBDS        = '........'
+  EIBREQID     = '........'


ABEND :   ABCD
CONTINUE EDF? (ENTER YES OR NO)                                REPLY: YES
ENTER:  CONTINUE
PF1 : UNDEFINED          PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK        PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY   PF11: EIB DISPLAY         PF12: UNDEFINED
```

図 46. タスク異常終了時の代表的な EDF 表示画面

本体には、COMMAREA、および EIB 内のフィールドの値の他に、以下の項目が表示されます。

- 異常終了コード。
- 異常終了コードが ASRA の (すなわち、プログラム割り込みが起きた) 場合は、割り込み時点のプログラム状況ワード (PSW)、および PSW によって示される割り込みの原因。
- PSW が、割り込みの原因となった命令がアプリケーション・プログラム内にあることを示している場合は、メイン・エントリー・ポイントと相対的な、その命令のオフセット。

EDF の使用

EDF の実行は、CEDF トランザクションまたは CEDX トランザクションのいずれかを呼び出すことによって行うことができます。

端末を使用しないトランザクションをテストしている場合には、CEDX トランザクションを使用してください。これを使用すると、トランザクションの名前を指定することができます。

端末と関連するトランザクションをテストしている場合には、以下の方法で EDF を実行することができます。

- 188 ページの『単一画面モードにおける EDF の使用』
- 190 ページの『二重画面モードでの EDF の使用』
- 191 ページの『EDF とリモート・トランザクション』
- 191 ページの『EDF および端末を使用しないトランザクション』
- 191 ページの『EDF および DTP プログラム』

一般に、どちらの方式でも好きな方を使用することができますが、一方または他方を必要とする状態がいくらかはあります。リモート・トランザクションには単一画

面モードを使用しなければなりません。端末の選択に影響を与えるその他の条件については、 176 ページの『EDF を使用する場合の制約事項』を参照してください。

プログラム実行の中断

代行受信点のそれぞれで何ができるかは EDF の能力にかかっています。例えば、以下のことを行うことができます。

- コマンドを実行する前に引数値を変更する。CICS コマンドの場合には、実際のコマンドを変更したり、オプションの追加または削除を行うことはできませんが、任意のオプションと関連する値を変更することができます。また、NOOP を使用してコマンド実行を完全に抑制することもできます。詳しくは、 193 ページの『重ね書きによる変更』を参照してください。
- 実行によって返される引数値を変更するか、あるいは応答コードを修正するかのいずれかによって、コマンドの結果を変更する。これにより、通常のテスト・データを使用している場合には達しにくいプログラムの分岐 (例えば、入出力エラーで何が起こるかなど) をテストすることができます。また、これが問題を除去するかどうかを検査するために、エラーの結果を迂回することができます。
- プログラムの作業用ストレージ、EIB、および DIB (DL/I プログラムの場合) を表示する。
- コマンド・インタープリター (CECI) を呼び出す。CECI のもとでは、プログラムに存在しないコマンドを実行して、補足情報を得たり、実行環境を変更することができます。
- CICS 領域の他の場所を表示する。
- プログラムの作業用ストレージと EIB および DIB のほとんどのフィールドを変更する。EDF は、ストレージの他の区域を変更できないようにして、ユーザー・タスクによる他のタスクへの妨害を停止します。
- 一時記憶キューおよび一時データ・キューの内容を表示する。
- 1 つ以上の特定の条件の組が完全に満たされるまで、EDF 表示画面を抑制する。これにより、テストが高速化します。
- 前の EDF 表示画面または保管済み画面を 10 個まで検索する。
- EDF モードをオフに切り替えて、アプリケーション・プログラムを普通に実行する。
- タスクを異常終了させる。

最初の 2 つのタイプの変更は、コマンドの表示画面の本文に値を重ね書きすることによって実行できます。この方法については、 193 ページの『重ね書きによる変更』を参照してください。その他の場合には、メニューでファンクション・キーを使用します。 195 ページの『EDF メニュー機能の使用』 には、実際に実行できること、およびその方法について説明されています。

```
TRANSACTION: DLID PROGRAM: DLID  TASK: 00049 APPLID: IYAHZCIB  DISPLAY:00
 ADDRESS: 00000000









WORKING STORAGE IS NOT AVAILABLE
ENTER:  CURRENT DISPLAY
PF1 : UNDEFINED         PF2 : BROWSE TEMP STORAGE  PF3 : UNDEFINED
PF4 : EIB DISPLAY       PF5 : INVOKE CECI          PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF  PF8 : SCROLL FORWARD HALF  PF9 : UNDEFINED
PF10: SCROLL BACK FULL  PF11: SCROLL FORWARD FULL  PF12: REMEMBER DISPLAY
```

図 47. CECI を開始できる代表的な EDF 表示画面

単一画面モードにおける EDF の使用

EDF を 1 端末でしか使用しない場合には、EDF の入出力はトランザクションからのものとインターリーブします。これは、複雑そうに思えますが、実際には至って簡単な作業です。はっきりとわかる特長はただ 1 つ、SEND コマンドに RECEIVE コマンドが続き、SEND コマンドによって送信される表示画面が二度現れます。一度は、SEND を実行するときで、もう一度は、RECEIVE コマンドを実行するときです。最初の表示画面に応答することは必ずしも必要ではありませんが、応答した場合には、EDF は最初の表示画面から入力された内容を保持して、2 番目の表示画面に表示します。

EDF を開始するには、以下のいずれかを行います。

- 消去した画面からトランザクション・コード CEDF を入力する
- 適切な PF キーを押す (EDF 用に PF キーが定義されている場合)

次に、テストするトランザクションを以下のようにして開始します。

1. CLEAR キーを押して画面を消去する
2. テストしたいトランザクションのトランザクション・コードを入力する

EDF とユーザー・トランザクションの両方で同じ端末を共用している場合、EDF は以下の時点でユーザー・トランザクション表示画面を復元します。

- トランザクションがオペレーターからの入力を必要とする場合
- トランザクション表示画面が変更された場合
- トランザクションの終了時
- EDF 表示画面を抑制する場合
- USER DISPLAY が要求された場合

ユーザー表示画面は、復元可能にしておくために、以下の時点で保管されることになります。

1. タスクの開始時に、タスクの最初の EDF 画面が表示される前
2. ユーザー表示画面が変更された場合には、次の EDF 画面が表示される前
3. SCREEN SUPPRESS モードを抜けるとき

CEDF が、オプション NOEDF で変換されたアプリケーション・プログラムで使用されている場合、もしくはそのリソース定義で CEDF について NO が指定されている場合、

- EDF が、いつ表示画面がそのアプリケーション・プログラムによって変更されたかを確認することは不可能です。
- よって、EDF は、その表示画面のコピーを後の使用のために、いつ保存するかを認識しません。
- つまり、1 または 3 の状態のいずれかが適用されない限り、次に表示される EDF 表示画面が、このアプリケーション・プログラムによって送信された表示画面を上書きしてしまうことになります。
- EDF はアプリケーションの現在の表示画面を保管しておらず、その画面が変更されたことを認識していません。
- 変更された (ただし、今回上書きされた) 表示画面は復元できません。

同様に、これらの状況では、

- CEDF がアプリケーションによって変更されようとしている場合、もしくはトランザクションがオペレーターからの入力を必要とする場合、CEDF は現在の表示画面を復元できません。
- つまり、アプリケーション・プログラムから基本機能に出力コマンドを発行すると、その結果、以前の EDF 表示画面からのランダム・バックグラウンド情報が画面に表示される可能性があります。
- 入力コマンドは、アプリケーション・プログラムからの表示画面ではなく以前の EDF 画面に対して実行できます。もしくは、それがトランザクションでの最初の入力コマンド受信だと、初期 TIOA の内容から情報を得る代わりに CEDF パネルからの明示的な入力を必要とする場合があります。

これらの考慮事項は、アプリケーション・プログラムが実行したすべての画面入出力操作に当てはまります。

EDF がトランザクション表示画面を復元する場合には、アラームを鳴らしたり、あるいはユーザー・トランザクションと同じ方法でキーボードに影響を及ぼしたりすることはありません。ユーザー・トランザクション・オプションの効果が見られるのは、SEND コマンドが処理されるときで、画面が復元されるときではありません。単一画面モードで NOEDF を指定した場合には、ユーザーには見えないので、ユーザー・プログラムはデータの送受信を行わないことに注意する必要があります。

EDF がトランザクション表示画面をカラー、プログラム式シンボル、または拡張強調機能を使用する装置上に復元する場合には、これらの属性はもはや存在せず、表示画面はプログラム式シンボルまたは拡張強調を使用しないモノクロームになります。また、アプリケーション・プログラムのインバウンド応答モードが、属性設定

キーを使用できるように「文字」に設定されている場合には、EDF はこのモードをリセットし、これらのキーが使用できないようにします。これらの変更が、トランザクションが正しく実行されるのを妨げている場合には、二重画面モードでテストする必要があります。

EDF セッションをトランザクションの途中で終了させた場合は、最新の RECEIVE コマンドの後に SEND コマンドが続いていなければ、EDF はキーボードをロックして画面を復元します。SEND コマンドが続いている場合に、キーボードはアンロックされます。

疑似会話型プログラムのチェック

EDF は、単一の端末から疑似会話型トランザクションをテストするために特殊な提供を行います。端末が、疑似会話型トランザクションを構成するいくつかのタスクの間で EDF モードから抜けた場合には、最初のタスクの後でデバッグすることは非常に難しくなります。そのために、タスクが終了するときに、EDF は、EDF モードを次のタスクまで継続するかどうかをオペレーターに尋ねます。疑似会話型タスクをデバッグ中である場合には、デフォルトが「yes」となっているので ENTER を押します。終了したら、「no」と応答します。

二重画面モードでの EDF の使用

二重画面モードでは、EDF 対話に 1 台の端末を使用し、もう 1 台の端末を、テスト中のトランザクションへ入力を送ったり、トランザクションから出力を受け取るために使用します。

EDF 端末からトランザクション `CEDF tttt` を入力することによって開始します。ここで、`tttt` はトランザクションをテストする端末の名前です。

CEDF がこの応答として出すメッセージは、2 番目の端末で既に実行中のトランザクションがあるかどうかによって異なります。2 番目の端末が使用中でない場合に、最初の端末に表示されるメッセージは以下のとおりです。

```
TERMINAL tttt: EDF MODE ON
```

PROGRAM INITIATION 表示画面が現れている時に、2 番目の端末でトランザクションが開始されるまで、これ以上何も起こりません。

また、二重画面で EDF を使用して、2 番目の端末で既に実行中のトランザクションをモニターすることができます。例えば、特定の端末でトランザクションがループしていると思われる場合には、別の端末に移り、このトランザクションを実行中の端末を指定して CEDF トランザクションを入力します。最初の端末に表示されるメッセージは以下のとおりです。

```
TERMINAL tttt: TRANSACTION RUNNING: EDF MODE ON
```

EDF は、次に実行される EXEC CICS コマンドで制御を受け取り、その後で、少なくとも 1 つの EXEC CICS コマンドが実行されていることを前提として、ループの原因になっている一連のコマンドを監視します。

EDF とリモート・トランザクション

二重モードで EDF を使用することができないのは、テスト中のトランザクション、あるいはそれを呼び出す端末が別の CICS 領域によって所有されている場合です。

EDF のもとで CRTE ルーティング・セッションを使用している最中にリモート・トランザクションが異常終了した場合には、EDF はタスクの異常終了画面を表示して、ユーザー・トランザクションにメッセージ DFHAC2206 を出します。CRTE セッションは、ユーザー・タスク異常終了によって影響を受けることはありません。また、異常終了後に EDF を続行しようとする場合には、CRTE ルーティング・セッション内では端末は EDF モードのままです。

その上、実行にも相違点があります。リモート・トランザクションの場合には、EDF が続行するかどうかにかかわりなく、EDF は各トランザクションの終了時にセッションのメモリーを除去します。つまり、設定済みのすべてのオプションおよびすべての保管済み画面は、疑似会話型シーケンスの中の個別タスク間で失われるということです。

EDF および端末を使用しないトランザクション

EDF を使用して、端末を使用せずに実行するトランザクションをテストすることができます。例えば、EXEC CICS START コマンドによって開始したトランザクション、または一時データ・トリガー・レベルによって開始したトランザクションです。端末を使用しないトランザクションをテストするには、CEDX *trnx* コマンドを使用してください。この *trnx* はトランザクション ID です。

CEDX を使用してトランザクションをテストするには、

- CEDX コマンドを入力する EDF 表示に使用する端末は、指定のトランザクションが実行される CICS 領域にログオンしなければなりません。
- CEDX コマンドは、指定のトランザクションが CICS によって開始される前に発行されなければなりません。CEDX コマンドを発行する時に既に実行している同じトランザクションで別のインスタンスは無視されます。

CEDX を使用してトランザクションをデバッグするときには、CICS は、CEDX コマンドで指定されたトランザクションの定義を修正することによって EDF 操作を制御し、特別なトランザクション・クラス DFHEDFTC を参照します。EDF をオフに切り換える場合 (CEDX *tranid*、OFF を使用する場合)、CICS はトランザクション定義を通常のトランザクション・クラスに返します。

EDF および DTP プログラム

リンク上のセッションをモニターするように EDF に指示して、リモート・リンクを介して分散トランザクション処理を使用しているトランザクションをテストすることもできます。これは、CICS のもとで実行中で、EDF がインストールされている参加システムのいずれか一方 (または両方) で行うことができます。(リモート・トランザクションの場合、単一画面モードを使用しなければならないため、トランザクションが別の CICS 領域からルーティングされた場合、これを行うことはできません。)

APPC および MRO リンクの場合には、リモート・システムのシステム ID (sysid) を次のように指定できます。

```
CEDF sysid
```

これにより、EDF は自分自身を、指定されたシステムに属している任意のセッションにまたがって生成される任意のトランザクションと関連付けます。

APPC、MRO、および LU6.1 リンクの場合には、トランザクションが使用中のセッション ID (sessionid) を次のように使用することができます。

```
CEDF sessionid
```

セッション ID は CEMT INQUIRE TERMINAL トランザクションを用いて判別できますが、このことは、EDF を開始する前に、トランザクションは実行中でなければならず、セッションが確立されていなければならないことを意味します。

分散トランザクション処理を使用するトランザクションが、それに関連付けられた端末も持つ場合、または端末からトランザクションを呼び出せる場合 (たとえ、トランザクションが端末を使用しなくても) は、EDF を使用して端末から通常の方法でトランザクションをテストすることができます。

リモート・システムでのトランザクションのテストを終了した場合には、CESF によって CICS からログオフする前に、その SYSID または sessionid で EDF をオフにする必要があります。例:

```
CEDF sysid,OFF
```

これに失敗すると、そのシステムへのリンクを使用する別のトランザクションが中断されることがあります。

EDF と分散プログラム・リンクのコマンド

EDF で単一または二重端末モードを使用して、分散プログラム・リンク (DPL) コマンドを含むトランザクションをテストすることができます。しかし、EDF は DPL コマンド呼び出しおよび応答画面を表示するだけです。リモート・プログラムによって出された CICS コマンドは表示されず、リモート・プログラムは異常終了し、メッセージ「**a remote abend has occurred**」が、異常終了が起きたシステムの SYSID と一緒に EDF 端末に返されます。制御をローカル・プログラムに返した後で、EDF は通常通りのテストを続行しますが、異常終了がリモート・プログラム側の場合には、PSW を表示しません。

EDF の停止

端末の EDF 制御を終了したい場合には、どちら側でテストしているかによって方法が異なります。テスト中のトランザクションがなおも実行中で、それを続行したいが、EDF を使用したくない場合には、END EDF SESSION ファンクション・キーを押します。タスク終了代行受信に達した場合には、EDF が、続行したいかどうかをたずねてきます。続行したくない場合には、応答を NO (YES がデフォルト) と重ね書きします。端末で実行中のトランザクションがなければ、画面をクリアして、次のように入力します。

```
CEDF ,OFF
```

(スペースおよびコンマは必須です。)

二重画面モードからログオフする場合には、画面をクリアして、CEDF tttt,OFF と入力します。

どの場合にも、空画面の最上部にメッセージ「**THIS TERMINAL: EDF MODE OFF**」が表示されます。

重ね書きによる変更

EDF によって行った変更のほとんどはメモリー内の情報の変更をともないます。これは、画面に表示されている情報に、代わりに使用したい情報を重ね書きするだけで行われます。最下部のメニュー域を除き、タブ・キーを使用してカーソルが停止する場所ならどこでも変更することができます。

画面を変更する場合には、以下の規則に従わなければなりません。

- CICS コマンド画面で、任意の引数値に重ね書きすることができるが、引数のキーワードには重ね書きできない。オプションの引数を除去することはできませんし、オプションを追加することも削除することもできません。
- コマンド表示画面 (作業用ストレージ表示画面ではなく) で引数を変更する場合は、表示画面上に示されている部分しか変更することができない。表示されている値の範囲外に重ね書きしようとした場合には、変更は行われず、診断メッセージは生成されません。引数が長過ぎてその一部しか画面に表示できない場合には、引数が指す作業用ストレージ内の区域を変更する必要があります。(アドレスを判別するためには、引数を 16 進形式で表示します。引数の位置のアドレスも表示されます。)
- 文字形式では、数値に必ず符号フィールドが付いているが、このフィールドに重ね書きできるのは、マイナスまたはブランクだけである。
- 引数を文字形式で表示する場合には、一部の文字が表示可能でない場合がある (小文字を含む)。EDF は、各表示不能文字をピリオドで置き換えます。ピリオドに重ね書きする場合には、ストレージに実際に入っているのはピリオド以外の文字である可能性があるので注意しなければなりません。文字をピリオドで重ね書きすべきではありません。そうした場合には、変更は無視され、診断メッセージは発行されません。ある文字をピリオドで重ね書きする必要がある場合には、PF2 キーを使用して表示を 16 進形式に切り替えてから、X'4B' と重ね書きすることができます。
- ストレージを文字と 16 進数の両方の形式で表示し、その両方に変更を行ったときに、変更内容に万一矛盾があった場合には、16 進フィールドの値が優先する。診断メッセージは発行されません。
- HANDLE CONDITION などの一部のコマンドの引数は、数値データまたは文字データではなく、プログラム・ラベルである。EDF がこれらの引数を表示する (そして修正を受け入れる) 形式は、以下のように使用するプログラム言語によって異なります。
  - COBOL の場合は、ヌル引数 (例えば、ERROR () など) が表示されるので、それを変更することはできません。
  - C および C++ の場合は、ラベルは無効です。
  - PL/I の場合は、ラベル定数のアドレスが使用されます。(例:ERROR (X'001D0016'))

– アセンブラー言語の場合は、プログラム・ラベルのアドレスが使用されます。(例: ERROR (X'00030C'))

HANDLE CONDITION コマンドにラベル値を指定しないと、EDF は括弧を付けずに条件名だけを表示します。

- 応答フィールドは、任意の例外条件の名前 (現行機能で起こることがある ERROR を含む)、あるいは語 NORMAL で重ね書きすることができる。EDF が続行する場合の効果は、プログラムが、指定の応答で指示されたどんな処置でも行う点です。EIB 画面の EIBRESP フィールドを対応する値に変更することによって、同じ効果を得ることができます。**コマンド実行終了**画面で EIBRESP 値または応答フィールドを変更した場合には、EIBRCODE が更新されます。EIBRESP は 2 番目の EIB 画面に現れ、変更できるのはこれだけです (EIBRCODE は保護されています)。EIB 画面の EIBRESP 値を変更しても同じ効果を得ることができます。EDF はこれを行うかどうかにしたがって、EIB およびコマンド画面の関連値を変更します。
- 使用している端末に大文字変換が指定されていない場合は、必ず大文字を入力するように注意する。
- 処理する前に、任意のコマンドを NOOP または NOP によって重ね書きできる。これはコマンドの処理を抑制します。ERASE EOF キーの使用、またはブランクによる重ね書きは、効果は同じです。画面が NOOP によって再表示された場合には、ERASE EOF キーによって verb 行全体を消去して、ENTER キーを押すことによって元の verb 行を復元することができます。

プログラム内のデータ域を表すフィールドを重ね書きした場合には、アプリケーション・プログラムのストレージの中が直接変更され、永続的になります。しかし、定数 (プログラム・リテラル) を表すフィールドを変更した場合には、同一の定数を使用するプログラムの他の部分またはそのプログラムを使用する他のタスクが影響を受けることがあるので、プログラム・ストレージは変更されません。コマンドは変更済みデータを使用して実行されますが、処理後にコマンドを表示したときには、元の引数値が再表示されます。例えば、次のようにコーディングしたコマンドを含むプログラムをテストしているものとします。

```
EXEC CICS SEND MAP('MENU') END-EXEC.
```

コマンドの実行前に、EDF のもとで名前 MENU を MENU2 に変更した場合には、使用されるマップは MENU2 ですが、応答に表示されるマップは MENU です。(「previous display」キーを使用して、使用するマップ名を検査することができます。) 同一コマンドを複数回処理する場合には、毎回、この変更を入力しなければなりません。

EDF の応答

任意のキーボード入力に対する EDF の応答は、以下に挙げる規則に示されている順に従います。

1. CLEAR キーを使用すると、EDF は、変更が行われていてもそれを無視して、画面を再表示します。
2. 変更した中で無効なものがある場合、EDF は、正しく変更されているものがあればそれを受け入れて、診断メッセージの入った画面を再表示します。

3. 画面番号を変更した場合には、EDF は他のすべての変更を受け入れ、要求された画面を表示します。
4. PF キーを使用する場合には、EDF はすべての変更を受け入れ、PF キーによって要求された処置を実行します。画面の最下部のメニューの PF キー定義の下にカーソルを位置付けて ENTER キーを押すと PF キーを押すのと同じです。
5. ENTER キーを押し、画面 (REPLY フィールド以外) を修正した場合には、EDF が変更を組み込んだ画面を再表示します。
6. ENTER キーを押し、画面 (REPLY フィールド以外) を修正しなかった場合には、効果は ENTER キーの意味によって異なります。ENTER キーが CONTINUE を意味する場合には、ユーザー・トランザクションは実行を続行します。ENTER キーが CURRENT DISPLAY を意味する場合には、EDF は現行状況表示画面を再表示します。

EDF メニュー機能の使用

指定された時点で使用できるファンクション・キーは、各 EDF 表示画面の最下部のメニューに表示されています (179 ページの図 35を参照してください)。その表示画面の ENTER キーの機能も記されます。すべての表示画面に適用される機能は常に同一キーに割り当てられますが、一部のキーの定義は表示画面および代行受信点によって異なります。オプションを選択するためには、指示されたファンクション・キーを押します。端末に 24 個のファンクション・キーがある場合には、EDF は PF13 ～ PF24 をそれぞれ PF1 ～ PF12 の重複として取り扱います。端末に PF キーがない場合には、カーソルを実行したいオプションの下に位置付けて、ENTER キーを押します。

ABEND USER TASK (ユーザー・タスクの異常終了)
モニター中のタスクを終了します。EDF は、メッセージ「ENTER ABEND CODE AND REQUEST ABEND AGAIN」を表示して、この処置の確認を要求します。カーソルによって指示された位置にコードを入力した後で、ユーザーはこの機能を再び要求して、指定のコードによって識別されるトランザクション・ダンプによってタスクを異常終了する必要があります。「NO」を入力した場合、タスクは、ダンプせずに、4 つの疑問符の 4 文字のデフォルト異常終了コード (*????*) で異常終了します。

文字 A で始まる異常終了コードは CICS が使用するために予約されています。CICS 異常終了コードを使用すると、予測しない結果になることがあります。

異常終了が既に進行中のとき、またはタスクの終了処理が始まっているときに、この機能を使用することはできません。

BROWSE TEMP STORAGE (一時記憶域のブラウズ)
一時記憶キュー CEBRxxxx の表示画面を作成します。ここで、xxxx は EDF を実行中の端末の端末 ID です。この機能は、作業用ストレージ (PF5) 画面からのみ利用可能です。その後に、 207 ページの『CEBR コマンドの使用』 で説明されている CEBR コマンドを使用し、一時記憶キューの表示または修正、および一時データ・キューの読み取りまたは書き込みを実行できます。

CONTINUE (続行)
何らかの変更を行い、その変更内容を組み込む場合には、現行の画面を再表示し

ます。変更を行わなかった場合には、続行 (CONTINUE) によって、テスト中のトランザクションは次の代行受信点まで実行を再開します。続行するために、ENTER を押します。

CURRENT DISPLAY (現行表示)
何らかの変更を行い、その変更内容を組み込む場合には、現行の画面を再表示します。変更を行わなかった場合には、EDF は、最新の代行受信点に関するコマンド画面を表示します。この機能を実行するためには、適切な画面で ENTER を押します。

DIB DISPLAY (DIB 表示画面)
DL/I インターフェース・ブロック (DIB) の内容を表示します。この機能は、作業用ストレージ (PF5) 画面からのみ利用可能です。DIB フィールドについて詳しくは、「*Application Programming: EXEC DLI Commands*」を参照してください。

EIB DISPLAY (EIB 表示画面)
EIB の内容を表示します。 EIB 表示画面の例については、 180 ページの図 36を参照してください。EIB のプログラミング情報については、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。COMMAREA が存在している場合にも、EDF はそのアドレスとダンプ様式のデータを 1 行だけ表示します。

INVOKE CECI (CECI の呼び出し)
CECI へのアクセス。この機能は、作業用ストレージ (PF5) 画面からのみ利用可能です。CECI が呼び出される画面の例については、 188 ページの図 47を参照してください。そこで、 211 ページの『第 13 章 コマンド・レベル・インタープリター (CECI)』 に説明されている CECI コマンドが使用できます。これらの CECI コマンドには、コマンド実行の前後で元のコマンドによって参照されるリソースに対する INQUIRE コマンドおよび SET コマンドが含まれます。二重画面モードで CECI を実行する場合の制約事項については、インバウンド応答モードを参照してください。このパネルから CECI を使用するのは、CEDF 内で CEBR を使用するのと同様です。

END EDF SESSION (EDF セッションの終了)
トランザクションの EDF 制御を終了します。トランザクションは、その点から実行を続行しますが、もはや EDF モードでは実行しません。

NEXT DISPLAY (次画面表示)
前画面表示 (PREVIOUS DISPLAY) の逆です。直前の表示画面に戻った場合に、このオプションによって、順方向に 1 つ後のものが表示され、表示番号は 1 だけ大きくなります。

PREVIOUS DISPLAY (前画面表示)
直前の表示を画面に送信します。他の表示画面を保管していなければ、これは直前のコマンド表示画面です。現行の代行受信点の表示番号は常に 00 です。直前の表示画面を要求すると、表示番号は 1 だけ小さくなり、最初の直前の表示画面は -01、もう 1 つ前は -02、以下同様にして、最も古い表示画面が -10 になります。それ以前の画面が使用可能でない場合には、PREVIOUS オプションがメニューから消え、対応するファンクション・キーは操作不能となります。

REGISTERS AT ABEND (異常終了時のレジスター)
ローカル ASRA 異常終了が起きたときの、レジスターの値が入っているストレージを表示します。ストレージのレイアウトは以下のとおりです。

- レジスター値 (0-15)
- 異常終了時の PSW (8 バイト)

場合によっては、EDF がレジスターの値を入手する前に、領域内で 2 番目のプログラム・チェックが起こった場合に、この機能は異常終了表示のメニューに現れないことがあります。これが起こった場合には、一般には、より多くの情報を得るためにもう一度テストを実行します。

REMEMBER DISPLAY (表示記憶)
通常はメモリーに保持されない EIB 表示画面などの表示画面を EDF メモリーに入れます (EDF は各コマンドの開始時および完了時に表示画面を自動的に保管します)。メモリーには最大 10 個までの表示画面を保持することができます。表示画面は発生順の逆に番号が付けられます (すなわち、-10 が最も古い表示画面で、-01 が最も新しい表示画面です)。表示画面と関連したすべてのページがメモリーに保持され、再呼び出しした時にスクロールすることができます。しかし、作業用ストレージ表示画面を保管した場合には、ビューの画面しか保管されないことに注意してください。

SCROLL BACK (逆スクロール)
1 つの画面にすべてが収まらない EIB、DIB、またはコマンド表示画面に適用されます。ビューの画面が表示画面の最初のものではなく、最初のオプションまたはフィールドの前に正符号 (+) がある場合には、SCROLL BACK を選択して、前の画面を表示することができます。例については、 180 ページの図 36を参照してください。

SCROLL FORWARD (順スクロール)
1 つの画面にすべてが収まらない EIB、DIB、またはコマンド表示画面に適用されます。これは、表示画面内の最後のオプションまたはフィールドの後に正符号 (+) が現れて、さらに画面があることを示します。SCROLL FORWARD を使用すると、次の画面が表示されます。

SCROLL BACK FULL (一画面逆スクロール)
EIB および DIB 表示画面の SCROLL BACK オプションと同じ機能を、作業用ストレージの表示画面に対して持っています。SCROLL BACK FULL は、作業用ストレージ表示画面を 1 画面分だけ逆方向に移動し、現行画面のアドレスより下位のストレージのアドレスを表示します。

SCROLL FORWARD FULL (一画面順スクロール)
EIB および DIB 表示画面の SCROLL FORWARD オプションと同じ機能を作業用ストレージの表示画面に対して持っています。SCROLL FORWARD FULL は、作業用ストレージ表示画面を 1 画面分だけ順方向に移動し、現行画面のアドレスより上位のストレージのアドレスを表示します。

SCROLL BACK HALF (半画面逆スクロール)
作業用ストレージ表示画面を 1/2 画面分だけ逆方向に移動させる点を除き、SCROLL BACK FULL と同様です。

SCROLL FORWARD HALF (半画面順スクロール)
作業用ストレージ表示画面を 1/2 画面分だけ順方向に移動させる点を除き、SCROLL FORWARD FULL と同様です。

STOP CONDITIONS (停止条件)

図 48 に示されているメニュー画面を作成します。この画面を使用して、SUPPRESS DISPLAYS キーを押した後にその表示画面を再開する時点を EDF に指示します。 STOP CONDITIONS と SUPPRESS DISPLAYS を一緒に使用して、部分的に動くことがわかっているプログラムを検査している場合に、EDF との間の対話を省略します。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 0086 APPLID: 1234567 DISPLAY: 00
DISPLAY ON CONDITION:-

     COMMAND:                       EXEC CICS
     OFFSET:                       X'......'
     LINE NUMBER:
     CICS EXCEPTION CONDITION:       ERROR
     ANY CICS CONDITION              NO
     TRANSACTION ABEND               YES
     NORMAL TASK TERMINATION         YES
     ABNORMAL TASK TERMINATION       YES

     DLI ERROR STATUS:
     ANY DLI ERROR STATUS




ENTER:  CURRENT DISPLAY
PF1 : UNDEFINED           PF2 : UNDEFINED         PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE   PF6 : USER DISPLAY
PF7 : UNDEFINED           PF8 : UNDEFINED         PF9 : UNDEFINED
PF10: UNDEFINED           PF11: UNDEFINED         PF12: REMEMBER DISPLAY
```

図 48. STOP CONDITIONS の代表的な EDF 表示画面

以下のイベントのいずれかまたはすべてを STOP CONDITIONS として指定できます。

- READNEXT ファイルや ENQ リソースのような特定のタイプの機能およびオプションの検出 (例えば、FEPI ADD や GDS ASSIGN など)
- 特定のオフセットまたは特定の行番号のコマンド (プログラムが DEBUG オプションを指定して変換されていることが前提) の検出
- DL/I エラー状況または特定の DLI エラー状況の発生
- 特定の例外条件の発生。CICS 例外条件を ERROR (デフォルト) に指定している場合には、EDF は ERROR 条件 (例えば、NOTOPEN、EOF、または INVREQ) に応じて画面を再表示します。EOF のような特定の条件を指定する場合には、ANY CICS CONDITION がデフォルトの NO のままであれば、EOF 条件が起こったときだけ、EDF は画面を再表示します。

  このフィールドを YES に変更した場合には、コマンドの結果がゼロでない EIBRESP 値 (例えば、NOTOPEN、EOF、または QBUSY) を返すときはいつも、EDF は、CICS 例外条件を指定変更して、画面を再表示します。
- CICS の処置として ERROR を起こす例外条件の発生 (例えば、INVREQ や NOTFND など)
- 異常終了の発生
- タスクの正常終了
- タスクの異常終了

STOP CONDITIONS は SUPPRESS DISPLAYS 機能を使用するために常に設定する必要があるわけではありません。というのは、以下の条件のいずれかが発生した場合、普通は表示を再開したいものと想定して、EDF はそれらのフィールドをデフォルトに設定するからです。

- CICS 例外条件
- トランザクション異常終了
- タスクの正常終了
- タスクの異常終了

これらは、 198 ページの図 48 に示されているオプションです。STOP CONDITIONS メニューを表示する場合に、適用しないこれらのデフォルトをオフにすることもできますし、ユーザー・プログラム特有の条件を追加することもできます。

STOP CONDITIONS のオフセットを使用する場合には、コマンドと対応した BALR 命令のオフセットを指定しなければなりません。このオフセットは、コンパイラーまたはアセンブラーによって生成されるコード・リストから判別することができます。 COBOL、C、C++、または PL/I の場合は、アセンブラー・リストを生成するコンパイラー・オプションを使用し、関連する BALR 命令を判別する必要があります。

行番号を使用する場合には、リスト上に現れたとおりに先行ゼロも含めて正確に指定しなければならず、また、コマンドが始まる行にしなければなりません。NUM または SEQUENCE 変換プログラム・オプションを使用した場合には、変換プログラムはソースに現れるとおりの行番号を使用します。そうでない場合には、変換プログラムが行番号を割り当てます。

SOURCE または VBREF のいずれか一方の変換プログラム・オプションを使用した場合には、行番号は、変換プログラム・リスト (変換プログラムのステップの SYSPRINT) で見つけることができます。STOP CONDITIONS に行番号を使用する必要があるときに、DEBUG 変換プログラム・オプションを使用した場合には、行番号は、CALL ステートメントのパラメーターとしてコマンドの変換済み形式に組み込まれ、コンパイル (アセンブル)・リストにも現れます。

CICS コマンドと同様、DL/I コマンドで表示の抑制の停止を EDF に指示することができます。これを行うには、コマンド行で修飾子「CICS」を「DLI」で重ね書きし、抑制を停止したい DL/I コマンドのタイプを入力します。DL/I プログラムは実行中であるか、あるいは同一タスク内で既に実行されていなければなりません。プログラム開始パネルの段階で DL/I コマンドを抑制することができます。

また、特定の DL/I 状況コードが起こった場合に、抑制を停止することもできます。使用可能な状況コードについて詳しくは、「アプリケーション・プログラミング: *EXEC DLI コマンド*」マニュアルの DL/I インターフェース・ブロック (DIB) にあるコードのリストを参照してください。

SUPPRESS DISPLAYS (表示の抑制)

指定された STOP CONDITIONS の 1 つが起こるまで、すべての EDF 表示画面を抑制します。しかし、条件が起こった場合には、最初に作成された時に画面に送られていなかったとしても、10 個前までのコマンド表示画面には依然としてアクセスできます。

SWITCH HEX/CHAR (16 進/文字の切り替え)
表示を 16 進形式と文字形式の間で切り替えます。切り替えが適用できるのはコマンド表示画面のみであり、保管された前の表示画面、STOP CONDITIONS 表示画面、または作業用ストレージ表示画面には適用されません。

WHERE オプションを含んでいる DL/I コマンド表示画面では、キー値 (各比較演算子の後に続いている式) しか 16 進数に変換できません。

UNDEFINED (未定義)
指示されたファンクション・キーが、現行の代行受信点では、現行表示画面に定義されていないことを意味します。

USER DISPLAY (ユーザー表示)
EDF で、トランザクションが EDF モードで実行されなかった場合の画面上の内容を表示します (これを使用することができるのは、単一の端末チェックアウトに関してだけです)。このキーを使用した後で EDF に戻るためには、ENTER キーを押します。

WORKING STORAGE (作業用ストレージ)
プログラム内の作業用ストレージ域の内容、または CICS 領域内の他の任意のアドレスの内容を確認できます。図 49 には、標準的な作業用ストレージの画面が示されています。

```
TRANSACTION: AC20 PROGRAM: DFH0VT1 TASK: 00030 APPLID: 1234567 DISPLAY:00
ADDRESS: 035493F0                    WORKING STORAGE
035493F0   000000  E3F14040 00000000 00010000 00000000   T1   ..........
03549400   000010  00000000 00000000 F1000000 00000000   ........1.......
03549410   000020  F0000000 00000000 F0000000 00000000   0.......0.......
03549420   000030  F0000000 00000000 F0000000 00000000   0.......0.......
03549430   000040  00000000 00000000 00000000 00000000   ................
03549440   000050  D7C1D5D3 00000000 D9C5C3C4 00000000   PANL....RECD....
03549450   000060  D3C9E2E3 00000000 C8C5D3D7 00000000   LIST....HELP....
03549460   000070  84000000 00000000 A4000000 00000000   d.......u.......
03549470   000080  82000000 00000000 C4000000 00000000   b.......D.......
03549480   000090  E4000000 00000000 C2000000 00000000   U.......B.......
03549490   0000A0  D5000000 00000000 E2000000 00000000   N.......S.......
035494A0   0000B0  7B000000 00000000 6C000000 00000000   #.......%.......
035494B0   0000C0  4A000000 00000000 F1000000 00000000   ¢.......1.......
035494C0   0000D0  F2000000 00000000 F3000000 00000000   2.......3.......

ENTER:  CURRENT DISPLAY
PF1 : UNDEFINED        PF2 : BROWSE TEMP STORAGE  PF3 : UNDEFINED
PF4 : EIB DISPLAY      PF5 : INVOKE CECI          PF6 : USER DISPLAY
PF7 : SCROLL BACK HALF PF8 : SCROLL FORWARD HALF  PF9 : UNDEFINED
PF10: SCROLL BACK FULL PF11: SCROLL FORWARD FULL  PF12: REMEMBER DISPLAY
```

図 49. 作業用ストレージの代表的な EDF 表示画面

作業用ストレージの内容は、ダンプ・リストと同様の形式すなわち、16 進形式および文字表現の両方で表示されます。作業用ストレージのアドレスが画面の最上部に表示されます。スクロール・コマンドを使用して区域全体をブラウズすることができるか、あるいは画面の最上部に新規アドレスを入力することができます。このアドレスは CICS 領域内のどこであってもかまいません。**作業用ストレージ**表示画面は、2 つの追加のスクロール・キーおよび EIB (コマンドが DL/I コマンドの場合は DIB) を表示するキーを提供します。

「作業用ストレージ」の意味は、以下のように、アプリケーション・プログラムのプログラミング言語によって異なります。

COBOL
プログラムの WORKING-STORAGE セクションに定義されているすべてのデータ・ストレージ。

C、C++ および PL/I
現行プロシージャーの動的ストレージ域 (DSA)。

アセンブラー言語
現行 DFHEISTG DSECT に定義されているストレージ。

アセンブラー言語プログラムは、作業用ストレージを常に確保するわけではありません。例えば、CICS コマンドを発行しない場合には必要のないこともあります。そのようなプログラムに LINK した場合には、「Register 13 does not address DFHEISTG」というメッセージを受け取る場合があります。メッセージが必ずしもエラーを意味するわけではなく、表示する作業用ストレージがないことを表しています。

COBOL プログラムの場合以外は、作業用ストレージは標準形式の保管域で始まります。すなわち、レジスターの 14-12 はオフセット 12 で始まり、レジスター 13 はオフセット 4 に保管されています。

作業用ストレージは画面で変更することができます。16 進セクションまたは文字セクションのいずれか一方を使用することができます。また、表示画面のヘッド位置の ADDRESS フィールドは 16 進アドレスによって重ね書きすることができます。その後、そのアドレスで始まるストレージは ENTER を押すと表示されます。これにより、アドレス・スペース内のすべての位置を調べることができます。重ね書きの使用方法については、193 ページの『重ね書きによる変更』で説明しています。

調べているプログラム・ストレージが、現在実行中のプログラムの (テスト中の特定のトランザクションに固有な) 作業用ストレージの一部でない場合、画面上の対応するフィールドは保護されています。これは、別のタスクに属するストレージまたは別のタスクに影響する可能性のあるストレージにユーザーが上書きできないようにするためです。

作業用ストレージの表示行の初期部分がブランクになっている場合、そのブランク部分は作業用ストレージの一部ではありません。これは、表示画面がダブルワードで位置合わせされているために起きることがあります。

タスクの始めと終わりでは、作業用ストレージは利用不能です。これらの環境では、ユーザーがアドレス・フィールドに重ね打ちすることによって、領域内の任意のストレージ域をなおも調べることができるように、EDF がブランク・ストレージ域表示画面を生成します。

通常の非 CICS リターンによって PL/I プログラムまたは言語環境プログラムを終了する場合には、EDF はそのリターンを代行受信せず、作業用ストレージは表示されないことに注意してください。代わりに RETURN コマンドを使用すると、実行前とプログラム終了時に EDF 表示画面が表示されることができます。

言語環境が有効であるプログラムを使用している場合は、プログラムが非 CICS リターンを使用して終了すると、プログラムの終了時に作業用ストレージが解放されます。この場合、作業用ストレージを表示用に使用することはできません。

第 12 章 一時記憶域のブラウズ (CEBR)

ブラウズ・トランザクション (CEBR) は、一時記憶キューをブラウズし、それらを削除するために使用することができます。さらに CEBR トランザクションを使用して、一時データ・キューの内容を調べるために一時記憶域に転送し、終了した時に一時データ・キューを再確立することができます。これらの転送を実行する CEBR コマンドで、レコードを一時データ・キューに追加したり、一時データ・キューからすべてのレコードを除去したりすることができます。

インストール・システムによっては、特に実動システムの場合、意図していない、あるいは、許可されていない修正が行われないようにするために、CEBR トランザクションの使用を制限していることがあります。インストール・システムが、一時記憶域および一時データ・キューを含む個別のリソースを保護する場合もあります。CEBR トランザクションの使用中に、セキュリティーの失敗が原因の異常終了が起きた場合は、おそらく、ユーザー ID がアクセスを許可されていないキューに、ユーザーがアクセスしようとしたと考えられます。

この章では、以下のことについて説明します。

- 『CEBR トランザクションの使用』
- 205 ページの『CEBR トランザクションの表示内容』
- 206 ページの『CEBR ファンクション・キーの使用』
- 207 ページの『CEBR コマンドの使用』
- 210 ページの『一時データでの CEBR トランザクションの使用』

CEBR トランザクションの使用

トランザクション ID CEBR の後に、ブラウズしたいキューの名前を続けて入力することによって、CEBR トランザクションを開始します。名前は、16 文字までのものを選択することができます。例えば、AXBYQUEUENAME111 という名前の一時記憶キューを表示するためには、CEBR AXBYQUEUENAME111 を入力して、ENTER を押します。キュー名に小文字が含まれている場合は、使用している端末で大文字への変換が抑制されるようにして、正しい組み合わせの大文字小文字を入力してください。 CICS はこれに応答して、例えば 204 ページの図 50 に示すようにキューの表示画面を表示します。

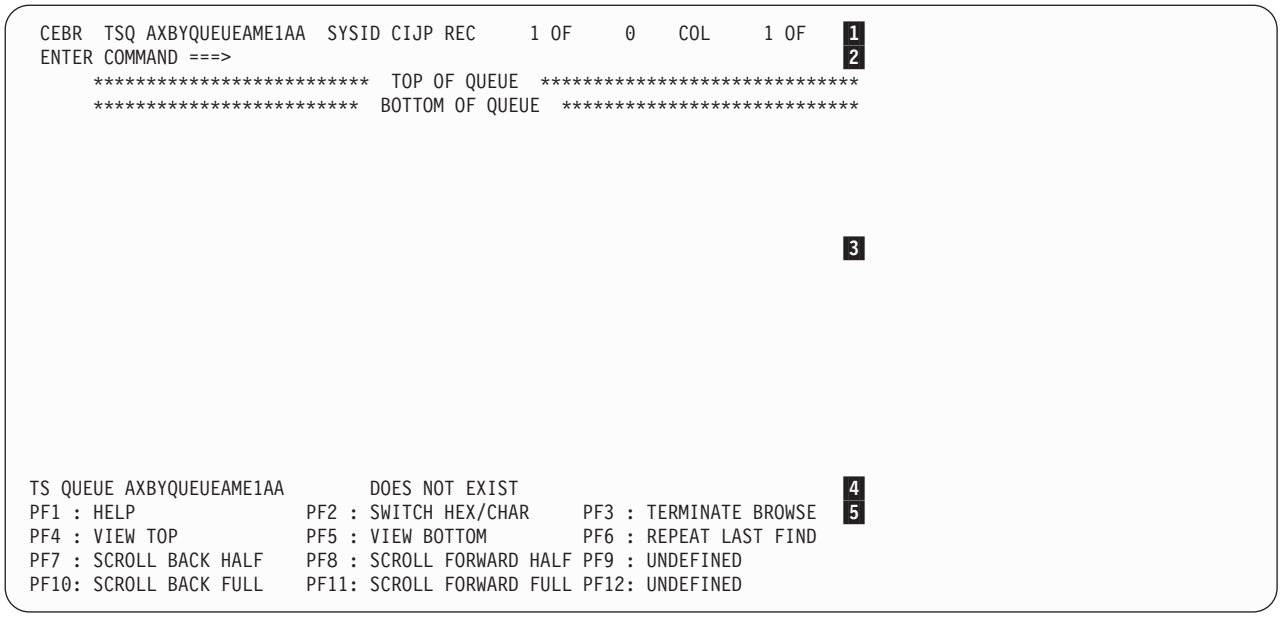
この代わりに、CEDF トランザクションから CEBR トランザクションを開始することができます。初期 CEDF 画面 (179 ページの図 35を参照してください) で PF5 を押してこれを実行します。これにより、作業用ストレージ画面が表示されるので、その画面で PF2 を押して一時記憶域をブラウズします (すなわち、CEBR トランザクションを呼び出します)。また、CEBR はブラウズされるキューで 'b' を入力して、CEMT I TSQ から開始することもできます。CEBR トランザクションは、CEBR という 4 文字の後に 4 文字の端末 ID を続けた形式の名前を持つ一時記憶キューを表示して、これに応答します (CEBR トランザクションを直接呼び出して、キュー名を指定しない場合に、CICS はこれと同じデフォルト・キュー名を使用します)。キュー名を指定しないで、または S21A 端末で EDF セッションから

© Copyright IBM Corp. 1989, 2009

CEBR トランザクションを呼び出した場合の結果が 205 ページの図 51 に示されています。CEDF トランザクションから CEBR トランザクションを入力した場合には、CEBR 画面から PF3 を押した時に EDF 画面に戻ります。

```
 CEBR  TSQ AXBYQUEUENAME111 SYSID CIJP REC     1 OF     3    COL     1 OF     5
 ENTER COMMAND ===>
      ************************  TOP OF QUEUE  *******************************
00001 HELLO
00002 HELLO
00003 HELLO
      ************************  BOTTOM OF QUEUE  ****************************









PF1 : HELP                PF2 : SWITCH HEX/CHAR     PF3 : TERMINATE BROWSE
PF4 : VIEW TOP            PF5 : VIEW BOTTOM         PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF    PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL    PF11: SCROLL FORWARD FULL PF12: UNDEFINED
```

図 50. 一時記憶キューの内容の典型的な CEBR 表示画面



1 ヘッダー
2 コマンド域
3 本体
4 メッセージ行
5 オプションのメニュー

図 51. デフォルトの一時記憶キューの典型的な CEBR 表示画面

CEBR トランザクションの表示内容

図 51 に示すように、CEBR トランザクション表示画面は、ヘッダー、コマンド域、本体 (基本表示域)、メッセージ行、およびこの時点で選択できる機能のメニューから構成されています。

ヘッダー

ヘッダーには、以下のものが表示されます。

- 実行中のトランザクション、すなわち、CEBR。
- 一時記憶キューの ID (204 ページの図 50 の AXBYQUEUEAME111 および 図 51 の AXBYQUEUEAME1AA)。画面を別のキューに切り替えたい場合には、ヘッダーのこのフィールドに重ね書きすることができます。キュー名に小文字が含まれている場合は、使用している端末で大文字への変換が抑制されるようにして、正しい組み合わせの大文字小文字を入力してください。
- 一時記憶域プール名またはリモート・システムに対応するシステム名。特に指定がなければ、ローカル・システム名が表示されます。共用またはリモート・キューをブラウズしたい場合には、ヘッダーのこのフィールドに重ね書きすることができます。
- 強調表示されたレコードの番号。

- キュー中のレコード数 (AXBYQUEUEAME111 では 3、AXBYQUEUEAME1AA ではなし)。
- 各レコードでの画面が始まる位置 (両方の場合とも 1 桁目)、および最長レコードの長さ (キュー AXBYQUEUEAME111 の場合は 22、キュー AXBYQUEUEAME1AA の場合はゼロ)。

コマンド域

コマンド域は、表示する内容および実行する機能を制御する、コマンドを入力する場所です。これらのコマンドの説明は、207 ページの『CEBR コマンドの使用』にあります。画面の最下部にあるオプションのメニューに表示されているファンクション・キーを変更することもできます。ファンクション・キーは、『CEBR ファンクション・キーの使用』で説明されています。

本体

本体は、キュー・レコードが表示される場所です。画面の各行は 1 つのキュー・レコードに対応しています。レコードが長過ぎて 1 行に入りきらない場合には、切り捨てられます。レコードの表示する部分を変更することができるので、次の画面でレコード全体を表示することができます。キューに、画面に入りきらない多くのレコードが含まれている場合には、順方向または逆方向にページを進めるか、あるいは表示を開始するレコードを指定することができるので、必要なすべてのレコードを表示することができます。

メッセージ行

CEBR は、本体とメニューの間のメッセージ行を使用して、メッセージ (例えば、205 ページの図 51 にある「Does not exist」メッセージなど) をユーザーに表示します。

CEBR ファンクション・キーの使用

常に使用できるファンクション・キーは、CEBR トランザクション画面の最下部に表示されています。キーの意味はすべての画面で同じです。端末に PF キーがない場合には、PF キーの説明の下にカーソルを置いて ENTER を押すことによって、キーと同じことが行えます。端末に 24 個のファンクション・キーがある場合には、CEBR トランザクションは PF13 ～ PF24 をそれぞれ PF1 ～ PF12 の重複として取り扱います。

PF1 HELP
CEBR トランザクションの実行中に使用できるすべてのコマンドをリストするヘルプ画面を表示します。ENTER を押すことによって、メイン画面に戻ることができます。

PF2 SWITCH HEX/CHAR
画面を文字形式から 16 進形式に切り替えたり、元に返したりします。

PF3 TERMINATE BROWSE
CEBR トランザクションを終了させます。CEBR トランザクションを直接入力した場合には、使用している端末を次のトランザクションのために解放します。EDF セッションから入力した場合には、入力した作業用ストレージ画面に戻ります。CEMT I TSQ から入力した場合には、CEMT 画面に戻ります。

PF4 VIEW TOP
キュー内の最初のレコードを表示するもので、TOP コマンドと同じ効力を持ちます。

PF5 VIEW BOTTOM
キュー内の最後のレコードを表示するもので、BOTTOM コマンドと同じ効力を持ちます。

PF6 REPEAT LAST FIND
前の FIND コマンドを繰り返します。

PF7 SCROLL BACK HALF
画面に入るレコード数の半分だけ逆方向に表示を移動して、画面の上半分にあるレコードが下半分に来るようにします。

PF8 SCROLL FORWARD HALF
画面に入るレコード数の半分だけ順方向に表示を進めて、画面の下半分にあるレコードが上半分に来るようにします。

PF9 VIEW RIGHT (または VIEW LEFT)
現在表示画面に表示されている桁のすぐ後 (右側) またはすぐ前 (左側) にある桁を表示するように、画面を変更します。レコード全体が画面の 1 行に入ってしまう場合には、このキーは未定義になります。レコードの終わりになるまで右方向に移動すると、左方向に反転してレコードの先頭に戻ります。また、COLUMN コマンドを使用して、表示を始める桁を変更することができます。

PF10 SCROLL BACK FULL
画面に入るレコード数の分だけ逆方向に画面を移動して、現在表示画面に表示されているレコードのすぐ前にあるレコードを表示します。

PF11 SCROLL FORWARD FULL
画面に入るレコード数の分だけ順方向に画面を進めて、現在表示画面に表示されているレコードのすぐ後にあるレコードを表示します。

CEBR コマンドの使用

一時記憶キューに入っているレコードの表示および処理に使用できる CEBR コマンドを、以下に示します。

BOTTOM
(省略形: B)

一時記憶キューの最後のレコードを表示します (最後のレコードが最終行になるようにして、画面の本体に入るだけレコードを表示します)。

COLUMN nnnn
(省略形: C nnnn)

各レコードの文字位置 (桁目) から開始してレコードを表示します。CEBR トランザクションを開始した場合のデフォルトの開始位置は、レコードの先頭文字です。

FIND /string
(省略形: F /string)

指定したストリングの次のオカレンスを検出します。検索は**現行レコード**の後にあるレコードから開始されます。現行レコードは強調表示されているレコードです。キューの初期表示では、現行レコードは 1 に設定されているので、検索は 2 レコード目から開始されます。

ストリングが検出された場合には、そのストリングを含むレコードが強調表示行となり、このレコードを 2 行目に表示するように表示が変更されます。正常な FIND の後で検索ストリングが表示されない場合には、それは表示上にあるレコードの桁にないからです。スクロール・キーまたは COLUMN コマンドを使用して表示画面を右方向または左方向に桁移動してストリングを表示してください。

例を以下に示します。

```
FIND /05-02-93
```

このコマンドは、ストリング「05-02-93」の次のオカレンスを見付けます。**/** 文字は区切り文字です。**/** にする必要はありませんが、検索引数に現れる文字であってはなりません。例えば、検索しているストリングが 『05-02-93』 ではなく、『05/02/93』 であった場合には、次のコマンドを使用することはできません。

```
FIND /05/02/93
```

検索ストリングに斜線 (/) がある場合、次の例は正常に機能します。

```
FIND X05/02/93      または      FIND S05/07/93
```

/ またはストリングのいずれかの数字以外の区切り文字であれば、機能します。検索ストリングにスペースがある場合には、ストリングの終わりにも区切り文字を繰り返さなければなりません。例を以下に示します。

```
FIND /CLARE JACKSON/
```

検索ストリングでは大文字小文字の区別はありません。FIND コマンドを入力した場合は、PF6 を押して FIND を繰り返す (すなわち、次に出てくるストリングを見つける) ことができます。

GET xxxx
(省略形: G xxxx)

指定の一時データ・キューを、画面上に現在表示されている一時記憶キューの終わりに転送します。これによって、キューの内容をブラウズすることができます。xxxx は、区画内一時データ・キューの名前、または入力用にオープンされている区画外一時データ・キューの名前でなければなりません。一時データ・キューのブラウズの詳細については、210 ページの『一時データでの CEBR トランザクションの使用』を参照してください。

LINE nnnn
(省略形: L nnnn)

画面の本体を nnnn の 1 つ前のキュー・レコードから開始して、現在行を nnnn に設定します (このような配置によって、後続の FIND コマンドがレコード nnnn の後から検索を開始することになります)。

PURGE
ブラウズ中のキューを削除します。

BMS 論理メッセージなど、内部生成されたキューの内容を、PURGE を使用して削除してはなりません。

注: リカバリー可能な一時記憶キューをパージする場合には、ユーザー・タスクを終了してからでないと、他のタスクがそのキューを更新 (レコードの追加、レコードの変更、またはパージ) することはできません。

PUT xxxx
(省略形: P xxxx)

ブラウズ中の一時記憶キューを指定の一時データ・キューにコピーします。xxxx は、区画内一時データ・キューの名前、または出力用にオープンされている区画外一時データ・キューの名前でなければなりません。一時データ・キューの作成または復元の詳細については、210 ページの『一時データでの CEBR トランザクションの使用』を参照してください。

QUEUE xxxxxxxxxxxxxxxx
(省略形: Q xxxxxxxx)

ブラウズ中のキューの名前を変更します。指定する値は、16 文字までの文字形式 (例えば、QUEUE ABCDEFGHIJKLMNOP など) でも 16 進形式 (例えば、QUEUE X'C1C2C3C4' など) でもかまいません。キュー名に小文字が含まれている場合は、使用している端末で大文字への変換が抑制されるようにして、正しい組み合わせの大文字小文字を入力してください。 CEBR トランザクションは、指定のキューにあるデータを表示して、これに応答します。

キュー名は、ヘッダーの現行値に重ね書きすることによって変更することもできます。

SYSID xxxx
(省略形: S xxxx)

キューを見付ける一時記憶域プール名またはリモート・システム名を変更します。

この名前は、ヘッダーの現行の SYSID 値に重ね書きすることによって変更することもできます。

注: CEBR トランザクションが実行されている CICS システムで、ISC がアクティブでない場合には、SYSID はデフォルトであるローカル SYSID となります。

TERMINAL xxxx
(省略形: TERM xxxx)

ブラウズ中のキューの名前を変更しますが、端末と関連付けられた、一時記憶キューの命名規則 (最初の 4 文字に定数、最後の 4 文字に端末名を使用する) を使用するアプリケーションに合わせたものです。新しいキュー名は、現行キュー名の最初の 4 文字の後に、xxxx を付けた形式になります。

TOP
(省略形: T)

CEBR トランザクションが、キューの最初のレコードから表示画面を開始します。

一時データでの CEBR トランザクションの使用

GET コマンドは、指定された一時データ・キューの各レコードを読み取り、ブラウズ中の一時記憶キューの終わりにそれを書き込みます。これは一時データ・キューが空になるまで行われます。その後、一時データ・キューにあったレコードを表示することができます。検査を終了した時には、一時記憶キューを一時データ・キューにコピーして返します (PUT コマンドを使用して)。この場合には、一時データ・キューは表示された時の内容と通常は同じですが、常に同じではありません。GET および PUT コマンドを使用する時には、以下の点に注意する必要があります。

- 一時データ・キューをブラウズした後で、変更しないでそのまま復元したい場合には、必ず、GET コマンドの時点で表示画面の一時記憶キューが空になるようにします。そうでない場合には、既存の一時記憶域レコードが、後続の PUT コマンドが発行された時に一時データ・キューにコピーされます。
- 一時データ・キューを取得した後で、それを返す前に、他のタスクがその一時データ・キューに書き込んでいる場合があります。PUT コマンドを発行した時には、新しいレコードの**後に**、一時記憶キューのレコードがコピーされるので、キューのレコードは、最初に作成された順序ではなくなっています。アプリケーションによっては、キューのレコードを順次処理することを前提としているものもあります。
- **リカバリー可能な**一時データ・キューを取得した後は、トランザクションが終了するまで、他のタスクはそのキューにアクセスすることはできません。CEDF トランザクションから CEBR トランザクションに入った場合には、CEDF トランザクションを終了しなければなりません。ただし、疑似会話型のトランザクションのシーケンスをデバッグしている場合には、「continue」(続行するかどうか) の質問に「yes」と応答することができます。CEBR トランザクションを直接呼び出す場合には、それを終了しなければなりません。
- 同様に、リカバリー可能一時データ・キューに PUT コマンドを発行した場合には、トランザクションが終了するまで、他のタスクはそのキューにアクセスすることはできません。

GET および PUT コマンドは対にして使用する必要はありません。PUT コマンドによって、常に一時記憶キューから一時データ・キューに追加することができます。一時データ・キューを読み取るコードをデバッグしている場合には、(CECI トランザクションまたは CEBR GET コマンドを指定して、またはプログラムによって) 一時記憶域にキューを作成して、一時記憶域から必要な回数だけ一時データ・キューを最新のものに更新することができます。同様に、対応する PUT コマンドなしで GET コマンドを使用して一時データ・キューを空にすることができます。

第 13 章 コマンド・レベル・インタープリター (CECI)

コマンド・レベル・インタープリター (CECI) トランザクションを使用して、CICS コマンドの構文を検査し、3270 画面でこれらのコマンドを対話式に処理できます。CECI を使うと、大部分のコマンドを実行させて、その結果を表示することができます。また、CICS コマンド・レベル・アプリケーション・プログラミングおよびシステム・プログラミング・インターフェース全体の構文を参照することができます。

CECI では、テスト・システムと対話して、テスト・データ、一時記憶キューの作成または削除を行ったり、あるいは間違ったデータを故意に使用してエラー・ロジックをテストすることができます。また、CECI を使用して、実動システムの破壊されたデータベース・レコードを修理することもできます。

インタープリターは非常に強力なツールなので、ご使用のシステムが接続時セキュリティーでその使用を制限している可能性があります (ご使用のシステムで使用される外部セキュリティー・マネージャーが、CECI および CECS トランザクションに対するセキュリティー属性を定義します)。このような制限が行われている場合に、選択したインタープリター・トランザクションの使用が許可されていない場合には、トランザクションを開始することができません。

この章では、以下のことについて説明します。

- 『CECI による表示内容』
- 217 ページの『CECI の使用』
- 219 ページの『CECI ファンクション・キーの使用』
- 222 ページの『コマンドの保管』
- 223 ページの『CECI の実行方法』

CECI による表示内容

CECI 画面はすべて同じ基本レイアウトをもっています。 218 ページの図 54 に示されているように、CECI 表示画面は以下のものから構成されています。

- 『コマンド行』
- 212 ページの『状況表示行』
- 215 ページの『本体』
- 216 ページの『メッセージ行』
- 216 ページの『ファンクション・キーにおける CECI オプション』

コマンド行

コマンド行は画面の 1 行目です。ここには、処理したいコマンドまたは構文を検査したいコマンドを入力します。これは完全な構文または省略した構文とすることができます。

コマンドの入力および省略方法の規則は以下のとおりです。

- EXEC CICS キーワードの指定は任意です。

© Copyright IBM Corp. 1989, 2009

- コマンドのオプションは、固有のものとするのに十分な文字数まで省略することができます。有効な省略形は、画面の本体に表示される構文で大文字で示されています。
- 文字ストリングは引用符で囲んでも囲まなくてもかまいませんが、すべて、その前にアンパーサンド (&) を付けない限り、文字ストリング定数として扱われます。付けた場合は、変数として扱われます。
- コマンドが処理された時に CICS から値を受け取るコマンドのオプションは、**レシーバー**と呼ばれ、指定する必要はありません。CICS から受け取った値は、構文表示に組み込まれ、変数を指定した場合には、コマンドが処理された後で変数に保管されます。
- 対立する 2 つのキーワードを持つ CECI コマンドを発行した場合には、CECI は最初のキーワードを無視してエラー・メッセージを出します。例えば、READ コマンドの場合は次のようなメッセージです。

  ```
  E INTO option conflicts with SET option and is ignored
  ```
- コマンドの前に疑問符 (?) を入れると、トランザクション・コード CECI を使用した場合でも、インタープリター・プログラムは構文検査の後で停止します。実行を続けたい場合には、疑問符 (?) を除去します。

次の例はコマンドの省略した形式を示しています。ファイル制御コマンドは、次のとおりです。

```
EXEC CICS READ FILE('FILEA') RIDFLD('009000') INTO(&REC)
```

コマンド入力行に、次のように入力することができます。

```
READ FIL(FILEA) RID(009000)
```

また、最短形式では次のようになります。

```
READ F(FILEA) RI(009000)
```

最初の形式では、INTO の指定があるので、変数 &REC が作成され、その中にデータが読み込まれます。ただし、INTO はレシーバー (前述に定義された通り) なので、省略することができます。省略した場合には、CICS が自動的に変数を作成します。

状況表示行

コマンドの解釈の処理を進めていくと、CECI は一連の表示画面を表示します。画面の本体の形式は、本質的にすべて同じです。本体には、コマンドの構文および選択したオプション値が表示されます。これらの画面の状況表示行は、コマンド処理の中の位置を示すもので、以下のもののうちの 1 つです。

- COMMAND SYNTAX CHECK
- ABOUT TO EXECUTE COMMAND
- COMMAND EXECUTION COMPLETE
- COMMAND NOT EXECUTED

これらのどの画面からも、追加の表示画面を選択することができます。それを実行すると、画面の本体には要求された情報が表示され、状況表示行によって以下のいずれかの表示画面として識別されます。

- EXPANDED AREA

- VARIABLES
- EXEC INTERFACE BLOCK
- SYNTAX MESSAGES

これらの画面については、 219 ページの『CECI ファンクション・キーの使用』で説明しています。これらの表示画面は、処理中いつでも要求することができます。その後、コマンド解釈シーケンスに戻ることができます。

状況表示行には、この他に NAME= という入力フィールドが 1 つあります。このフィールドは、変数を作成して、それに名前を付けるために使用されます。これについては、 219 ページの『変数』および 222 ページの『コマンドの保管』で説明します。

コマンド構文検査

状況表示行が (218 ページの図 54) に示されているような) **コマンド構文検査**であることを示している場合、コマンド入力行に入力されたコマンドは構文検査されているが、処理はまだ開始されないことを示しています。CECS を入力した場合、あるいはコマンドの前に疑問符 (?) を付けた場合には、常にこの状況となります。また、コマンドの構文検査によって重大エラー・メッセージが出された場合にも、この状況になります。

さらに、インタープリターが実行できないコマンドの 1 つを実行しようとした場合にも、この状況になります。CECS または CECI を使用して、コマンドを構文検査することができますが、インタープリターでは、以下のコマンドについてこれ以上処理することはできません。

- インタープリターが提供していない環境に依存する EXEC CICS コマンドは以下のとおりです。
  - FREE
  - FREEMAIN
  - GETMAIN
  - HANDLE ABEND
  - HANDLE AID
  - HANDLE CONDITION
  - IGNORE CONDITION
  - POP HANDLE
  - PUSH HANDLE
  - SEND LAST
  - SEND PARTNSET
  - WAITCICS
  - WAIT EVENT
  - WAIT EXTERNAL
- 区分画面を参照する BMS コマンド (画面が区分けされた後では、表示画面を復元することができないためです)
- EXEC DLI
- CPI 通信 (CPI-C) コマンド

- SAA リソース・リカバリー・インターフェース (CPI-RR) コマンド

コマンド実行開始

この (図 52 に示されているような) 表示画面は、**command syntax check (コマンド構文検査)** で停止する理由が設定されなかった場合に現れます。

```
READ FILE('FILEA') RIDFLD('009000')
STATUS:  ABOUT TO EXECUTE COMMAND                          NAME=
 EXEC CICS  READ
  File( 'FILEA   ' )
  < SYsid() >
  SEt() | Into()
  < Length() >
  RIdfld( '009000' )
  < Keylength() < GEneric > >
  < RBa | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < Update < Token() > >









PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

図 52. コマンド実行開始の典型的な CECI 表示画面

この時点で画面を変更しないで ENTER キーを押した場合には、CECI はコマンドを実行します。ただし、この時点でもまだ修正することができます。変更した場合には、CECI は前のコマンドを無視し、スクラッチから新しいコマンドを処理します。これは、次に表示される画面が、コマンドを実行できない場合には **command syntax check (コマンド構文検査)**、コマンドが正しい場合には **about to execute command (コマンド実行開始)** であることを意味します。

CICS コマンドの完了

この画面 (215 ページの図 53 に示すような画面) は、変更されていない「**about to execute command (コマンド実行開始)**」の画面の ENTER キーに応えて、インタープリター・プログラムがコマンドを実行した後に表示されます。

```
 INQUIRE FILE NEXT
 STATUS:  COMMAND EXECUTION COMPLETE                        NAME=
  EXEC CICS  INquire File( 'DFHCSD  ' )
   < STArt | END | Next >
   < ACcessmethod( +0000000003 ) >
   < ADd( +0000000041 ) >
   < BAsedsname( '                                          ' ) >
   < BLOCKFormat( +0000000016 ) >
   < BLOCKKeylen( -0000000001 ) >
   < BLOCKSize( -0000000001 ) >
   < BRowse( +0000000039 ) >
   < Cfdtpool( '        ' )
   < DElete( +0000000043 ) >
   < DIsposition( +0000000027 ) >
   < DSname( 'CFV01.CICS03.PSK.CSD                       ' ) >
   < EMptystatus( +0000000032 ) >
   < ENAblestatus( +0000000033 ) >
   < EXclusive( +0000000001 ) >
   < Fwdrecstatus( +0000000361 ) >
+  < Journalnum( +00000 ) >


  RESPONSE: NORMAL            EIBRESP=+0000000000 EIBRESP2=+0000000000
PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

図 53. コマンド実行完了の典型的な CECI 表示画面

コマンドの処理が終わって、結果が画面に表示されます。

レシーバーは、指定されていてもいなくても、CICS 提供の値とともに強調表示されます。

本体

「**command syntax check (コマンド構文検査)**」、「**about to execute command (コマンド実行開始)**」、および「**command execution complete (コマンド実行完了)**」の各画面の本体には、これら 3 つの表示画面に共通の情報が含まれています。

コマンドの完全な構文が表示されます。コマンド行に指定されたオプションまたはデフォルトによって設定されるオプションは、**レシーバー**と同様にコマンドの実行に使用されることを示すために、強調表示されます。大括弧の **< >** は、これらの大括弧の中からオプションを選択できることを示しています。構文にエラーがあった場合には、CECI は本体に続くメッセージ領域 (216 ページの『メッセージ行』で説明されている) でそれを診断します。複数の診断メッセージがある場合には、残りのメッセージは PF9 を使用して表示することができます。

引数は、文字または 16 進形式で表示することができます。形式を切り替えるためには、PF2 を使用することができます。文字形式では、一部の文字 (一部の端末では小文字も含めて) が表示されません。CECI はピリオドとしてそれを表示します。実際の値を表示するには、16 進数に切り替える必要があります。また、それを修正する場合は、 218 ページの『変更方法』 で説明されているように、注意が必要です。

オプションの値がその行に対して長過ぎる場合には、最初の部分しか表示されませんが、まだ続くことを示すために「...」が後に続けて表示されます。カーソルをオプション値の始めに位置決めして ENTER キーを押すことによって値全体を表示することができます。これにより、 219 ページの『拡張域』で説明している拡張表示画面が作成されます。

コマンドに 1 つの画面に収まるより多くのオプションがある場合には、現行表示画面の最後のオプションの左側に、さらにオプションが続くことを示す正符号 (+) が表示されています。この例は 215 ページの図 53 に示されています。PF キーでスクロールすることによって追加のページを表示することができます。

メッセージ行

CECI はメッセージ行を使用してエラー・メッセージを表示します。コマンドの実行後には、メッセージ行にその応答コードが表示されます。 218 ページの図 54 には、ユーザーが必須フィールドを省略した場合のエラー・メッセージが示されています。メッセージの前にある「**S**」は、(実行が不能な) 重大エラーであることを示します。また、他に警告メッセージ (**W** のフラグが付けられた) およびエラー・メッセージ (**E** のフラグが付けられた) がありますが、これらは情報を提供するだけで、実行を続けられないわけではありません。**E** メッセージは、オプションの組み合わせが正しくなく、予定通りの結果にならず、実行を続ける前にコマンドを検討する必要があることを示しています。

複数のエラー・メッセージがある場合には、CECI は、すべてのメッセージを含む別の表示画面を作成し、メッセージ行を使用してエラーの数およびその重大度を通知します。 219 ページの『CECI ファンクション・キーの使用』 に説明されているように、PF9 を使用するとメッセージを表示できます。

215 ページの図 53 に示されているのは、コマンドの実行結果を示すというメッセージ行の 2 つ目の使用方法です。CECI は、テキスト (215 ページの図 53 の例では NORMAL) および 10 進形式 (EIBRESP および EIBRESP2 値) の両方で情報を提供します。

ファンクション・キーにおける CECI オプション

画面の最下部の一行に、表示画面の PF キーの効果を示すメニューが提供されています。

ここでは PF キーについて説明します。端末に PF キーがない場合には、メニューの必要な項目の下にカーソルを位置決めして ENTER キーを押すと、同じ効果が得られます。

PF1 HELP
コマンド・インタープリターの使い方および PF キーの意味について詳しく説明した HELP パネルを表示します。

PF2 HEX
(SWITCH HEX/CHAR) 16 進形式と文字形式の間で表示画面を切り替えます。これはモード切り替えです。次にこのキーを押すまで後続のすべての画面がここで選択したモードとなります。

PF3 END
(END SESSION) インタープリターの現行セッションを終了させます。

PF4 EIB
(EIB DISPLAY) EXEC インターフェース・ブロック (EIB) の内容を表示します。この画面の例は 221 ページの図 56 に示されています。

PF5 VAR
(VARIABLES) 現行コマンド・インタープリター・セッションと関連したすべての変数を、その名前、長さ、および値を提供して表示します。この PF キーの使用について詳しくは、 219 ページの『変数』を参照してください。

PF6 USER
(USER DISPLAY) ユーザー画面の現在の内容 (すなわち、それまでに処理されたコマンドがインタープリターではなく、通常のプログラムによって実行されていた場合に、端末に表示される内容) を表示します。このキーは、SEND MAP などの端末コマンドが実行されて初めて意味をもちます。

PF7 SBH
(SCROLL BACK HALF) 本体を 1/2 画面分逆方向にスクロールします。

PF8 SFH
(SCROLL FORWARD HALF) 本体を 1/2 画面分順方向にスクロールします。

PF9 MSG
(DISPLAY MESSAGES) コマンドの構文検査時に生成されたメッセージをすべて表示します。

PF10 SB
(SCROLL BACK) 本体を画面 1 つ分逆方向にスクロールします。

PF11 SF
(SCROLL FORWARD) 本体を画面 1 つ分順方向にスクロールします。

CECI の使用

2 つのトランザクション ID、CECS または CECI のどちらかの後に、テストしたいコマンドの名前を続けて入力することによって、コマンド・レベル・インタープリターを開始します。

コマンド・オプションをリストすることもできますが、後からこれを実行することもできます。例を以下に示します。

```
CECS READ FILE('FILEA')
```

または

```
CECI READ FILE('FILEA')
```

CICS は、 218 ページの図 54 に示されているように、コマンドおよびそれと関連した機能、オプション、および引数を表示して応答します。コマンドを忘れた場合には、CECI が使用可能なコマンドのリストを用意しているので、これを使って開始することができます。プログラミング用に 「*CICS アプリケーション・プログラミング・リファレンス*」や 「*CICS System Programming Reference*」で説明されているコマンドは、どれでも使用することができます。

```
READ FILE('FILEA')
STATUS:  COMMAND SYNTAX CHECK                                  NAME=
 EXEC CICS  READ
  File( 'FILEA   ' )
  < SYsid() >
  ( SEt() | Into() )
  < Length() >
  RIdfld( '009000' )
  < Keylength() < GEneric > >
  < RBa | Xrba | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < UNcommitted | Consistent | REpeatable | Update  < Token() > >
  < Nosuspend >




PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

図 54. コマンド構文検査の典型的な CECI 表示画面

トランザクション・コード CECS を使用した場合、インタープリターはコマンドの構文が正しいかどうかを検査します。CECI を使用した場合には、構文が正しい場合に 1 回コマンドを実行するオプションがあります。CICS は 2 つのトランザクション ID を使用して、構文検査と実行に異なるセキュリティーを割り当てることができます。

変更方法

CICS がコマンドを実行するまでは、コマンド行の内容を変更するか、本体の構文表示画面に表示されているオプション値を変更するか、または**変数**画面上の変数の値を変更することによって、コマンドを変更することができます。(また、コマンドを実行した後でも変更できますが、別のコマンド用に準備されるものでない限り、効果はありません。)

コマンド行または**変数**画面上で変更を行った時には、CECI トランザクションが続く間、変更内容は有効です。ただし、構文画面の本体に変更を行った時には、変更は一時的なものです。これらの変更内容が有効なのは、コマンドが実行されるまでだけであり、コマンド行には影響しません。

既に述べた通り、すべての端末ですべての文字が表示できるわけではありません。表示画面が 16 進形式ではなく文字形式である場合には、CECI はこれらの文字をピリオド (X'4B') として表示します。ピリオドの上に重ね書きする場合には、現行値がピリオドではなく表示不能文字であることを認識しておいてください。

さらに、表示画面が文字モードの場合には、文字をピリオドに変更することはできません。ピリオドに変更しようとしても、CECI は変更を無視し、診断メッセージも発行しません。このような変更を行うには、表示画面を 16 進数に切り替えて、ピリオドを表す値 (X'4B') を入力する必要があります。

同様に、16 進形式で変更する場合にも、制約事項があります。文字をブランクに変更する必要がある場合、16 進数表示画面からコード (X'40') を入力することはできません。再度、変更は無視され、CECI はメッセージを発行しません。その代わりに、文字モードに切り替えてから、文字をブランクにしなければなりません。

変更が行われるたびに、CECI は構文を検査してエラーがないことを確認します。実行ストッパーがある場合には、**command syntax check (コマンド構文検査)** から処理が再始動され、ない場合には、**about to execute command (コマンド実行開始)** から処理が再始動されます。変更されていない「**about to execute command (コマンド実行開始)**」画面で ENTER を押した場合にのみ、CECI はコマンドを実行します。

CECI ファンクション・キーの使用

追加の情報画面は、該当する PF キーを押した時に利用可能となります。これについては、216 ページの『ファンクション・キーにおける CECI オプション』で説明しています。変更していない画面で ENTER キーを押して、元の画面に戻ることができます。

拡張域

この表示画面は、カーソルで選択した変数を表示するために、画面の本体全体を使用します。カーソルは、構文表示画面のオプションの値の始め、または変数表示画面の変数のアンパーサンドの下に位置付けることができます。ENTER を押すと、拡張域表示画面が表示されます。情報がフルスクリーンに収まらない場合には、スクロール・キーを使用してすべての情報を表示することができます。

変数

図 55 は、PF5 を押して、**変数**表示画面の要求の結果表示された画面を示しています。現行のインタープリター・セッションと関連した各変数ごとに、この画面は、その名前、長さ、および値を表示します。

```
 READ FILE('FILEA') RIDFLD('009000')INTO(&REC)
 VARIABLES   LENGTH   DATA
&DFHC        +00016   THIS IS A SAMPLE
&DFHW        +00046   EXEC CICS WRITEQ QUEUE('  CIS200') FROM(&DFHC)
&DFHR        +00045   EXEC CICS READQ QUEUE('  CIS200') INTO(&DFHC)
&REC         +00080   482554 D694 72 WIDGET, .007 TEST 100









PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER            9 MSG
```

図 55. CECI セッションと関連した変数の典型的な CECI 表示画面

最初に表示される 3 つの変数は CECI によって作成され、明示的にそれを削除しない限り、常に表示されます。これらは、 222 ページの『コマンドの保管』で説明しているコマンド・リストの作成に役立つように、また例になるように設計されています。

これらの 3 つの後で、ユーザーが作成した変数が表示されます。 219 ページの図 55 の 4 番目の例、&REC は次のコマンドを実行した結果です。

```
READ FILE('FILEA') RID('009000') INTO(&REC)
```

通常、コマンド行のオプションに提供された値は文字ストリング定数として処理されます。ただし、場合によっては、2 つのコマンドをオプション値によって接続したい時など、この値を表す変数を指定する必要があることがあります。

例えば、CECI を使ってレコードを変更する場合は、まず初めに次のように入力します。

```
EXEC CICS READ UPDATE INTO(&REC)
          FILE('FILEA') RID('009000')
```

そして、変数 &REC を変更することによって、レコードに、必要な変更を加えてから、次のように入力します。

```
EXEC CICS REWRITE FROM(&REC) FILE('FILEA')
```

1 桁目のアンパーサンド (&) によって、変数を指定していることを CECI に指示しています。

また、変数は、オプションの値によってコマンドの長さがコマンド入力域の行の長さを超える場合にも、有用です。必要な値を入れて変数を作成し、コマンドにその変数を指定すると、行の長さの制限を超えずに済みます。

変数の定義

変数のデータ・タイプは、フルワード、ハーフワード、またはパック 10 進数とすることができます。作成方法としては、以下のものがあります。

- レシーバーの変数に命名する (例えば、 219 ページの図 55 における &REC など)。コマンドが処理された時に、変数が作成されています。データ・タイプおよび長さは、オプションによって暗黙に指定されます。
- 既に定義された変数のリストに新しい項目を追加する。新しい変数を作成するためには、変数画面の最初の未使用行の適切な桁に、変数の名前および長さを入力してから、ENTER キーを押します。文字変数の場合には、変数が定義されている長さを使用します。フルワードまたはハーフワードの場合には、**F** または **H** を入力します。パック変数の場合には、バイト単位の長さを使用し、前に **P** を付けます。

  文字変数はブランクに初期設定されます。それ以外の変数は、適切な形式でゼロに初期設定されます。変数を作成すると、**変数**表示画面でデータ・フィールドを修正することによって値を変更することができます。
- 特定のオプションの拡張域表示画面を表示した時に、状況表示行の NAME フィールドを使用する。この場合には、構文表示画面でオプションの下にカーソルを

位置決めして、ENTER キーを押します。次に NAME フィールドに変数名を入力してもう一度 ENTER キーを押すことによって、表示されたオプション値と関連付けたい変数名を割り当てます。

- 既存の変数をコピーする。この場合には、コピーする変数の拡張域表示画面を表示して、表示されている名前の上に新しい変数の名前を重ね書きしてから ENTER キーを押します。
- 構文表示画面で NAME フィールドを直接使用する。これによって、コマンド行に文字ストリングの内容を持つ文字変数が作成されます。この変数は、 222 ページの『コマンドの保管』で説明しているコマンド・リストで使用されます。

また、変数は削除することもできますが、通常はその必要がありません。これはセッション終了時に CECI によってすべての変数が廃棄されるからです。セッションの終了前に変数を削除するためには、カーソルを名前の始めにあるアンパーサンドの下に位置付けて ERASE EOF を押してから ENTER キーを押します。

EXEC インターフェース・ブロック (EIB)

PF4 を押すことによって CECI トランザクションと関連した EIB を表示することができます。図 56 には、EXEC インターフェース・ブロック (EIB) の内容の例を示してあります。

```
 READ FILE('FILEA') RIDFLD('009000')
 EXEC INTERFACE BLOCK
   EIBTIME      = +0124613
   EIBDATE      = +0091175
   EIBTRNID     = 'CECI'
   EIBTASKN     = +0000046
   EIBTRMID     = 'S200'
   EIBCPOSN     = +00004
   EIBCALEN     = +00000
   EIBAID       = X'7D'
   EIBFN        = X'0000'
   EIBRCODE     = X'000000000000'
   EIBDS        = '........'
   EIBREQID     = '........'
   EIBRSRCE     = '        '
   EIBSYNC      = X'00'
   EIBFREE      = X'00'
   EIBRECV      = X'00'
   EIBATT       = X'00'
   EIBEOC       = X'00'
+  EIBFMH       = X'00'


PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

図 56. EIB の典型的な CECI 表示画面

エラー・メッセージの表示画面

CECI がメッセージ行に表示できるメッセージより多くのメッセージがある場合には、PF9 を押せば、そのすべてを表示することができます。

```
READ
SYNTAX MESSAGES
 S FILE must be specified.
 S RIDFLD must be specified.









PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH       10 SB 11 SF
```

図 57. メッセージ表示の典型的な CECI 表示画面

コマンドの保管

場合によっては、CECI のもとで 1 つのコマンドまたは一連のコマンドを繰り返し実行したい場合もあります。これを行うための 1 つの手法として、コマンドを含んだ一時記憶キューを作成する方法があります。そして、そのキューから、コマンドの読み取りと実行を交互に行います。

CECI は、キューを作成してそこからコマンドを実行するためのショートカットを用意しています。キューを作成するには、以下のようにします。

1. CECI セッションを開始します。
2. コマンド行に保管したい最初の (または次の) コマンドを入力して、状況表示行の NAME フィールドに &DFHC を入れてから ENTER を押します。この処置によって、通常の構文検査が行われ、さらに CECI が常に定義する 3 つの変数の最初のものである &DFHC の値としてコマンドが保管されます。 (219 ページの図 55を参照してください。) この時点で変数表示画面を選択した場合には、&DFHC がコマンドの値になっていることが表示されます。
3. 構文が正しい場合は、(**about to execute command (コマンド実行開始)** 画面で) 実行する前に、コマンド行を &DFHW に変更して ENTER を押します。これにより、CECI は実行コマンドに &DFHW の値を使用します。 &DFHW は CECI が提供する 2 番目の変数であり、ここには変数 &DFHC の内容 (すなわち、実行するコマンド) を「 CItttt」という名前の (この「tttt」は端末の名前であり、文字「CI」前の 2 つのブランクがあります) 一時記憶キューに書き込むコマンドが収容されています。
4. この WRITEQ コマンドを (**command execution complete (コマンド実行完了)** 画面から) 実行します。これによって、実行するコマンドがキューに保管されます。

5. 複数のコマンドを保管したい場合には、それぞれのコマンドに対して 2 (222 ページ) から 4 (222 ページ) のステップを繰り返します。

リストから保管しておいたコマンドを実行したい場合は、以下のようにします。

1. コマンド行に &DFHR を入力して ENTER を押します。&DFHR は CECI 提供の最後の変数であり、既に書き込まれているキューを読み取るコマンドが入っています。このコマンドを実行すると、保管しておいた最初の (または次の) コマンドが変数 &DFHC に入ります。
2. コマンド行に &DFHC を入力して ENTER を押します。CECI は &DFHC の値 (保管しておいたコマンド) でコマンド行を置き換えます。ENTER を押して、コマンドを実行します。
3. 保管しておいたコマンドがすべて実行されるまで、コマンド行に &DFHR と &DFHC を交互に入力しながら、上記 2 つのステップを繰り返します。

この手順は必要に応じて変更することができます。例えば、ステップ (2) をスキップするだけで、手順の中のコマンドをスキップすることができます。通常に入力したコマンドと同じ方法で、保管したコマンドのオプションを、実行前に変更することができます。

保管済みコマンド列を繰り返し実行したい場合は、キューの先頭から読み取るように位置変更するために、READQ コマンドを最初に実行するときにオプション ITEM(1) を指定する必要があります。

CECI の実行方法

インタープリターを正しく使用するには、その働き方についていくつかのことを知っておく必要があります。以下のことが含まれます。

- CECI セッション
- 異常終了
- 例外条件
- プログラム制御コマンド
- 端末の共用
- コマンドの保管
- 共用ストレージ

CECI セッション

インタープリターは、CICS 提供のプログラムを使用してトランザクションとして実行されます。これは会話型であり、セッションの開始 (CECI を入力する) と終了 (PF3) の間で行われることはすべて、1 つのタスクの 1 つの論理単位であることを意味します。これは、実行するコマンドによって起こるロックおよびエンキューは、セッションの間、その実行状態が続くということを意味します。例えば、リカバリー可能ファイルから更新用にレコードを読み取る場合には、CECI を終了するまで、他のタスクでそのレコードを使用することができません。

異常終了

CECI は、すべてのコマンドを実行するときに NOHANDLE オプションを指定しているので、実行エラーによって異常終了が引き起こされることは、通常はありません。

さらに CECI は、セッションの始めに HANDLE ABEND コマンドも出して、異常終了が起こった場合でも制御権を失わないようにします。したがって、異常終了が起こっても、CECI がそれを処理するので、リソース・バックアウトはありません。保護リソースに相互に関連した一連の更新を行っている場合には、更新しないリソースがいくつかある場合には、その他の更新も完了できないということに注意する必要があります。この状態にあることがわかった場合には、SYNCPOINT ROLLBACK コマンド、または CANCEL オプションを指定して ABEND コマンドを実行して、リカバリー可能リソースでこれより前のコマンドの効果を除去することができます。

例外条件

一部のコマンドでは、指定されたオプションがすべて正しい場合でも、CECI が例外条件を返すことがあります。これは、一部のコマンドで、CECI が明示的に指定しないオプションを使用するために起こります。例えば、CECI のもとでは、ASSIGN コマンドは常に例外条件 INVREQ を返します。CECI は、要求された情報を正しく戻すことができた場合でも、他のオプションから情報を入手しようとするので、その一部に無効なものが含まれることがあります。

プログラム制御コマンド

インタープリターはそれ自身がアプリケーション・プログラムなので、一部のプログラム制御コマンドは、解釈した結果がこれらのコマンドをアプリケーション・プログラムで実行した結果とは異なる場合があります。例えば、ABEND コマンドは、CANCEL オプションを使用しない限り、上で説明したように代行受信されます。

LINK コマンドを実行した場合には、ターゲット・プログラムは実行されますが、インタープリターの環境では、予期した通りにならない場合があります。特に、リンクされたプログラムでユーザー表示画面を修正した場合には、インタープリターが変更内容を認識することはできません。

同様に、XCTL コマンドを解釈する場合には、CECI は指定されたプログラムに制御権を渡したままで、制御権が戻ってくることはないので、CECI セッションは終了します。

端末の共用

解釈されるコマンドが、インタープリターが使用しているのと同じ画面を使用する場合、コマンド・インタープリターは、インタープリター表示画面とユーザー表示画面の間で画面を共用するように管理します。

ユーザー表示画面は以下の場合に、復元されます。

- 処理中のコマンドがオペレーターからの入力を必要とする場合
- 処理中のコマンドがユーザー表示画面を変更することになる場合

- USER DISPLAY が要求される場合

したがって、SEND コマンドの後に RECEIVE コマンドが続く場合、SEND コマンドによって送信された表示画面は、最初は SEND コマンドの処理時点で、次に RECEIVE コマンドの処理時点で、2 回現れます。SEND コマンドに応答する必要はありませんが、応答すると、インタープリターはその応答を保管しておき、RECEIVE コマンド用に画面を復元するときに再表示します。

インタープリターがユーザー表示画面を復元する時には、アラームを鳴らしたり、SEND コマンドを処理する時と同じようにキーボードに影響を及ぼしたりすることはありません。このことは EDF と同様です (188 ページの『単一画面モードにおける EDF の使用』を参照してください)。

共用ストレージ: LENGTH オプションを指定しない ENQ コマンド

通常、LENGTH オプションを指定せずに EXEC CICS ENQ コマンドを使用すると、リソースとして、ストレージ内の特定の場所 (アドレス) にデータ域が指定されます。複数のタスクがこのリソースでエンキューすることができますが、どのタスクもストレージ内の同じ場所を参照する必要があります。CECI は共用ストレージではなく独自の作業用ストレージを使用するため、この振る舞いをエミュレートすることはできません。

CECI で LENGTH オプションを指定せずに ENQ コマンドを実行すると、CICS は CECI タスクが所有しているストレージ内のアドレスでエンキューします。その他のタスクは、CECI かどうかに関係なく、この同じストレージでエンキューすることはできません。CECI は、変数のために共用ストレージを使用することをサポートしていません。

CECI タスク内で ENQ コマンドを実行するときにストレージ・アドレスを RESOURCE オプションとして指定して、LENGTH オプションを追加し、その後、別の CECI タスクまたは非 CECI タスクで LENGTH オプションを指定せずに同じストレージ・アドレスを指定しても、希望の振る舞いをエミュレートすることはできません。LENGTH オプションが指定されている場合、CICS はその場所ではなくリソースの値でエンキューします。したがって CICS は、LENGTH オプションが指定されているエンキューと指定されていないエンキューを別のエンキューとみなし、そのタスクは希望どおりには直列化されません。

LENGTH オプションが指定されている場合にはそのデータ域の場所 (CECI によって所有されているストレージかその他のストレージか) は問題にならないため、複数のタスクから発行された同じ ENQ コマンドに LENGTH オプションが指定されている場合は、そのエンキューは予期したとおりに動作します。

第 14 章 CICS アプリケーションでのデバッガーの使用準備

CICS は、バグの分離と修正、およびアプリケーションのテストを行うための、ワークステーション・ベース・デバッガーおよびホスト・ベース・デバッガーの使用をサポートします。CICS アプリケーションでデバッガーを使用する前に、以下の作業を行う必要があります。

始める前に

このタスクについて

1. ワークステーション・ベース・デバッガーとホスト・ベース・デバッガーのいずれかを選択する。 アプリケーション・プログラムをデバッグする場合は、デバッグ・ツールを使用してプログラムと対話します。例えば、ストレージの検査、ブレークポイントの設定、またはコードのステップスルーを行いたい場合があります。この対話が、デバッグ・セッション です。 CICS では、デバッグ・セッションを行う環境を選択することができます。

   ワークステーション・ベース
   ワークステーションでの デバッガー・クライアント は、デバッグ・タスクの実行に使用する、グラフィカル・ユーザー・インターフェースを提供します。デバッガー・クライアントは、CICS システムで実行されるデバッガー・サーバー と通信し、デバッグされているプログラムと対話します。

   詳しくは、 235 ページの『第 15 章 ワークステーションから CICS アプリケーションをデバッグする』を参照してください。

   ホスト・ベース
   CICS システムで実行中のデバッグ・ツールは、デバッグ・タスクの実行に使用する端末インターフェースを提供します。デバッグ・ツールは、実行時にアプリケーションと直接対話します。

   CICS は、ホスト・ベース・デバッグのためのデバッグ・ツールをサポートします。詳しくは、 237 ページの『第 16 章 CICS アプリケーションでのデバッグ・ツールの使用』を参照してください。

   アプリケーションによって、デバッグ要件は異なる場合があります (例えば、ホスト・ベース・デバッグ・セッションでは Java プログラムをデバッグできません)。CICS では、異なるユーザーが、同じ領域でワークステーション・ベース・デバッグとホスト・ベース・デバッグを同時に使用することができます。
2. デバッグ・ツールが使用するアプリケーション・プログラムを代行受信する (他のプログラムを代行受信しない) ことを確認する。 テスト・システムや開発システムにおいても、アプリケーション・プログラムの大部分は、ほとんどの場合正常に機能します。デバッグの実行中には、一度に 1 つのアプリケーションに集中したいことがあります。同時に、同僚が別のアプリケーションをデバッグしたがっていることも考えられます。そのため、CICS システムで大部分のプログラムを正常に実行させながら、使用しているデバッグ・セッションと対話するシ

© Copyright IBM Corp. 1989, 2009

ステム、および他のユーザーのデバッグ・セッションと対話するシステムでこれらのプログラムを指定する方法が必要となります。

デバッグ・プロファイルは、これらすべての実行を可能にします。デバッグ・プロファイルは、一緒にデバッグされる一連のアプリケーション・プログラムを指定します。プロファイルをアクティブにすると、プロファイルで定義されたプログラムは、ユーザーの指定したデバッグ・セッションを使用して、デバッガーの制御下で実行します。プロファイルを非アクティブにすると、デバッグ・プロファイルで参照されていないプログラムと同様に、プログラムは再び正常に実行します。デバッグ・プロファイルを使用すると、特定のプログラムのデバッグに使用するデバッグ・セッションの特性を定義することもできます。

詳しくは、『デバッグ・プロファイル』を参照してください。

3. デバッガーとの対話用にプログラムを準備する。 CICS は、さまざまな言語で作成されたアプリケーション・プログラムをサポートします。コンパイル済み言語プログラム (COBOL、PL/I、C、C++、および言語環境プログラムが使用可能なアセンブラー・サブルーチン) は、言語環境プログラムの制御下で実行されます。Java プログラムは、Java 仮想マシン (JVM) で実行されます。基本的に、プログラムには 2 つの異なるランタイム環境があるため、プログラムをデバッガーと対話させる方法は 2 つあります。
   - コンパイル済み言語プログラムでは、デバッガーと対話させるプログラムをコンパイルする時期を決定し、適切なコンパイラー・オプションを指定する必要があります。詳細については、コンパイラーの資料を参照してください。
   - Java プログラムでは、実行時にデバッガーと対話させるプログラムを決定し、適切な JVM オプションを指定することができます。詳しくは、「*Java Applications in CICS*」を参照してください。
4. CICS システムがデバッグ環境をサポートするようにセットアップする。 CICS システムにデバッグ・プロファイルがある場合は、すべてのプロファイルが非アクティブになっていても、プログラムの開始時にはオーバーヘッドがあります。このオーバーヘッドは、たとえ小さいものであっても、ハイパフォーマンス・システムでは多くの場合許容されません。一般に、このようなシステムで、アプリケーションのデバッグを行うことはありません。そのため、デバッグ・プロファイルの使用はオプションであり、このプロファイルをユーザーが使用する場合には、システム・プログラマーが CICS を構成する必要があります。

例

次のタスク

デバッグ・プロファイル

デバッグ・プロファイルは、一緒にデバッグされる 1 つ以上のアプリケーション・プログラムのまとまりを指定します。

例:

- システム CICS1 で実行中のプログラム PYRL01 のすべてのインスタンス
- 『setBankAccount』 で始まる名前を持つすべての Java クラス

- ユーザー APPDEV02 が実行する、『'PYRL'』 で始まる名前を持つすべてのプログラム

CICS は、デバッグ・プロファイルで以下の情報を使用して、デバッガーの制御下でプログラムのインスタンスを実行する必要があるかどうかを決定します。パラメーターで、以下のものを指定します。
- プログラムを実行中のトランザクション
- トランザクションと関連付けられた端末。端末 ID または VTAM ネット名を指定できます。
- プログラムの名前
- COBOL プログラムでは、コンパイル単位の名前 (プログラムまたはクラス名)
- Java オブジェクトおよびステートレス CORBA オブジェクトでは、クラス名
- Enterprise Beans では Bean 名
- Enterprise Beans およびステートレス CORBA オブジェクトでは、メソッド名
- サインオン・ユーザーのユーザー ID
- トランザクションを実行中の CICS 領域のアプリケーション ID

パラメーターの多くは汎用であり、これを使用すると、同じ文字 (例えば、TRN0、TRN1、TRN2、TRNA、TRNB など) で始まる一連の値を指定することができます。

デバッグ・プロファイルには、以下の追加情報が含まれます。

状況 プロファイルの状況: *アクティブ* または*非アクティブ*:
- アクティブになっているプロファイルは、デバッグが必要な領域でプログラムが開始されるたびに検査されます。

  注: アクティブ状態のプロファイルを変更すると、その変更は即時に反映されます。プログラムの次回の開始時に、プログラムをデバッガーの制御下で実行するかどうかを決定するために、変更後のパラメーターが使用されます。
- 非アクティブ状態のプロファイルは、プログラムの開始時に無視されます。

デバッグのディスプレイ装置の設定
デバッグのディスプレイ装置の設定では、デバッガーとの対話方法を指定します。
- Java プログラムでは、ワークステーションでデバッグ・ツールを使用できます。
- コンパイル済み言語プログラムでは、以下を使用できます。
  3270 端末
  ワークステーション上のデバッグ・ツール

JVM プロファイル名
Java プログラムの場合に限り、プログラムのデバッグ時に使用される JVM プロファイルを指定できます。

デバッグ・ツールおよび言語環境プログラムのオプション
コンパイル済み言語プログラムの場合に限り、プログラムのデバッグ時にデバッグ・ツールおよび言語環境プログラムに渡されるオプションを指定できます。

以下のような種類のプログラムで、デバッグ・プロファイルを作成できます。

コンパイル済み言語プログラム
Java アプリケーション・プログラム
Enterprise Beans
ステートレス CORBA オブジェクト

プロファイルに保管される情報は、プログラムの種類に応じて異なります。

プロファイルは、複数の CICS 領域で共用可能な CICS ファイルに保管されます。複数の CICS 領域によって共有される 1 つのプロファイルは、すべての領域においてアクティブまたは非アクティブになります。ある領域でアクティブに、別の領域では非アクティブにすることはできません。

CICS は、システムでデバッグ・プロファイルを使用するようにセットアップされている場合にオプションで生成される、一連のサンプル・プロファイルを提供します。独自のプロファイルを作成する際の開始点として、これらのプロファイルを使用することができます。

デバッグ・プロファイルを使用してデバッグ対象のプログラムを選択する

デバッグ対象のプログラムを選択するには、1 つ以上のデバッグ・プロファイルを作成する必要があります。各プロファイルは、プログラムのインスタンスをデバッガーの制御下で実行するかどうかを決定する際に CICS が使用する、複数のパラメーターを指定します。

プロファイルは、アクティブであっても非アクティブであってもかまいません。アクティブなプロファイルの 1 つがプログラム・インスタンスと一致する場合、プログラムはデバッガーの制御下で実行されます。非アクティブなプロファイルは、CICS がプログラムを開始する際に検査されません。プロファイルは、作成時には非アクティブです。

表 10 には、デバッグ・プロファイルのパラメーターを使用して、コンパイル済み言語プログラムのプログラム・インスタンスを選択する方法の例を示しています。231 ページの表 11 には、デバッグ・プロファイルのパラメーターを使用して、Java プログラムのプログラム・インスタンスを選択する方法を示しています。

表 10. コンパイル済み言語プログラムのデバッグ・プロファイル・パラメーターの例

デバッグ・プロファイル	トランザクション	端末	プログラム	ユーザー	アプリケーション ID
プロファイル 1	PRLA	T001	PYRL01	TESTER5	CICSTST2
プロファイル 2	PRLA	*	PYRL02	*	*
プロファイル 3	PRL*	*	*	*	CICSTST3

表 11. Java プログラムのデバッグ・プロファイルの例

デバッグ・プロファイル	トランザクション	Bean	メソッド	ユーザー	アプリケーション ID
プロファイル 4	PRLA	NewEmployee	setBasicSalary	TESTER5	CICSTST2

これは、各プロファイルが、デバッガーの制御下で実行されるプログラムを制御する方法です。

プロファイル 1

この例では、表内のすべてのパラメーターが明示的に指定されています。プログラム PYRL01 は、これらすべての条件が満たされた場合にのみ、デバッガーの制御下で実行されます。

- トランザクションが PRLA である
- トランザクションが、端末 T001 からの端末入力によって開始されている
- トランザクションが、ユーザー TESTER5 によって実行中である
- トランザクションが、領域 CICSTST2 で実行中である

プロファイル 2

この例では、表内のパラメーターの一部が*汎用*パラメーターであり、* として指定されます。このタイプの汎用パラメーターは、すべての値にマッチングするします。このプロファイルは、トランザクション PRLA の下で実行されるプログラム PYRL02 のすべてのインスタンスがデバッガーの制御下にあることを指定します。

プロファイル 3

この例には、別の種類の汎用パラメーターが含まれています。PRL* は、文字「PRL」で始まるすべての値にマッチングするします。このプロファイルは、領域 CICSTST3 で、文字「PRL」で始まる ID を持つトランザクションの下で実行されるすべてのプログラムが、デバッガーの制御下にあることを指定します。

プロファイル 4

メソッド setBasicSalary は、以下の条件がすべてが満たされる場合にのみ、デバッガーの制御下で実行されます。

- トランザクションが PRLA である
- メソッドが Bean NewEmployee のメソッドである
- トランザクションが、ユーザー TESTER5 によって実行中である
- トランザクションが、領域 CICSTST2 で実行中である

プログラムが予期せずにデバッガーの制御下で開始されないようにするため、デバッグ・プロファイルで指定するパラメーターは慎重に選択する必要があります。

- 可能であれば、パラメーターの全部または大部分の値を指定して、特定の環境下の特定プログラムにデバッグを制限します。できるだけ、汎用値ではなく特定の値を使用します。
- 可能であれば、各デバッグ・プロファイルで、ユーザー ID およびアプリケーション ID を明示的に指定します。

- 実動領域でプログラムをデバッグすることはお勧めできませんが、必要な場合もあります。その場合には、すべてのパラメーターが明示的に指定されているデバッグ・プロファイルを使用してください。
- デバッグ・プロファイルは、使用する必要があるときにのみアクティブにし、使用後は即時に非アクティブにします。

デバッグ・プロファイルでの汎用パラメーターの使用

デバッグ・プロファイルでは、多くのパラメーターで汎用値を使用できます。汎用パラメーターを指定するには、*ワイルドカード文字* としてアスタリスク (*) を使用します。ワイルドカード文字は、単独で使用することも、パラメーターの終わりに使用することもできます。パラメーターを空白にすることは、アスタリスクを指定したことに相当します。

このタスクについて

例:

* は、使用可能なすべての値にマッチングする
TR* は、TR、TRA、TRAA、および TRAQ にマッチングする
TRA* は、TRA、TRAA、および TRAQ にマッチングするが、TR はマッチングしない

ワイルドカードを使用すると、始動時のプログラムが、複数のアクティブなプロファイルにマッチングする可能性があります。このような場合に、CICS は、以下の原則に基づいて、最もマッチングするプロファイルを選択します。

- ワイルドカードが使用されていても、使用されていなくても、すべてのパラメーターが正確に一致する必要があります。
- ワイルドカードを含まないプロファイルが最もマッチングすると見なされます。
- その次に、* を含むプロファイルが検討されます。このグループでは、* 文字の数が最も少なく、明示的に指定された文字が最も多く含まれているプロファイルが最もマッチングすると見なされます。

トランザクション TRAA を例に考えてみます。

- TRAA は、最もマッチングする (すべての文字が一致する) と見なされる
- TRA* は、TR* より適したマッチングである

デバッグ・プロファイルでは、ワイルドカードを複雑な方法で使用しないことをお勧めします。これは、数多くのプロファイルの中で、指定されたプログラム・インスタンスに最もマッチングするものが、必ずしも明白にならないことがあるからです。ただし、複雑な方法で使用する必要がある場合は、233 ページの図 58 の情報を利用すると、複数のプロファイルのうち最もマッチングするものを正確に把握することができます。

フィールドごとに、以下を順次行います。

1. フィールドごとに文字の数を数える (* は除外するが、末尾ブランクを含む) (*C*)
2. * 文字の数を数える (*A*)
3. フィールドの長さを決定する (*L*)
4. *M* を *C -(L * A)* として計算する。*M* が負になる場合がある点に注意してください。

各プロファイルで順番に、すべてのフィールドの *M* の値を合計する (*R*)

R が最大値であるプロファイルが最もマッチングすると見なされます。複数のマッチング・プロファイルで *R* が同じ最大値になる場合は、CICS が、プロファイルの作成された順番に基づいて 1 つを選択します。

図 58. デバッグ・プロファイルのマッチング・アルゴリズム

第 15 章 ワークステーションから CICS アプリケーションをデバッグする

ワークステーションで実行されるデバッグ・ツールを使用して、CICS アプリケーションをデバッグすることができます。

この環境では、デバッグ・ツールに対する 2 つのコンポーネントがあります。

- ワークステーションで実行されるデバッガー・クライアント。これは、アプリケーション・プログラムと対話するデバッガー・クライアントが提供する、グラフィカル・ユーザー・インターフェース (GUI) を介して入手できます。デバッガー・クライアントを使用すると、ブレークポイントの設定、プログラムのステップスルー、およびプログラムが使用する変数の検査などを行えます。
- アプリケーション・プログラムと同じシステムで稼働し、デバッガー・クライアントと通信するデバッガー・サーバー。

ワークステーションでデバッガー・クライアントを使用すると、以下の種類の CICS アプリケーションをデバッグすることができます。

- コンパイル済み言語 (COBOL、PL/I、C、C++) で作成されたアプリケーション
- 言語環境プログラムが使用可能なアセンブラー・サブルーチン
- JVM で実行中の Java アプリケーション
- コンパイル済み言語プログラムと Java プログラムの組み合わせを使用するアプリケーション

ワークステーションでデバッガー・クライアントを使用して、PLT プログラムをデバッグすることはできません。

デバッガー・クライアントとして、以下を使用できます。

WebSphere® Studio Enterprise Developer
WebSphere Studio Application Developer

コンパイル済み言語および言語環境プログラムが使用可能なアセンブラー・サブルーチンでは、デバッガー・サーバーとして以下の製品を使用できます。

- デバッグ・ツール

Java プログラムでは、デバッガー・サーバーは、デバッグ・モードで稼働中の Java 仮想マシン (JVM) です。

ワークステーションからアプリケーションのデバッグを準備する

始める前に

ワークステーションを使用して CICS アプリケーションをデバッグする前に、システム・プログラマーがデバッグ用の CICS 領域を準備する必要があります。

このタスクについて

以下のタスクを完了する必要があります。

© Copyright IBM Corp. 1989, 2009

1. ワークステーションに適切なデバッガー・クライアントをインストールする。以下の製品をデバッガー・クライアントとして使用できます。

   WebSphere Studio Enterprise Developer
   WebSphere Studio Application Developer

   これらの製品の資料には、インストールおよび使用上の必要な情報が含まれています。
2. 1 つ以上の デバッグ・プロファイル を作成する。 各デバッグ・プロファイルは、デバッガーの制御下で実行されるプログラムを指定します。

   注: デバッグ・プロファイルは、JVM プロファイルと同じものではありません。Java アプリケーションをデバッグするには、両方のプロファイルが必要です。
3. COBOL、PL/I、C または C++ で作成されたプログラム、または言語環境プログラムが使用可能なアセンブラー・サブルーチンをデバッグする場合は、デバッグ・セッションを実行する方法を検討し、適切なオプションを使用してプログラムをコンパイルする。詳しくは、「*Debug Tool for z/OS User's Guide*」を参照してください。
4. Java プログラムをデバッグする場合は、デバッグが使用可能になっている Java 仮想マシン (JVM) でプログラムを実行する必要がある。 そのためには、以下の作業を行います。
   a. JVM のデバッグを使用可能にするパラメーターで JVM プロファイルを作成する。詳しくは、「*Java Applications in CICS*」のCICS JVM で実行されているアプリケーションのデバッグを参照してください。
   b. Java プログラムのデバッグ・プロファイルを作成するときに、JVM プロファイルを指定する。JVM プロファイルを指定しない場合は、JVM は PROGRAM 定義で指定されたプロファイルを使用します。
5. ワークステーションでデバッガー・クライアントを始動する。
6. WebSphere Studio をデバッガーとして使用している場合は、プログラムに少なくとも 1 つのブレークポイントを設定する。
7. デバッグ対象のプログラム・インスタンスを定義するデバッグ・プロファイルをアクティブにする。コンパイル済み言語プログラムでプロファイルをアクティブにする場合は、プログラムの実行時に始動するデバッグ・セッションの属性を指定する、デバッグ・オプションを定義する必要があります。

タスクの結果

これらのステップをすべて完了したら、最後のステップで選択したプログラムは、デバッガーの制御下で実行されるようになります。

第 16 章 CICS アプリケーションでのデバッグ・ツールの使用

デバッグ・ツールを使用すると、プログラムのテスト、アプリケーション・プログラムの実行の検査、モニター、および制御を行うことができます。

デバッグ・ツールについて詳しくは、「*Debug Tool for z/OS User's Guide*」を参照してください。

デバッグ・ツールについて

デバッグ・ツールを使用すると、プログラムのテスト、CICS アプリケーション・プログラムの実行の検査、モニター、および制御を行うことができます。

デバッグ・ツールを使用すると、以下の種類の CICS アプリケーションをデバッグすることができます。

- コンパイル済み言語 (COBOL、PL/I、C、C++) で作成されたアプリケーション
- 言語環境プログラムが使用可能なアセンブラー・サブルーチン
- コンパイル済み言語プログラムと Java プログラムの組み合わせを使用するアプリケーション。デバッグ・ツールは、これらのアプリケーションの Java である部分をデバッグしません。

デバッグ・ツールを使用して PLT プログラムをデバッグすることはできません。

デバッグ・ツールは、以下の 4 つの方法で使用することができます。

単一端末モード
デバッグ・ツールは、アプリケーションと同じ端末に画面を表示します。

二重端末モード
デバッグ・ツールは、アプリケーションが使用する端末とは異なる端末に画面を表示します。

バッチ・モード
デバッグ・ツールには端末がありませんが、コマンド・ファイルを使用して入力し、出力をログに書き込みます。

リモート・デバッグ・モード
デバッグ・ツールは、デバッガー・クライアントを処理して、ワークステーションに結果を表示します。

デバッグ・ツールについて詳しくは、「*Debug Tool for z/OS User's Guide*」を参照してください。

注: 単一端末モードまたは二重端末モードでデバッグ・ツールを使用する場合、デバッグ・ツールが使用する端末は、アプリケーションを実行中の領域内のローカル端末でなければなりません。アプリケーション専有領域のデバッグ・ツールと対話するために、端末専有領域で端末を使用することはできません。

© Copyright IBM Corp. 1989, 2009

デバッグ・ツールによるデバッグ・アプリケーションの準備

始める前に

デバッグ・ツールを使用して CICS アプリケーションをデバッグする前に、システム・プログラマーがデバッグ用の CICS 領域を準備する必要があります。詳しくは、「*CICS アプリケーション・プログラミング・ガイド*」を参照してください。

このタスクについて

その後、ユーザーが以下のタスクを完了します。

1. デバッグ・セッションを実行する方法を検討し、適切なオプションを使用してプログラムをコンパイルする。 詳しくは、「*Debug Tool for z/OS User's Guide*」を参照してください。
2. 1 つ以上のデバッグ・プロファイルを作成する。各デバッグ・プロファイルは、デバッガーの制御下で実行されるプログラムを指定します。
3. デバッグ対象のプログラム・インスタンスを定義するデバッグ・プロファイルをアクティブにする。プロファイルをアクティブにする場合は、デバッガーとの対話に使用するディスプレイ装置を指定する必要があります。

タスクの結果

これらのステップをすべて完了したら、最後のステップで選択したプログラムは、デバッグ・ツールの制御下で実行されるようになります。

第 4 部 CICS アプリケーションのプログラミング手法

CICS のプログラムを作成するために知っている必要がある概念と技法。

© Copyright IBM Corp. 1989, 2009

第 17 章 アプリケーション設計

これらのトピックでは、CICS アプリケーションの設計に役立つ基本概念について説明します。パフォーマンスと効率を改良できる可能性がある設計上の変更点について説明しますが、効率化のプログラミングに関する詳しい説明は、271 ページの『第 18 章 パフォーマンスの設計』を参照してください。

CICS でインプリメントされるプログラミング・モデルは、3270 用に設計されたプログラミング・モデルから継承されたもので、会話型の端末向けアプリケーションの特性の多くを備えています。プログラミング・モデルには、基本的に次の 3 つのスタイルがあります。

- 端末開始型、すなわち会話型モデル
- 分散プログラム・リンク (DPL)、または RPC モデル
- START、すなわちキューイング・モデル

開始後は、アプリケーションは通常これらのモデルや、このモデルを継続および分散するその他の方法 (例えば、疑似会話、RETURN IMMEDIATE、あるいは DTP など) を使用します。これらのモデル間の主な違いは、状態 (例えばセキュリティー) を保守する方法です。このため、状態がアプリケーション設計に欠かせない要素になります。別のアプリケーション・モデルに変換しようとする際には、これが最大の問題になります。

疑似会話型モデルは、ほとんどの場合、端末開始トランザクションに関連付けられています。このモデルは、会話型モデルの効率的なインプリメンテーションとして開発されたものです。HTTP などの 1-in および 1-out プロトコルの使用が増すにつれ、DPL または RPC モデルに疑似会話型特性を追加する必要が出てきます。

疑似会話型設計と会話型設計

会話型トランザクションの場合には、ユーザーの各応答の処理に費やされる時間の長さは、入力を待っている時間の長さと比較して、著しく短いのです。会話型トランザクションは、トランザクションとユーザーが会話に入るように、端末からの複数の入力をともないます。**非会話型**トランザクションは、入力を 1 つしか持っていません (この入力がトランザクションを呼び出します)。このトランザクションが入力、端末への応答、および終了を行います。

プロセッサーの速度は、データ・セットへのアクセスを考慮に入れても、端末の伝送時間よりかなり高速です。これは、ユーザーの応答時間よりかなり高速であることを意味します。ユーザーが考えながら入力する必要がある場合や、たくさんの文字を入力する必要がある場合には、なおさらです。したがって、会話型トランザクションは、非会話型トランザクションの何倍もの長時間、ストレージおよびその他のリソースを占有します。

疑似会話型トランザクションのシーケンスには、一連の非会話型トランザクションが含まれ、これがユーザーには、複数の入力表示画面をともなう単一の会話型トラ

© Copyright IBM Corp. 1989, 2009

ンザクションのように見えます。このシーケンスの中の各トランザクションが、1 つの入力の処理、応答の返信、および終了を行います。

疑似会話型トランザクションは、終了前に、次に同じ端末から開始されるトランザクションが使用するデータを、そのトランザクションが到着したときにいつでも使用できるように、先送りしておくことができます。疑似会話型トランザクションは、RETURN コマンドの TRANSID オプションを使用して、次のトランザクションを指定することができます。ただし、その装置で別のトランザクションが開始された場合には、設計した疑似会話型チェーンが、そのトランザクションによって中断されることがある点に注意する必要があります。ただし、RETURN コマンドの IMMEDIATE オプションを指定している場合はこの限りではありません。この場合は、この端末用のキューに入っている他のトランザクションに関係なく、TRANSID コマンドで指定したトランザクションが付加されます。

応答が書き込まれた時点から、ユーザーが次の入力を送り、CICS がそれに応答する次のトランザクションを開始するまでの間は、端末のトランザクションは存在しません。通常、入力と入力の間にプログラム内に格納される情報は、COMMAREA またはこの目的のために CICS が提供する他の機能の 1 つを使用して、シーケンス内のあるトランザクションから次のトランザクションに渡されます (詳しくは、281 ページの『第 19 章 トランザクション間のデータの共用』を参照してください)。

会話型プログラミングと疑似会話型プログラミングの選定に当たっては、主に次の 2 つの問題点について考慮してください。

- ストレージおよびプロセッサー使用などの**競合**リソースに対するトランザクションの影響。トランザクションを構成する制御ブロック、データ域、およびプログラムにはストレージが必要で、プロセッサーはタスクの開始、処理、および終了を行うために必要です。会話型プログラムは、同等の疑似会話型シーケンスを構成するトランザクションの合計と比較しても持続時間が長いので、ストレージに ***非常に*** 大きな影響を与えます。しかし、トランザクションは入力ごとに 1 つずつ開始されるのではなく、1 つしか開始されないので、プロセッサー・オーバーヘッドは少なくなります。
- リカバリー可能データ・セットのレコード、リカバリー可能一時データ・キュー、エンキュー項目などの**排他使用**リソースに対する影響。会話型プログラムは、非会話型トランザクションの対応するシーケンスよりはるかに長い時間、これらのリソースを保留します。この観点からすると、迅速な応答には疑似会話型トランザクションの方が適していますが、リカバリーおよび保全性を重視する場合は、会話型トランザクションを優先的に使用することになります。

要約すると、会話型タスクは作成しやすいにしても、パフォーマンス (特に仮想記憶域の必要量) や、その会話型タスクを含む CICS システムの操作容易性全体への効果の両面で、重大な不利益があります。現在、プロセッサーは過去と比較して、一段と多くの実記憶域をもち、一段と強力になって大型化し、少量の会話型タスクは苦にならなくなっていますが、会話型アプリケーションを使用する場合には、急速に仮想記憶域の制約に達する可能性があります。アプリケーション・プログラムを境界の上で実行する場合は、おそらく、仮想記憶域の制約に達する前に ENQ 問題にぶつかります。

CICS は、作業単位 (UOW) で行われるリカバリー可能リソース (データ・セット、一時データ、および一時記憶域など) への変更が完全に行われるか、あるいはまっ

たく行われないようにします。UOW は、トランザクションが SYNCPOINT コマンドを発行しない限り、トランザクションと同等です。このコマンドを発行すると、同期点と同期点の間 UOW が持続します。同期点および UOW の詳細については、「*CICS Recovery and Restart Guide*」のリカバリー機能および再始動機能を参照してください。

トランザクションがリカバリー可能リソースを変更した場合には、CICS は、元のトランザクションが完了するまで、そのリソースを変更しようとする他のどのトランザクションに対しても、そのリソースを使用不能にします。会話型トランザクションの場合には、このような未確定のリソースは比較的長い時間にわたり、他の端末で使用不能になる可能性があります。

例えば、あるユーザーがリカバリー可能データ・セットの特定のレコードを更新しようとしており、最初の更新が終了する前に別のユーザーがそのレコードを更新しようとした場合には、2 番目のユーザーのトランザクションが中断されます。この点は長所であり、短所でもあります。最初のユーザーがレコードを変更している間、2 番目のユーザーにはそのレコードへの更新を始めさせたくないものです。一方のユーザーは古くなろうとしているレコードのバージョンを処理し、これらの変更によって他のユーザーの変更内容が消去されてしまうためです。一方、トランザクションがレコードを更新するために読み取ろうとしたときに、2 番目のユーザーを長時間、説明が何もないままで待たせたくないこともあります。

しかし、疑似会話型トランザクションを使用した場合には、リソースは非常に短い間 (すなわち、コンポーネントである短いトランザクションの間) 使用不能になるだけです。しかし、UOW を複数のトランザクションにまたがるように拡張することはできないので、これらのトランザクションの 1 つだけですべてのリカバリー可能リソースを更新することができない限り、リカバリーは不可能です。そこで、リカバリー可能リソースへの更新をこの方法で切り分けることができない場合には、会話型トランザクションを使用する必要があります。

上記の例は、疑似会話型トランザクションについてのさらなる問題を提起しています。すべての更新をトランザクションのシーケンスの最後に制限することができても、最初のユーザーが変更内容をまだ入力している間に、2 番目のユーザーがそれと同じレコードに対する更新トランザクションを始めるのを、防ぐ手だてはありません。これは、保全性を確実にするためには、追加のアプリケーション論理が必要であることを意味します。エンキューするためになんらかの形式を使用するか、あるいは更新を適用する前に、トランザクションによってレコードの元のバージョンを現行バージョンと比較することができます。

端末の割り込み機能

会話型タスクが実行中の場合には、CICS は、他のタスクがそのタスクの端末にメッセージを送信できないようにします。この点は長所であり、短所でもあります。長所は、予期しないメッセージ (例えば、ブロードキャスト) がユーザーとマシンのダイアログに割り込んだり、さらに悪化し、形式設定された表示画面を壊したりすることがない点です。短所は、重要な情報、例えば、制御オペレーターが 10 分後に CICS をシャットダウンしようとしていることなどが、エンド・ユーザーに通知できないことです。さらに重要なことは、エンド・ユーザーがうっかり会話を終了させないままにすると、実際に、正常な CICS シャットダウンを妨げたり遅らせたりする場合があることです。

疑似会話型アプリケーションでは、一対の会話の間にメッセージを送ることができます。これは、シャットダウン警告のような通知が送達できることを意味します。このメッセージが表示画面の内容を乱すことがあり、TRANSID オプションを使った RETURN コマンドによって制御されたトランザクションのシーケンスを妨害することがあります。しかし、これは、IMMEDIATE オプションを使用するか、対話のリンクしたシーケンスの中間で端末を強制的に NOATI 状況にすることによって、避けることができます。

タスクの開始方法

作業が CICS で開始される (つまり、タスクが開始される) 方法は、以下の 2 つのうちいずれかです。

1. 非送信請求入力から
2. 自動タスク開始 (ATI) によって

自動タスク開始が起こるのは以下の場合です。

- 既存のタスクが別のタスクを作成するように CICS に要求する場合。START コマンド、RETURN コマンドの IMMEDIATE オプション (592 ページの『RETURN IMMEDIATE』で説明)、および SEND PAGE コマンド (834 ページの『SEND PAGE コマンド』で説明) はすべて、これを行います。
- CICS が一時データ・キューを処理するタスクを作成する場合 (675 ページの『自動トランザクション開始 (ATI)』を参照)。
- CICS が BMS ROUTE 要求で出されるメッセージを送達するタスクを作成する場合 (853 ページの『第 68 章 メッセージ・ルーティング』を参照)。 CICS 提供のトランザクション CMSG の後で行われる CSPG タスクがこの例です。 CMSG は、宛先リスト中の各ターゲット端末用に CSPG トランザクションを作成するために、ROUTE コマンドを使用します。

しかし、タスクを開始する基本メカニズムは非送信請求入力です。ユーザーが、既存タスクの**プリンシパル装置** ではない端末からの入力を伝送した場合には、CICS はそれを処理するタスクを作成します。入力を送信した端末は新規タスクのプリンシパル装置になります。

プリンシパル装置

CICS によって、タスクはただ 1 つの端末、すなわちそのプリンシパル装置と直接通信することができます。CICS がプリンシパル装置を割り当てるのは、タスクを開始する時点で、タスクはプリンシパル装置をその期間「所有」します。他のタスクは、所有しているタスクが終了するまでその端末を使用することはできません。タスクがそのプリンシパル装置とは違う端末との通信を必要とする場合には、その端末をプリンシパル装置として持つ別のタスクを作成して間接的に通信しなければなりません。この要件は印刷処理との接続では最も一般的に起こり、このようなタスクの作成方法については、 752 ページの『CICS プリンターの使用』 で説明しています。

注:

1. 端末の宛先が TCAM 制御下にある場合、SEND コマンドのプリンシパル装置ではない端末の宛先を指定することができますが、これは、明らかに例外規則です。これが可能であるのは、TCAM 端末との通信が常に

キューイングされるためです。このように、タスクは宛先端末に対し直接書き込みを行いませんが、TCAM によって続いてその端末に送達されるキューに書き込みをします (615 ページの『第 41 章 TCAM の使用』を参照してください)。853 ページの『第 68 章 メッセージ・ルーティング』で説明されている BMS ルーティングは、キューによる他の端末へのもう 1 つの間接的アクセス形式です。

2. CICS Transaction Server for z/OS バージョン 4 リリース 1 では、ローカル TCAM 端末はサポートされません。サポートされている TCAM 端末は、TCAM の DCB インターフェース (ACB ではない) によって CICS TS 3.1 以前の端末専有領域に接続されているリモート端末のみです。

他のシステムからの非送信請求入力は同じ方法で処理されます。CICS はその入力を処理するタスクを作成し、入力が到着した会話をプリンシパル装置として割り当てます (したがって、別のシステムとの会話はプリンシパル装置または代替装置のいずれか一方となることがあります。ある CICS 領域のタスクが別の CICS 領域との会話を開始した場合には、この会話は開始タスクの代替装置ですが、受信システムによって作成されるパートナー・タスクのプリンシパル装置です。これとは対照的に、端末は常にプリンシパル装置です)。

代替装置

タスクはただ 1 つの端末と直接通信することはできますが、1 つ以上のリモート・システムとの通信を確立することもできます。これは、CICS に、そのシステムとの会話を**代替装置**として割り当てるように要求することによって行います。タスクは、プリンシパル装置を所有するのと同じ方法で代替装置を「所有」します。所有権は、割り当て時点からタスクが終了するまで、あるいはタスクがその装置を解放するまで持続します。

すべてのタスクがプリンシパル装置を持っているわけではありません。非送信請求入力からのタスクは常に定義によって実行されますが、自動タスク開始からのタスクはプリンシパル装置を必要とすることも必要としないこともあります。必要とする場合には、CICS は、要求した装置がそのタスクに割り当てるために利用可能になるまで、タスクの開始を待機します。

どのトランザクションか?

非送信請求入力を受信すると、CICS はその処理方法をどのように決めるのか? すなわち、その入力を処理するために作成されるタスクはどのトランザクションを実行するのか? 簡潔に言うと、その答えは、同一のプリンシパル装置を持つ直前のタスクが、最後に戻ったときに、通常 CICS に対して、TRANSID オプションによって、終了直後に次にどのトランザクションを実行するかを伝えるということになります。疑似会話型トランザクションのシーケンスの場合はほとんど常にこの方式で、メニュー方式アプリケーションの場合も通常は同様です。それに失敗し、開始されるシーケンスを入手した場合には、CICS は入力の先頭から数文字をトランザクション・コードとして解釈します。しかし、ことはそれより複雑で、正確には以下のように処理されます。ステップ番号はテストを行う順序を示します。
246 ページの図 59 (このロジックの図) を参照してください。

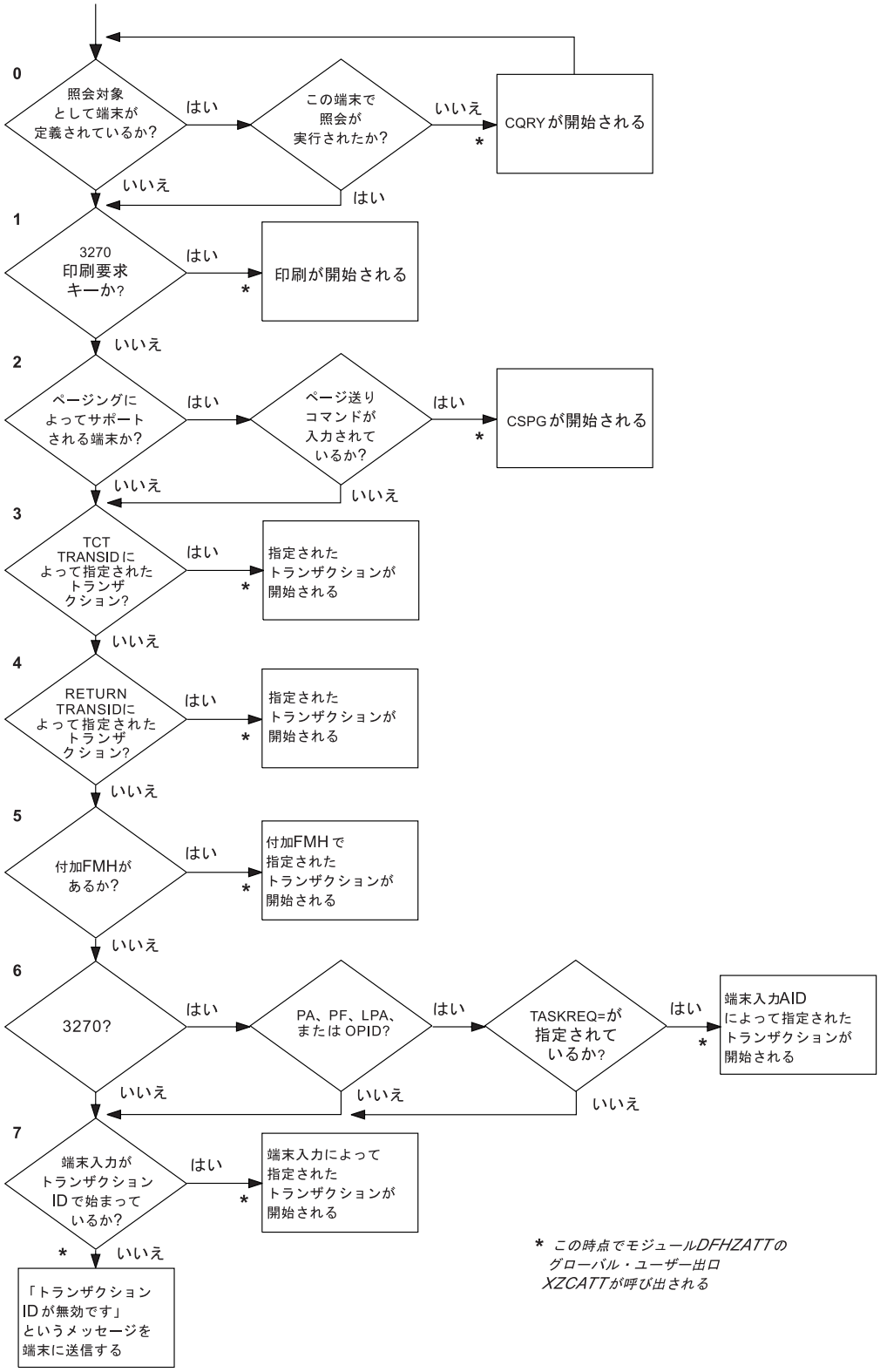


図 59. 実行するトランザクションの判別

0. 端末からの初期入力で、CICS はその入力を処理するタスクの作成前に準備タスクをスケジュールする場合があります。このタスクは CICS 提供の

「照会」トランザクション CQRY を実行します。これにより、端末はハードウェア特性 (拡張属性、文字セットなど) の一部をエンコードした記述を伝送します。

CQRY によって、システム・プログラマーは、端末定義からこれらの詳細を省略することによって、端末ネットワークの保守を単純化することができます。これが起こるのは、端末定義をそのように指定し、入力を処理するためにどのトランザクションを使用するかの後続の判別に何の影響もない場合だけです。

1. 端末が 3270 で、入力が「印刷要求キー」の場合には、画面の内容を印刷する CICS 提供のトランザクション CSPP が開始されます。この機能の詳細については、758 ページの『CICS 印刷キー』を参照してください。この目的の場合には、「3270 論理装置」または 3270 データ・ストリームを受け入れる他のすべての装置は、3270 としてカウントします。

2. フル BMS サポートが存在し、端末のタイプが BMS 端末ページ送りによってサポートされ、さらに入力がページ送りコマンドの場合には、その要求を処理する CICS 提供のトランザクション CSPG が開始されます。 BMS サポート・レベルについては、775 ページの『BMS サポート・レベル』で説明しています。同じセクションに BMS がサポートする端末がリストされています。システム初期設定テーブルの PGRET、SKRxxxx、PGCHAIN、PGCOPY、および PGPURGE オプションは、ページ送りコマンドを定義します。ページ送りはフル BMS を必要とするので、CICS システムがそれより低いレベルしか含んでいない場合には、このステップはスキップされます。

3. 端末定義が、特定のトランザクションを使用して、その端末からのすべての非送信請求入力を処理する必要があることを指示している場合には、指示されたトランザクションが実行されます (存在する場合には、この情報は TERMINAL 定義の TRANSACTION 属性に表示されます)。

4. 端末での直前のタスクで、タスクを終了する RETURN コマンドの TRANSID オプションが指定された場合には、名前を指定されたトランザクションが実行されます。

5. 入力データに付加機能管理ヘッダーがある場合には、ヘッダー内の付加機能名が 4 文字の CICS トランザクション ID に変換され、そのトランザクションが実行されます。

6. 端末が 3270 で、アテンション ID がトランザクションとして定義されている場合には、そのトランザクションが実行されます。 636 ページの『アテンション・キー』 に、アテンション ID についての説明があります。これは、対応した TRANSACTION 定義の TASKREQ 属性を持つトランザクション ID として定義されます。

7. 先行するすべてのテストが失敗した場合には、入力のイニシアル文字を使用して実行するトランザクションを識別します。使用される文字は、データ・ストリーム中のすべての制御情報の後で、最初のフィールド分離記号の前の先頭の文字 (最大 4 文字まで)、または、その次の 3270 制御文字 (X'00' から X'3F' まで) です。フィールド分離記号は、システム初期設定テーブルの FLDSEP オプションで定義されます (デフォルトはブランクです)。

入力に、このような文字がない場合 (例えば、CLEAR キーを使用した場合)、あるいは入力と一致するトランザクション定義がない場合には、CICS はどのトランザクションを実行するかを判別することができず、「トランザクション識別名が正しくない」というメッセージを端末に送信します。

注: どのトランザクションを実行するかを決定するためのこの論理が適用されるのは、非送信請求入力を処理するために開始されるタスクに対してだけです。自動トランザクション開始の場合には、トランザクションは常に既知です。 START コマンドまたは RETURN IMMEDIATE コマンドを使用してタスクを作成するときに、TRANSID オプションにトランザクションを指定します。同様に、どのトランザクションを使用して一時データ・キューを処理するかをキュー定義に指定します。メッセージをルーティングするために作成されるタスクは、常に CICS 提供のトランザクション CSPG を実行します。

ビジネス・ロジックと表示ロジックの分離

一般に、アプリケーションを、再使用が可能なビジネス・コードを含む部分と、クライアントに対する表示を受け持つ部分に分割するのは、いい方法です。この方法をとると、各部分を個別に最適化してパフォーマンスを向上させることができ、ビジネス・ロジックをさまざまな表示形式で再使用することができます。

ビジネス・ロジックと表示ロジックを分離する場合、以下の点を考慮する必要があります。

- アプリケーションのこの 2 つの部分に類縁性がないようにする。
- DPL 制限のある API に注意する。詳しくは、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。
- 隠れた表示依存関係 (EIBTRMID の使用など) に注意する。

249 ページの図 60は、エンド・ユーザーからのデータを受け入れ、ファイル内のレコードを更新して、エンド・ユーザーに応答を返すという、単純な CICS アプリケーションを示したものです。このプログラムを実行するトランザクションは、疑似会話では 2 番目のトランザクションです。最初のトランザクションは、エンド・ユーザーの端末に BMS マップを送信しています。2 番目のトランザクションは、EXEC CICS RECEIVE MAP コマンドを使用してこのデータを読み取り、ファイル内のレコードを更新して、EXEC CICS SEND MAP コマンドで応答を送信します。

EXEC CICS RECEIVE コマンドおよび EXEC CICS SEND MAP コマンドは、トランザクションの表示ロジックの一部ですが、一方、EXEC CICS READ UPDATE コマンドおよび EXEC CICS REWRITE コマンドは、ビジネス・ロジックの一部です。

```
..
EXEC CICS RECEIVE MAP
..
EXEC CICS READ UPDATE
..
EXEC CICS REWRITE
..
EXEC CICS SEND MAP
..
```

図 60. 単一アプリケーション・プログラムでの CICS 機能

CICS アプリケーション設計でのモジュラー・プログラミングの正常な原則は、表示ロジックをビジネス・ロジックから分離し、連絡域と EXEC CICS LINK コマンドを使用して、この両者から単一のトランザクションを作成することです。図 61 と図 62 にアプリケーション設計に対するこのアプローチを示します。

```
..
EXEC CICS RECEIVE MAP
..
EXEC CICS LINK..
..
EXEC CICS SEND MAP
..
```

図 61. 表示ロジック

```
..
EXEC CICS ADDRESS COMMAREA
..
EXEC CICS READ UPDATE
..
EXEC CICS REWRITE
..
EXEC CICS RETURN..
```

図 62. ビジネス・ロジック

トランザクションのビジネス・ロジックは、一度表示ロジックから分離されて連絡域インターフェースを与えられれば、さまざまな表示方法で再使用することができます。例えば、 409 ページの『分散プログラム・リンク (DPL)』を使用すると、2 層モデル、すなわち CICS ビジネス・ロジック・インターフェース (この場合、表示ロジックは HTTP ベース) による CICS Web サポートをインプリメントできます。

マルチスレッド化: 再入可能なプログラム、準再入可能なプログラム、およびスレッド・セーフ・プログラム

マルチスレッド化は、アプリケーション・プログラムの単一のコピーによって、複数のトランザクションを平行して処理することができる手法です。例えば、1 つのトランザクションが、あるアプリケーション・プログラムを実行し始めるとします。EXEC CICS コマンドに到達し、ディスパッチャーに対して CICS WAIT および呼び出しが生じる場合には、別のトランザクションがそのアプリケーション・プログラムの同じコピーを実行することができます (これを、完了までプログラムを

実行する単一スレッドと比較してください。単一スレッドでは、あるトランザクションによるプログラムの処理が完了してから、別のトランザクションがそれを使用することができます)。

マルチスレッド化では、すべての CICS アプリケーション・プログラムが、準再入可能になっている必要があります。すなわち、これらのプログラムは、入り口点と出口点の間で逐次再使用可能になっている必要があります。EXEC CICS インターフェースを使用する CICS アプリケーション・プログラムは、自動的にこの規則に従います。COBOL、C、および C++ プログラムの場合は、再入可能性は、プログラムが呼び出されるたびに入手される作業用ストレージの最新コピーによって、確実なものにされます。書き込み可能な静的ストレージを持たず、当然再入可能である、C および C++ プログラムに対してさえ、コンパイルまたはプリリンク・ユーティリティー上では、常に RENT オプションを使用する必要があります。CICS 変換プログラムによって挿入される一時変数および DFHEIPTR フィールドは通常、書き込み可能静的変数として定義されるために、RENT オプションが必要です。これらのプログラムを再入可能のままにするには、変数データが、PL/I では静的ストレージとして、あるいはアセンブラー言語ではプログラム CSECT での DC として、表示されるようなことがあってはなりません。

CICS は、アプリケーション・プログラムが再入可能なものとしてコンパイルされ、リンク・エディットされるよう要求すると同時に、プログラムを準再入可能か、スレッド・セーフかのいずれかとして識別します。これらの属性については、以後のセクションで説明します。

準再入可能なアプリケーション・プログラム

CICS は、CICS 管理のタスク制御ブロック (TCB) の下で、ユーザー・プログラムを実行します。プログラムが (そのプログラムのリソース定義の CONCURRENCY 属性で) 準再入可能と定義される場合、CICS は必ず、CICS 準再入可能 (QR) TCB の下で、そのプログラムを呼び出します。マルチスレッド化コンテキストでの準再入可能プログラムの要件は、プログラムが複数の TCB 上で同時に実行される場合ほど、厳密なものではありません。

CICS は、アプリケーション・プログラムが一貫性のある条件を保証するように、それが再入可能であることを要求します。実際には、アプリケーション・プログラムは本当の意味で再入可能でなくても構いません。CICS が期待するのは、「準再入可能性」です。つまり、アプリケーション・プログラムは、制御権が入り口点および EXEC CICS コマンドの前後でそれに渡される際に、首尾一貫した状態でなければならないということです。このような準再入可能性は、アプリケーション・プログラムの個々の呼び出しが、それ以前の実行によって、あるいは複数の CICS タスクによる同時マルチスレッド化によって、影響を受けないことを保証します。

例えば、アプリケーション・プログラムはその実行可能コード、またはプログラム・ストレージ内で定義される変数を修正することは可能ですが、タスクが制御権を失い、別のタスクが同じプログラムを実行する可能性が生ずる前に、そのような変更は取り消すか、コードおよび変数を再度初期化しなければなりません。

CICS の準再入可能ユーザー・プログラム (アプリケーション・プログラム、ユーザー置換可能モジュール、グローバル・ユーザー出口、およびタスク関連ユーザー出口) は、QR TCB に基づく CICS ディスパッチャーによって、制御権を与えられま

す。この TCB の下で実行する場合、プログラムが CICS 要求の間に制御権を解放するまで、他の準再入可能プログラムは実行できないことは確実です。この時点ではユーザー・タスクは中断され、プログラムは依然として「使用中」です。同じプログラムを別のタスク用に再度呼び出すことはできます。これは、アプリケーション・プログラムが同時に複数のタスクによって使用できることを意味します。ただし、一度に 1 つのタスクしか実行できません。

プログラムが確実にお互いの作業用ストレージに干渉できないようにするために、CICS はアプリケーション・プログラムを実行するたびに、個別に作業用ストレージのコピーを入手します。したがって、ユーザー・アプリケーション・プログラムを 11 のユーザー・タスクが使用している場合は、該当する動的ストレージ域 (DSA) に作業用ストレージ域のコピーが 11 あります。

準再入可能性により、プログラムは、グローバル共用リソース (例えば CICS 共通作業域 (CWA)) にアクセスできるようになります。その際には、他のプログラムによる同時アクセスからそれらのリソースを保護する必要もありません。そのようなリソースは、実行プログラムが次の CICS 要求を発行するまで、効率よくそのプログラムに独占的にロックされています。したがって、例えば、アプリケーションは CWA のフィールドを、比較およびスワップ (CS) 命令を使用せずに、あるいはリソースをロック (行列待ち) せずに、更新することができます。

注: CICS QR TCB は、グローバル・リソースにアクセスするすべてのユーザー・タスクが QR TCB に基づいて実行される場合に限り、そのリソースの排他制御を通じて保護を提供します。別の (オープン) TCB の下で同時に実行する他のタスクからの自動保護は、提供しません。

プログラムで長い計算を行う場合には、注意してください。アプリケーション・プログラムは、1 つの EXEC CICS コマンドから次のコマンドまで制御を保つので、QR TCB 上の他のトランザクションの処理は完全に除外されます。しかし、タスク制御 SUSPEND コマンドを使用すれば、他のトランザクション処理に取り掛かることができます。詳細については、647 ページの『第 46 章 タスク制御』を参照してください。ランナウェイ・タスク時間間隔は、トランザクション定義およびシステム初期設定パラメーター ICVR によって制御されていることに、注意してください。CICS は、IVCR で指定されている間隔が満了する前に制御を返さないタスクを除去します。

スレッド・セーフ・プログラム

CICS オープン・トランザクション環境 (OTE) では、アプリケーション・プログラム、タスク関連ユーザー出口 (TRUE)、グローバル・ユーザー出口プログラム、およびユーザー置換可能モジュールは、スレッド・セーフとして CICS に対して定義され、オープン TCB 上で同時に実行できます。

これが原因で、共用リソースへの他のプログラムからの同時アクセスを防ぐために、準再入可能性に依存することはできません。その上、準再入可能プログラムは、オープン TCB の下で並行して実行されるユーザー・タスクがアクセス可能な共用リソースにアクセスする場合には、リスクを伴う可能性もあります。そのため、共用リソースにアクセスするためにユーザー・プログラムが使用する手法は、他のプログラムによる同時アクセスの可能性を考慮に入れなければならないということです。

共用リソースの保全性を保守しながらオープン・トランザクション環境のパフォーマンスを向上させるには、共用リソースへの同時アクセスを禁止するためにシリアル化手法を使用する必要があります。共用リソースにアクセスする際に適切なシリアライズ手法を使用するプログラムを、スレッド・セーフと言います (完全再入可能という用語を使用する場合もありますが、これは誤解を招く可能性があるので、スレッド・セーフの方が望ましいでしょう)。

ENABLE PROGRAM コマンドで OPENAPI オプションを使用して使用可能にされたタスク関連ユーザー出口 (TRUE) が組み込まれているアプリケーションを使用した場合のみ、スレッド・セーフにしてパフォーマンスを向上させることができます。このようなタスク関連ユーザー出口は、オープン API TRUE として知られています。オープン API TRUE は、L8 モード・オープン TCB の下で制御され、作成、管理、およびサブタスク TCB 間での切り替えを行わずに非 CICS の API を使用することができます。CICS DB2 接続機能が使用する CICS DB2 タスク関連ユーザー出口は、CICS が DB2 バージョン 6 以上に接続されている場合に、オープン API TRUE として作動します。これにより、(「*CICS DB2 Guide*」で説明されているように) CICS DB2 アプリケーションがスレッド・セーフであることでパフォーマンスを向上できます。

プログラムをスレッド・セーフにすることの目的は、オープン TCB と QR TCB 間を交互に切り替えることではなく、これらのプログラムがオープン TCB 上に残ることができるようにすることにあります。CICS が DB2 バージョン 6 以降に接続されている場合は、以下のような環境で TCB の切り替えが起こります。

- スレッド・セーフとして定義されていないプログラムから DB2 要求が出されると、CICS は、(プログラムが実行されている) QR TCB からオープン TCB に切り替え、DB2 要求の完了時に再度 QR TCB に切り替えます。
- スレッド・セーフとして定義されていないユーザー出口プログラムが DB2 要求の過程で使用されると、CICS は (DB2 要求が実行されている) オープン TCB から QR TCB に切り替えます。ユーザー出口プログラムは QR TCB 上で実行され、タスクは、DB2 要求を完了するためにオープン TCB に切り替えられます。例えば、XRMIIN および XRMIOUT グローバル・ユーザー出口は、DB2 要求の過程で呼び出される可能性があります。出口プログラムがスレッド・セーフとして定義されていない場合に、この TCB の切り替えが発生します。出口プログラムがスレッド・セーフとして定義されている場合は、オープン TCB 全体を通じて処理が継続します。
- スレッド・セーフとして定義されており、オープン TCB 上で実行されているプログラムが、スレッド・セーフではない任意の EXEC CICS コマンドを呼び出すと、CICS はオープン TCB から QR TCB に切り替えて、非スレッド・セーフ・コードを実行します。さらに、プログラムは QR TCB 上で実行を続行します。プログラムからそれ以上の DB2 要求が出されない場合、QR TCB への切り替えは、残りのアプリケーション・コードの実行に要する時間内における QR TCB の使用量を増加させるため、不利益にしかなりません。ただし、プログラムがさらに DB2 要求を行う場合、CICS は再度オープン TCB に切り替える必要があります。
- スレッド・セーフとして定義されており、オープン TCB 上で実行されているプログラムが、スレッド・セーフとして定義されていないタスク関連ユーザー出口プログラムを呼び出すと、CICS は QR TCB に切り替え、タスク関連ユーザー出口プログラムに制御を渡します。タスク関連ユーザー出口プログラムが処理を完

了すると、アプリケーション・プログラムは、非スレッド・セーフ EXEC CICS コマンドを発行した後に行うものと同じ方法で、QR TCB 上での実行を続行します。

- スレッド・セーフとして定義されており、オープン TCB 上で実行されているプログラムがスレッド・セーフ CICS コマンドを呼び出すと、グローバル・ユーザー出口をコマンドの実行の一環として呼び出すことができます。スレッド・セーフとして定義されていないグローバル・ユーザー出口プログラムを使用する場合、CICS は QR TCB に切り替え、グローバル・ユーザー出口プログラムに制御を渡します。ユーザー出口プログラムが処理を完了すると、CICS はオープン TCB に切り替え、スレッド・セーフ CICS コマンドの処理を続行します。
- スレッド・セーフとして定義されており、オープン TCB 上で実行されているプログラムが完了すると、CICS はタスク終了のために QR TCB に切り替えます。この切り替えは、常に必要です。

CICS DB2 アプリケーションで TCB 切り替えの発生が最大になるのは、プログラムがすべての DB2 要求の後に非スレッド・セーフ・ユーザー出口プログラムおよび非スレッド・セーフ EXEC CICS コマンドを使用した場合です。特に、CICS-DB2 メインライン・パス (例えば、XRMIIN または XRMIOUT で使用可能にされるプログラム) で非スレッド・セーフ出口プログラムを使用すると、CICS をバージョン 5 以前に接続している場合より多くの TCB 切り替えが発生します。

アプリケーション・プログラムをオープン TCB 上に残す場合は、以下のようにします。

1. **プログラムのロジックをスレッド・セーフにする。** つまり、EXEC CICS コマンド間のネイティブ言語コードをスレッド・セーフにする必要があります。プログラムをスレッド・セーフとして CICS に対して定義し、スレッド・セーフではないアプリケーション・ロジックを組み込む場合は、予想できない事態が発生し、CICS は起こりうる結果を防ぐことができません。このトピックの後半には、スレッド・セーフ・アプリケーション・ロジックの作成に関する詳細情報が記載されています。
2. **プログラムがスレッド・セーフ EXEC CICS コマンドのみを使用するようにする。** 「*CICS アプリケーション・プログラミング・リファレンス*」および「*CICS System Programming Reference*」のコマンド構文図では、スレッド・セーフであるコマンドを「このコマンドはスレッド・セーフです」という文で示します。これらは、「*CICS アプリケーション・プログラミング・リファレンス*」や、「*CICS System Programming Reference*」の付録にもリストされています。オープン TCB 上で実行されているプログラムに非スレッド・セーフ EXEC CICS コマンドを組み込む場合、CICS はオープン TCB から QR TCB に切り替えて、コマンドが正常に処理されるようにします。TCB の切り替えは、アプリケーションのパフォーマンスに悪影響を及ぼす可能性があります。

   明示的にコーディングした EXEC CICS コマンドを確認するだけでなく、ユーザーのプログラムで使用され、結果として CICS サービスを使用することになる、高水準言語構文または言語環境プログラムの呼び出し可能サービスにも注意してください。この方法で使用される CICS サービスには、非スレッド・セーフ CICS コマンドが含まれ、QR TCB へのスイッチバックが発生する場合があります。特に、COBOL ステートメント DISPLAY UPON SYSOUT、PL/I と C++ 出力タイプの一部、および言語環境プログラム呼び出し可能サービス CEEMOUT

と CEE3DMP は、データを言語環境プログラム一時データ宛先の CESE と CESO に書き込みます。この処理には EXEC CICS WRITE TD コマンドが含まれ、このコマンドはスレッド・セーフではありません。

3. **プログラムがスレッド・セーフとして CICS に対して定義されるようにする。** これを行うには、プログラム・リソース定義の CONCURRENCY 属性を使用します。プログラムを CICS に対してスレッド・セーフとして定義することによって、(プログラムに組み込まれているすべての EXEC CICS コマンドではなく) アプリケーション・ロジックのみをスレッド・セーフであると指定することになります。CICS は、TCB の切り替えを使用することによって、EXEC CICS コマンドを正常に処理することができます。プログラムをオープン TCB 上で実行できるようにするために、CICS は、アプリケーション・ロジックがスレッド・セーフであることを保証する必要があります。
4. **プログラムの使用する実行パスにある任意のユーザー出口プログラムがスレッド・セーフ標準にコーディングされ、さらに スレッド・セーフとして CICS に対して定義されるようにする。** 動的プラン出口、グローバル・ユーザー出口、またはタスク関連ユーザー出口などがこれに該当します (タスク関連ユーザー出口では、ENABLE PROGRAM コマンドで OPENAPI オプションを指定して出口プログラムを使用可能にすると、CICS が、出口のプログラム定義の CONCURRENCY 設定を OPENAPI で指定変更することに注意してください)。CICS が DB2 バージョン 6 以降に接続されている場合は、CICS DB2 タスク関連ユーザー出口 DFHD2EX1 はスレッド・セーフです。「*CICS DB2 Guide*」には、CICS DB2 要求にとって特に重要なその他の出口に関する詳細情報が記載されています。これには、(スレッド・セーフとして定義されていない) デフォルトの動的プラン出口 DSNCUEXT、(スレッド・セーフとして定義されている) 代替の動的プラン出口 DFHD2PXT、およびグローバル・ユーザー出口 XRMIIN と XRMIOUT が含まれます。また、EXEC CICS コマンドの前後に呼び出されるグローバル・ユーザー出口 XEIIN および XEIOUT、PPT 定義プログラムが制御を受け取る前に呼び出される XPCFTCH についても注意してください。任意のベンダー・ソフトウェアから提供されるユーザー出口プログラムがスレッド・セーフ標準にコーディングされ、スレッド・セーフとして CICS に対して定義されるようにしてください。
5. **ユーザー出口プログラム** (グローバル・ユーザー出口またはタスク関連ユーザー出口) をコーディングする場合は、このプログラムをスレッド・セーフとして定義し、呼び出し元のスレッド・セーフ・アプリケーションと同じ L8 TCB 上で使用できる。

   ENABLE PROGRAM コマンドで OPENAPI オプションを指定してタスク関連ユーザー出口を使用可能にすることで、この出口を L8 TCB の下で制御し、サブタスク TCB の作成と管理を行わずに非 CICS API を使用し、自身のためのオープン・トランザクション環境を利用できるようになります (ENABLE PROGRAM コマンドで OPENAPI オプションを指定して出口プログラムを使用可能にすると、CICS が出口のプログラム定義の CONCURRENCY 設定を OPENAPI で指定変更します)。

   グローバル・ユーザー出口プログラム (リカバリー出口点で実行される出口プログラムなど) が CICS の初期化のなるべく早い時点で使用可能になるようにするため、一般的な方法として、第 1 フェーズの PLT プログラムからそれらのプログラムを使用可能にします。この場合、最初の段階の PLT プログラムは、CICS

の初期化の早い時期に実行されるので、リソース定義は使用可能になっていないことに注意してください。これは、インストールされた PROGRAM 定義 (またはプログラム自動インストール・ユーザー・プログラム) を使用して出口プログラムを CICS に定義できないことを意味します。その代わり、CICS によりデフォルトの定義が自動的にインストールされます (これは「システム自動インストール」と呼ばれています)。このインストールは、プログラム自動インストールが、PGAIPGM システム初期化パラメーターでアクティブとして指定されているかどうかにかかわらず実行されることに注意してください。自動インストール・ユーザー・プログラムは起動されず、定義の変更も許可されません。

CICS は、CONCURRENCY(QUASIRENT) を使用して第 1 フェーズの PLT プログラムで使用可能に設定された出口プログラムをインストールします。すなわち、出口プログラムは準再入可能として定義されます。**第 1 フェーズの PLT グローバル・ユーザー出口プログラムをスレッド・セーフとして定義するには、EXEC CICS ENABLE コマンドにおいて THREADSAFE キーワードを指定します。**このことにより、システム自動インストール・プログラム定義の CONCURRENCY(QUASIRENT) 設定が指定変更されます。第 1 フェーズの PLT プログラムの詳細については、「*CICS Customization Guide*」を参照してください。

グローバル・ユーザー出口プログラムは、ENABLE PROGRAM コマンドで THREADSAFE オプションを使用することにより、使用可能にできます。「*CICS Customization Guide*」には、このタイプのプログラムの作成方法に関する一般情報が記載されています。タスク関連ユーザー出口プログラムの場合、「*CICS Customization Guide*」 で、このタイプのプログラムがオープン・トランザクション環境を正常に活用する方法の詳細について参照してください。OPENAPI オプションを使用して出口プログラムを使用可能にすると、プログラムのロジックがスレッド・セーフであることを CICS に示すことになる点に注意してください。

プログラムのアプリケーション・ロジックをスレッド・セーフにするには、共用リソースへのアクセス時に適切なシリアル化手法を使用するようにしてください。ファイルや一時データ・キュー、一時記憶キュー、DB2 テーブルなど、ほとんどのリソースでは、CICS 処理は自動的に、スレッド・セーフ方式でのアクセスを保証します。上述したように、これらのリソース上で作動する CICS コマンドの一部は、コマンドをオープン TCB 上で実行できるようにする適切なシリアル化手法を使用するようにコーディングされます (つまり、これらはスレッド・セーフ・コマンドです)。これに該当しない場合は、CICS は、QR TCB に強制的に切り替えてスレッド・セーフ処理を保証し、コマンドの動作にかかわらずリソースへのアクセスをシリアライズ化します。しかし、共用ストレージのような、ユーザー・プログラムが直接アクセスするリソースでは、スレッド・セーフ処理はユーザー・プログラムの責任において保証されます。

共用ストレージの典型的な例は、CICS CWA、グローバル・ユーザー出口のグローバル作業域、および、共用オプションを持つアプリケーション・プログラムが明示的に獲得するストレージです。以下の EXEC CICS コマンドのオカレンスを検索すると、アプリケーション・プログラムがこれらのタイプの共用ストレージを使用しているかどうかがわかります。

- ADDRESS CWA
- EXTRACT EXIT

- GETMAIN SHARED

これらのコマンドの一部はそれ自体がスレッド・セーフですが、すべてのコマンドがグローバル・ストレージ域にアクセスできるため、これらのコマンドに従い、グローバル・ストレージ域を使用するアプリケーション・ロジックは、非スレッド・セーフになる可能性があります。スレッド・セーフかどうかを確認するには、アプリケーション・プログラムに、並行更新に対する保護に必要な同期ロジックが組み込まれていなければなりません。

ロード・モジュール・スキャナー・ユーティリティーには、サンプル・テーブル DFHEIDTH が組み込まれており、これを使用すると、既存のアプリケーション・プログラムのアプリケーション・ロジックがスレッド・セーフであるかどうかを確認することができます。DFHEIDTH には、共用ストレージへのアクセスを可能にする CICS コマンドのフィルター・セットが含まれています。シリアル化を保証して並行更新を防ぐために必要な同期ロジックがプログラムの適所にある場合を除き、これらのコマンドを使用するとプログラムを非スレッド・セーフにすることができます。

提供されている DFHEIDTH は、プログラムに非スレッド・セーフ CICS コマンド (すなわち、QR TCB への切り替えを発生するコマンド) があるかどうかについてテストするのではなく、アプリケーション・ロジックが非スレッド・セーフになる可能性を生じさせる CICS コマンドが、アプリケーションで使用されているかどうかを判別するものであることに注意してください。提供されたサンプル・テーブルを変更して、検出したい非スレッド・セーフ CICS コマンドを追加することができます。別のサンプル・テーブル DFHEIDNT も提供されており、これは、ユーザーのアプリケーションにある非スレッド・セーフの CICS コマンドを特定する場合に役立ちます。ロード・モジュール・スキャナーの使用法については、「*CICS Operations and Utilities Guide*」を参照してください。

注: 共用リソースを使用するプログラムを識別する場合は、自己修正する任意のプログラムを組み込まなければなりません。そのようなプログラムは、ストレージを効率的に共用しているので、リスクを伴うと考えなければなりません。

共用リソースにアクセスする際にスレッド・セーフ処理を提供するために使用できる手法は、次のとおりです。

- そのリソースが比較およびスワップ命令を使用して、他のプログラムによって同時に変更されている場合は、アクセスを再試行する。
- 排他制御権を獲得して、他のプログラムがそのリソースにアクセスできないようにするには、以下の方法で、そのリソースをキューに入れる。
  - アプリケーション・プログラムでは、EXEC CICS ENQ コマンド
  - グローバル・ユーザー出口プログラムでは、CICS エンキュー (NQ) ドメインへの XPI ENQUEUE 関数呼び出し
  - ENQ などの MVS サービス (L8 TCB が使用可能な場合に限り、オープン API タスク関連ユーザー出口で) QR TCB の下で実行できるアプリケーションで MVS サービスを使用すると、待ち状態に置かれている TCB が原因で、パフォーマンスが低下する可能性がある点に注意してください。
- EXEC CICS LINK コマンドを使用して準再入可能プログラムにリンクすることによって、準再入可能と定義されるプログラムに限っては、共用リソースへのアクセスを実行する。

この手法は、スレッド・セーフ・アプリケーション・プログラム、およびオープン API タスク関連ユーザー出口にのみ、適用されます。準再入可能と定義されるリンク済みプログラムは QR TCB の下で稼働し、CICS の準再入可能性が提供するシリアル化を利用することができます。準再入可能モードの場合でさえ、シリアル化は、プログラムが制御権を保持し、待機しない場合にのみ提供されることに注意してください (詳細については、250 ページの『準再入可能なアプリケーション・プログラム』を参照してください)。

- 共用リソースにアクセスするすべてのトランザクションを、制限付きトランザクション・クラス (TRANCLASS)、すなわち MAXACTIVE(1) として指定される、アクティブ・タスクの数で定義するクラスに入れる。この最後の方法は、非常に粗雑なロック機構を効率的に提供しますが、パフォーマンスに重大な影響を及ぼす可能性があります。

注: スレッド・セーフという用語は、個々のプログラムのコンテキストで定義されるものですが、ユーザー・アプリケーション全体は、共用リソースにアクセスするすべてのアプリケーション・プログラムが規則に従っている場合に、スレッド・セーフと考えることができます。スレッド・セーフ規格に従って正しく書かれているプログラムは、同じリソースにアクセスする別のプログラムがスレッド・セーフ規則に従っていない場合は、共用リソースを安全に更新することができません。

静的または動的に呼び出されるルーチンのスレッド・セーフに関する考慮事項

プログラムに CONCURRENCY(THREADSAFE) を定義する場合、そのプログラムから静的または動的に呼び出されるすべてのルーチン (例えば、COBOL ルーチン) もまたスレッド・セーフの規格でコーディングする必要があります。

プログラム相互のリンクに EXEC CICS LINK コマンドが使用されている場合、そのプログラム・リンクのスタック・レベルがインクリメントされます。ただし、静的または動的に呼び出されるルーチンは、CICS のコマンド・レベル・インターフェースを使用した引き渡しを必要としないため、プログラム・リンクのスタック・レベルがインクリメントされません。COBOL ルーチンの場合の静的呼び出しでは、単純な分岐リンクがリンク・エディット時に解決されるアドレスに関与します。動的呼び出しの場合は関与するプログラム定義がありますが、そのプログラム定義は言語環境でプログラムのロードを許可する場合にのみ必要です。その後、単純な分岐リンクが実行されます。そのため、静的または動的の方式のいずれかでルーチンが呼び出される場合、CICS はそれがプログラムの変更であるとみなしません。そのルーチンを呼び出したプログラムがまだ実行中であるとみなされ、そのプログラムのプログラム定義が依然として現行のものであるとみなされます。

呼び出し側プログラムのプログラム定義で CONCURRENCY(THREADSAFE) が宣言されている場合、呼び出される側のルーチンもその仕様に準拠している必要があります。CONCURRENCY(THREADSAFE) 属性が指定されたプログラムは DB2 呼び出しから戻ってもオープン TCB 上に残るため、スレッド・セーフではないプログラムに対しては適切ではありません。例えば、トランザクションの初期プログラムであるプログラム A が、COBOL ルーチンであるプログラム B に対して動的呼び出しを発行する状況を考えます。CICS のコマンド・レベル・インターフェースは関与していないため、CICS はプログラム B への呼び出しを認識せず、現行プログラムがプログラム A であると認識します。プログラム B は DB2 呼び出しを発行します。DB2 呼び出しから戻った時点で、CICS はそのプログラムがオープン TCB

に残ることができるかどうか、またはそのプログラムが QR TCB にスイッチバックしてスレッド・セーフの処理を確実にする必要があるかどうかを決定する必要があります。この実行のため、CICS は CONCURRENCY 属性を調べ、現行プログラムがプログラム A であるとみなします。プログラム A に CONCURRENCY (THREADSAFE) が定義されている場合、CICS はオープン TCB での処理の継続を許可します。実際にはプログラム B が実行中であるため、処理を安全に継続するには、プログラム B はスレッド・セーフの規格でコーディングされている必要があります。

OPENAPI プログラム

OPENAPI プログラム (つまり、リソース定義内で OPENAPI 属性を指定して定義されたプログラム) は、(EXECKEY 属性に応じて) メイン CICS QR TCB の代わりに、オープン L8 または L9 モード TCB 上で起動されます。

OPENAPI プログラムの責任

OPENAPI プログラムは、QR TCB による制約からは自由ですが、それでも CICS システム全体、およびそれを使用する L8 または L9 TCB の将来のユーザーの両方に対して責任があります。L8 または L9 TCB は、それが割り振られる先の CICS タスクで使用されるよう専用化されますが、CICS タスクが完了すると、その TCB は、「クリーン」な状態であれば、ディスパッチャーが管理するその種の TCB のプールに返されます。(ここでクリーンではない TCB とは、L8 または L9 モードの TCB を使用するタスクが、OPENAPI プログラムにおいて処理不能の異常終了になることを意味します。そのプログラムにおいて、CICS で検出できないスレッド・セーフの制約事項に違反したことを意味しているわけではありません。) TCB は特定の OPENAPI プログラムによる使用に専用化されているわけではなく、L8 モード TCB の割り振り先である CICS タスクによって呼び出される、すべての OPENAPI プログラムおよび OPENAPI TRUE によって使用されることに注意してください。また、OPENAPI プログラムを呼び出すアプリケーション・プログラムがスレッド・セーフの規格でコーディングされ、スレッド・セーフとして CICS に定義された場合は、そのプログラムからリターン時にも L8 モード TCB での実行を継続します。

スレッド・セーフに関する制約事項: OPENAPI プログラムでは、以下に対して、問題の原因となるような方法でオープン TCB 環境の実行を処理してはなりません。

- オープン TCB で実行する可能性のあるアプリケーション・プログラム論理
- 同一のタスクにより呼び出される OPENAPI TRUE
- オープン TCB を使用する可能性のある将来のタスク
- CICS 管理コード。

ユーザーの責任において、ユーザーの OPENAPI プログラムで他の (非 CICS) API を使用する場合は、以下について注意する必要があります。

- CICS サービスを呼び出す場合、または CICS に戻る場合、OPENAPI プログラムでは、そのプログラムへの入り口での状態に MVS プログラミング環境を復元する必要があります。これには、仮想記憶間モード、ASC モード、要求ブロック (RB) レベル、リンケージ・スタック・レベル、TCB ディスパッチング優先順位などとともに、追加されたすべての ESTAE の取り消しが含まれます。

- CICS タスクの終了時に、OPENAPI プログラムは、別の CICS トランザクションによる再利用に適合した状態でそのオープン TCB から抜けるようにする必要があります。特に、タスクの終了のために特別に獲得した非 CICS リソースは、必ずすべて解放してください。以下のようなリソースがその対象になります。
  - 動的に割り振られたデータ・セット
  - オープン ACB または DCB
  - STIMERM 要求
  - MVS 管理のストレージ
  - ENQ 要求
  - 接続されたサブタスク
  - ロードされたモジュール
  - 所有するデータ・スペース
  - 追加されたアクセス・リスト項目
  - 名前/トークンのペア
  - 固定ページ
  - セキュリティー設定 (TCBSENV をゼロに設定する必要があります)
- OPENAPI プログラムでは、CICS 全体のオペレーションに影響する以下の MVS システム・サービスを使用しないでください。
  - CHKPT
  - ESPIE
  - QEDIT
  - SPIE
  - STIMER
  - TTIMER
  - XCTL / XCTLX
  - すべての TSO/E サービス。
- OPENAPI プログラムでは、L8 または L9 モード TCB において、MVS 言語環境の各サービスを使用している言語環境プログラムを起動しないでください。これは、L8 および L9 モード TCB が、CICS の各サービスを使用した言語環境用に初期化されているためです。

FORCEQR システム初期設定パラメーターの使用

OTE を活用するためにスレッド・セーフとして定義されたプログラムと共にアプリケーションを実行している場合 (例えば、CICS DB2 アプリケーション内)、 1 つ以上のプログラムがスレッド・セーフでないと通知する問題が生じる可能性があります。この問題が発生した場合は、**FORCEQR** システム初期設定パラメーターを使用して、すべてのアプリケーション・プログラムを強制的に QR TCB に置くことができます。これは、問題の調査中にサービス休止状態にすることができないアプリケーションが置かれている実動領域で、特に有効です。

このパラメーターのデフォルトは FORCEQR=NO です。この値は、CICS が、ユーザーのプログラム・リソース定義の中の CONCURRENCY 属性を受け入れることを意味します。一時的な処置として、スレッド・セーフ定義されたプログラムに関連

する問題を調査および解決する間、FORCEQR=YES と設定することができます。プログラムが OTE 下 でオープン TCB の使用を再開する用意ができたら、これを FORCEQR=NO に戻すことを忘れないでください。

再入不能プログラム

CICS が実行する再入不能アプリケーション・プログラムを妨げるものは、何もありません。ただし、その種のアプリケーション・プログラムは、マルチスレッド環境では、終始一貫した結果を提供することはありません。

再入不能アプリケーション・プログラム、または関連したアプリケーション・プログラムの実行により修正可能な、テーブルまたは制御ブロックを使用するには、リソース定義で RELOAD(YES) オプションを指定してください。RELOAD(YES) の結果、プログラムまたはモジュールの最新コピーが、各要求ごとにストレージにロードされます。このオプションで、再入不能プログラムまたはテーブルにアクセスするマルチスレッド・タスクが、それぞれそのプログラムの独自のコピーから作業し、CICS 領域で実行される他の並列タスクによって、プログラムの別のバージョンに施される変更の影響を受けないことが、保証されます。

RELOAD(YES) について詳しくは、「*CICS Resource Definition Guide*」の PROGRAM 定義属性を参照してください。

CICS は、RENT 属性でリンク・エディットされたプログラムを、CICS 読み取り専用動的ストレージ域 (DSA) にロードします。CICS は、RMODE(24) プログラム用の RDSA、および RMODE(ANY) プログラム用の ERDSA を使用します。デフォルトでは、これら DSA のストレージは読み取り専用のキー 0 保護ストレージから割り振られ、それらにロードされるモジュールを、キー 0 または監視プログラム状態で稼働するプログラム以外のすべてのプログラムから保護します (CICS が RENTPGM=NOPROTECT システム初期化パラメーターを使用して初期設定を行う場合は、読み取り専用のキー 0 ストレージは使用せず、代わりに CICS キー・ストレージを使用します)。

再入不能プログラムまたはモジュールを実行したくない場合は、読み取り専用ではない DSA にロードしなければなりません。SDSA および ESDSA は、再入不能ユーザー・キー・プログラムおよびモジュールのための、ユーザー・キー・ストレージ域です。

CICS DSA の詳細については、「*CICS System Definition Guide*」の動的ストレージ域を参照してください。

トランザクション内のデータの格納

CICS は、トランザクション内のデータおよびトランザクション間のデータを格納するために各種の機能を提供します。それぞれの機能は、トランザクション内の他のプログラムおよび他のトランザクションに利用可能なようにデータを残す方法、それを実施する方法、さらにそのオーバーヘッド、リカバリー、およびエンキュー特性によって異なっています。

トランザクションの存続期間中、存在する格納機能には、次のものがあります。

- トランザクション作業域 (TWA)

- ユーザー・ストレージ (SHARED オプションを使用しないで発行された、GETMAIN コマンドを介して)
- COMMAREA
- プログラム・ストレージ

これらの区域のすべては主記憶装置の機能で、同一の基本ソース、すなわち動的ストレージ域 (DSA) および拡張動的ストレージ域 (EDSA) から得られます。これらのどれもリカバリー可能ではなく、どれもリソース保護キーによって保護することはできません。しかし、これらの区域は、アクセス可能性と期間が異なるので、それぞれが異なるストレージ要件を満たします。

トランザクション作業域 (TWA)

トランザクション作業域 (TWA) は、トランザクションの開始時に割り振られ、2進ゼロに初期設定されます。この区域はトランザクションの全期間を通じて持続し、トランザクション内のすべてのローカル・プログラムにアクセス可能です。

分散プログラム・リンク・コマンドを介してリンクされるすべてのリモート・プログラムは、クライアント・トランザクションの TWA に対するアクセス権をもっていません。 TWA のサイズは、トランザクション・リソース定義の TWASIZE オプションによって決定されます。このサイズが非ゼロの場合、TWA は常に割り振られます。TWASIZE の決定の詳細については、「*CICS Resource Definition Guide*」のを参照してください。

また、TWA の使用と関連したプロセッサー・オーバーヘッドは最小です。これにアクセスするために GETMAIN コマンドの必要はなく、単一の ADDRESS コマンドを使用してそのアドレスを指定します。TASKDATAKEY オプションは、TWA が CICS キーまたはユーザー・キー・ストレージのどちらで入手されるかを決定します (CICS キーおよびユーザー・キー・ストレージについての詳細は、 653 ページの『ストレージ制御』を参照してください)。トランザクション定義の TASKDATALOC オプションは、獲得したストレージが 16MB 境界の上に置けるかどうかを決定します。

TWA は、かなり小さなデータ・ストレージ要件、および、サイズが比較的固定されており、トランザクションの間は多少とも使用される大きいストレージ要件に適しています。TWA はトランザクションの期間全体にわたって存在するので、TWA サイズが大きいと、非会話型トランザクションより、会話型トランザクションの場合の方がはるかに大きく影響します。

ユーザー・ストレージ

ユーザー・ストレージは、トランザクション内のすべてのプログラムに利用可能ですが、LINK コマンドまたは XCTL コマンドを使用してプログラム間で受け渡しするためには、いくらか労力が必要です。そのサイズは、固定していなくて、トランザクションが要求したときにただちに獲得し (GETMAIN コマンドを使用して)、必要なくなるとすぐに戻すことができます。したがって、ユーザー・ストレージは、サイズがまちまちな、またはトランザクションより短期間の、大きなストレージ要件にとって有用です。

USERDATAKEY および CICSDATAKEY で GETMAIN コマンドの TASKDATAKEY オプションを指定変更する方法について詳しくは、 653 ページの『ストレージ制御』を参照してください。

GETMAIN コマンドの SHARED オプションによって、獲得済みのストレージはタスクの終了後に保存されます。ストレージは、同一端末であるタスクから次のタスクに連絡域経由で受け渡しすることができます。最初のタスクは、RETURN コマンドの COMMAREA オプションに連絡域のアドレスを返します。2 番目のタスクは、ADDRESS コマンドの COMMAREA オプションのアドレスにアクセスします。ストレージが必ず共通ストレージ内にあるようにするためには、GETMAIN コマンドの SHARED オプションを使用する必要があります。

GETMAIN コマンドはプロセッサー・オーバーヘッドがあるため、少量のストレージにはこのコマンドを使用しないでください。少量の場合は TWA を使用するか、あるいは区域を一緒にして大きい要求にグループ化する必要があります。GETMAIN コマンドによって獲得されるストレージは、結合された要求を使用した場合には、いくらか長く保留することができますが、プロセッサー・オーバーヘッドと参照セットのサイズの両方とも減少します。

LINK コマンドおよび XCTL コマンドにおける COMMAREA

連絡域 (COMMAREA) は、トランザクション内の 2 つのプログラムの間、または同一端末の 2 つのトランザクションの間で、情報を転送するために使用する機能です。トランザクション間での COMMAREA の使用については、 286 ページの『RETURN コマンドでの COMMAREA の使用』を参照してください。

2 つの関連したプログラムが明示的なステップを踏んで、トランザクション内で後から呼び出すことがある他のプログラムにデータを利用可能にしない限り、COMMAREA 内の情報はその 2 つの関連したプログラムにのみ利用可能です。あるプログラムを別のプログラムにリンクする場合には、COMMAREA はリンクしているプログラムがアクセス権を持つ任意のデータ域とすることができます。COMMAREA はしばしばそのプログラムの作業用ストレージまたは LINKAGE SECTION にあります。この区域で、リンクしているプログラムは、呼び出しているプログラムにデータを渡すことと、そのプログラムから結果を受け取ることの両方を行うことができます。

あるプログラムが別のプログラムに制御権を移動する (XCTL コマンド) 場合には、制御権が移動した後は、呼び出し側プログラムとその制御ブロックはもはや利用可能ではなくなっている可能性があるので、CICS は指定の COMMAREA をストレージの新規区域にコピーすることがあります。いずれか一方の場合に、制御を受け取るプログラムに区域のアドレスが渡され、CICS コマンド・レベル・インターフェースがアドレッシング可能にセットアップします。詳しくは、 325 ページの『第 21 章 プログラム制御』を参照してください。XCTL が使用される場合、CICS は COMMAREA を 16MB 境界より下にコピーすることによって、COMMAREA を受け取るプログラムが COMMAREA をアドレッシングできるようにします。

COMMAREA は、受け取るプログラムのアドレッシング・モードおよび EXECKEY 属性に応じて、必要な場合には USERKEY ストレージにコピーされます。EXECKEY についての詳細は、 653 ページの『ストレージ制御』を参照してください。

CICS は、伝送されるバイトの数を減らすように設計されたアルゴリズムを含んでいます。そのアルゴリズムは、伝送の前には、COMMAREA から一部の後書きの 2 進ゼロを除去し、伝送の後でそれらを復元します。これらのアルゴリズムの操作は、常にフルサイズの COMMAREA を参照するアプリケーション・プログラムにとって透過的です。

LINK コマンドにおいて COMMAREA を使用する際のオーバーヘッドは、ごく小さなものです。CICS が、プログラムで使用している、より大きな区域のストレージから COMMAREA を作成する場合は、XCTL および RETURN コマンドを使用するとオーバーヘッドが多少大きくなります。

LINK および XCTL コマンドのチャネル

CICS プログラム間のデータ転送の最新の方法として、連絡域 (COMMAREA) を使用する代わりに、**チャネル**を使用します。チャネルには、COMMAREA に対するいくつかの利点があります。319 ページの『チャネルの利点』を参照してください。 LINK または XCTL コマンドでチャネルを受け渡すには、COMMAREA オプションの代わりに CHANNEL オプションを使用します。

チャネルについては、289 ページの『第 20 章 拡張プログラム間データ転送: 新しい COMMAREA としてのチャネル』で説明しています。

プログラム・ストレージ

CICS は、CICS プログラムを使用している各トランザクション用に、その CICS プログラムの変数域のコピーを別個に作成します。この区域は、**プログラム・ストレージ**と呼ばれます。この区域は、COBOL では WORKING-STORAGE SECTION と呼ばれ、C、C++、および PL/I では自動ストレージと呼ばれ、アセンブラー言語では DFHEISTG セクションと呼ばれます。 TWA のように、この区域は固定サイズで、CICS によって割り振られ、GETMAIN コマンドを発行する必要はありません。EXEC CICS インターフェースは、アドレッシング可能に自動的にセットアップします。しかし、TWA とは違い、このストレージはトランザクションの期間ではなく、プログラムの実行中だけ持続します。この点が、プログラムの外側に必要ないデータ域および小さいか、あるいは大きい場合には、サイズが固定でプログラムの実行時のすべてまたはほとんどに必要なデータ域の場合に、プログラム・ストレージを有用にしています。

一時記憶域

一時記憶域は、複数のトランザクションに利用可能にする必要があるデータを格納するための、CICS の基本的な機能です。

一時記憶域のデータ項目は、キューに保持され、キューの名前は、データを保管するプログラムによって動的に割り当てられます。複数の項目が入っている一時記憶キューは、小さいデータ・セットと考えることができ、そのレコードは、項目番号によって順次または直接のいずれかでアドレッシングすることができます。キューに単一項目しか入っていない場合には、名前付きのスクラッチパッド域と考えることができます。

一時記憶域のデータ共用は、メインまたは補助記憶装置が 1 つ以上の一時記憶域プールによって置換することができるということを意味します。

一時記憶域は、次の方式によって実行されます。

- 最初の項目を作成するコマンド上で指定される内容によって決定される、特定のキューを使用することによって。
- キューが動的ストレージ域から取ったスペース内の主記憶装置に保持されるように、MAIN オプションを指定することによって。
- キューが入力シーケンス化した VSAM データ・セットに書き込まれるように、AUXILIARY オプションを使用することによって。

いずれの方式を使用したとしても、項目の索引は、CICS が主記憶装置に作成します。

QNAME オプションがインストール済みの TSMODEL リソース定義の接頭部と一致する場合は、TSMODEL で指定される MAIN または AUXILIARY 値は、コマンドで指定される優先順位をとります。

一時記憶キューの定義に TSMODEL を使用する方法の詳細については、「*CICS Resource Definition Guide*」のを参照してください。

一時記憶域のデータ共用を追加すると、並行してサポートすることができる一時記憶キューの別のタイプが付与されます。これらの一時記憶キューは、ローカル、リモート、または共有として定義することができ、カップリング・ファシリティーの TS プールに保管することができます。

これらの方式には、次の特性があります。

- 主一時記憶域には、補助記憶装置が必要とするよりはるかに多くの仮想記憶域が必要です。一般に、存続時間が短いか、または頻繁にアクセスされる小さいキューにだけ使用する必要があります。補助一時記憶域は、存続時間が比較的長いか、あるいはたまにしかアクセスされない比較的大容量のデータのために特別に設計されています。主記憶装置に入れるキューと補助記憶装置に入れるキューを決めるためには、存続時間が 1 秒を超えるかどうかといった目安を設定するといいでしょう。
- 補助記憶装置のキューはリカバリー可能にできますが、主記憶装置のキューはリカバリー可能にはできません。
- 共用一時記憶域は、リカバリー不能キューにのみ適用されます。
  - 一度に 1 つのトランザクションしか、リカバリー可能一時記憶キューを更新することはできません。したがって、キューをリカバリー可能にすることを選択する場合には、エンキューが発生する可能性があることを念頭に置いてください。
  - 競合をできるだけ排除するためには、バッファーおよび VSAM ストリングが十分になるようにする必要があります。
- タスクが一時記憶域に書き込もうと試みて、使用可能なスペースがない場合には、CICS は通常、タスクを中断します。しかし、このタスクは、HANDLE CONDITION NOSPACE コマンドか、あるいは WRITEQ TS コマンドの RESP または NOHANDLE オプションのいずれかを使用することによって、この状況において制御を回復することができます。タスクは、中断された場合には、他のタスクが主記憶装置または VSAM データ・セット内の必要なスペースを解放するまで再開されません。これは、特に待機中のタスクが排他使用リソースを所有し

ている場合は、予期しない応答遅延になることがあります。応答遅延の場合には、排他使用リソースを必要としている他のすべてのタスクも待機しなければなりません。

- リカバリーの必要がない低容量のシステムで主一時記憶域を排他使用すると、一段と効率を上げることができます。VSAM アクセス方式の追加主記憶装置の所要量に対する必要性と、一時記憶域レコード用の主記憶装置に対する必要性を持ったさらに大きい一時記憶域プログラムとの、全体のバランスを取る必要があります。

一時記憶域について一般に考慮すべき点は、次の通りです。

- 一時記憶キューに対するデータの書き込みまたは読み取りを行うたびに、EXEC CICS コマンドを使用する必要があり、CICS は内部索引を使用して、データの検索または挿入を行う必要があります。これは、主一時記憶域を使用するためのオーバーヘッドが、CWA または TCTUA の場合より大きいことを意味します。補助記憶装置 (通常最も頻繁に使用される) だけでなく、通常、データ・セット I/O もあるので、オーバーヘッドはさらに増加します。
- 一時記憶域は必要になるまで割り振る必要はありません。一時記憶域は必要な長さだけ保持する必要があり、項目を作成するコマンドを発行するまで項目サイズは固定されません。これは、比較的大容量のデータおよび長さまたは期間が変化するデータには良好な選択となります。
- 一時記憶キューは作成時に名前を付けることができるという事実は、保管済みデータに対して非常に強力な形式の直接アクセスを提供します。キュー名内に端末名またはレコード・キーを含めることによって、端末、データ・セット・レコードなどに対するスクラッチパッド域にアクセスすることができます。
- 一時記憶域では、リソース保護が可能です。

区画内一時データ

区画内一時データには、補助一時記憶域と共通する特性がいくつかあります (***区画外*** 一時データについて詳しくは、279 ページの『効果的な順次データ・セットのアクセス』を参照してください)。一時記憶域のように、区画内一時データは、単一データ・セットに、CICS が主記憶装置で保守する索引と一緒に保持されるデータのキューからなります。

一時データは、補助一時記憶域を使用するのと同じ多くの目的で使用できますが、次のような重要な違いがあります。

- 一時データには一時記憶域と同じ動的特性はありません。一時記憶キューと違って、一時データ・キューは、データがアプリケーション・プログラムによって書き込まれるときに作成することができません。しかし、一時データ・キューは、CICS の実行中は RDO を使用して定義し、インストールすることができます。
- 一時データ・キューは順次に読み取らなければなりません。各項目は一度だけ読み取ることができます。トランザクションが項目を読み取った後で、その項目はキューから除去され、その他のどのトランザクションでも利用不能になります。対照的に、一時記憶キューの項目は、順次または直接に (項目番号によって) 読み取ることができます。これらは、何回でも読み取ることが可能で、キュー全体が除去されるまでキューから除去されません。

これら 2 つの特性は一時データをスクラッチパッド・データに不適当なものとしますが、監査証跡および印刷出力のような待機データには適切なものとします。実際、順次に一度読み取られるデータの場合には、一時データは一時記憶域よりは好ましいものです。

- 一時記憶キューの項目は変更可能ですが、一時データ・キューの項目の変更はできません。
- 一時データ・キューは常にデータ・セットに書き込まれます (主一時記憶域と対応する一時データの形式はありません)。
- キューに項目を書き込むことによって、特定のトランザクションを (例えば、キューを処理するために) 開始させるように、一時データ・キューを定義することができます。START コマンドを使用して、同様の機能を実行できる場合がありますが、一時記憶域には、「トリガー」メカニズムと対応するものはなにもありません。
- 一時データには、一時記憶域より多くのリカバリー・オプションがあります。一時データ・キューは、物理的にも論理的にもリカバリー可能です。
- 区画内および区画外の一時データに対するコマンドは同一なので、2 つのタイプのデータ・セットを切り替えることができます。これをするためには、アプリケーション・プログラムそのものではなく、一時データ・キューの定義のみを変更してください。一時記憶域には、この種の機能と対応するものはありません。

GETMAIN SHARED コマンド

GETMAIN コマンドの SHARED オプションを使用して獲得したストレージは、その獲得側のタスクの終了時に解放されません。こうすることによって、あるタスクは、別のタスクが使用するためにデータをストレージに残すことができます。このストレージは、獲得したタスクまたは別のタスクのいずれかによって FREEMAIN コマンドが発行されるまで、解放されません。

ユーザー独自のデータ・セット

ユーザー独自のデータ・セットを使用して、トランザクションとトランザクションの間のデータを保管することもできます。この方式のオーバーヘッドは、処理される命令、バッファー、制御ブロック、およびユーザー・プログラミング要件という点ではおそらく最大ですが、特別の機能および柔軟性を提供します。データ・セットをリカバリー可能リソースとして定義できるだけでなく、順方向リカバリーのためにデータ・セットに対する変更内容のログをとることもできます。アクセスの競合を防ぐために、(一時記憶域および一時データ・セットと同様に) データ・セットに対するストリングの数を指定することができ、リソース保護の手段も活用できます。

CICS コマンドに渡される区域の長さ

CICS コマンドに LENGTH オプションが含まれている場合には、長さは通常符号付きハーフワード 2 進数値として受け入れられます。長さの理論的上限は 32KB です。実際には、限界はこれより小さくなり、それぞれのコマンドによって変化します。また、限界はデータ・セット定義、リカバリー可能性要件、バッファー・サイズ、およびローカル・ネットワーキング特性によって異なります。

LENGTH オプション

COBOL、C、C++、PL/I、およびアセンブラー言語では、変換プログラムは長さを処理します。LENGTH オプションの指定が必要な場合の詳細などのプログラミング情報については、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。 CICS コマンド・オプションで指定する長さは、できれば、24KB を超えないようにする必要があります。

多くのコマンドがアプリケーション・プログラムと CICS の間でデータの転送を行います。すべての場合で、転送するデータの長さは、アプリケーション・プログラムが指定する必要があります。

多くの場合、SET オプションが指定されていれば、LENGTH オプションは必ず指定しなければなりません。各コマンドの構文およびそれに関連するオプションにより、この規則を適用するかどうかが示されます。

WAIT EXTERNAL コマンドおよび多くの QUERY SECURITY コマンドには、リソースの状況または定義を示すオプションがあります。CICS は、CICS 値データ域に、これらのオプションに関連する値を提供します。これらのオプションは、括弧で囲まれた cvda を持つコマンドの構文で示されます。CVDA のプログラミング情報については、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

ジャーナル・コマンドの場合、この制限は LENGTH 値と PFXLENG 値の合計に適用されます (377 ページの『ジャーナル処理』を参照してください)。

ジャーナル・レコード

ジャーナル・レコードの場合には、ジャーナル・バッファー・サイズが限界を 64KB 以下にするように要求されることがあります。この限界は、LENGTH 値と PFXLENG 値の合計に適用されることに注意してください。

データ・セット定義

一時記憶域、一時データ、およびファイル制御の場合には、データ・セット定義の限界が 24KB より低く設定されることがあります。詳しくは、「*CICS System Definition Guide*」のデータ・セットの定義(データ・セットの作成に関する情報) および「*CICS Resource Definition Guide*」のFILE リソース定義 (各ファイルのリソース定義に関する情報) を参照してください。

推奨

すべてのシステムのすべてのコマンドで、LENGTH 指定には 24KB が良好な作業限界です。ユーザー指定のレコードおよびバッファー・サイズを満たす限り、この限界によって、エラーが起こったり、アプリケーションに対する制約になったりすることはありそうにもありません。

24KB の限界が大きな障害となることはおそらくありません。オンライン・プログラムが、効率および応答時間のために、これほど大量のデータを処理することはめったにありません。

注: LENGTH オプションの値は、コマンドがアドレッシングするデータ域の長さを超えないようにしてください。

エラーの最小化

このセクションでは、ご使用のアプリケーションからエラーをなくす方法について説明します。これらの方法の中には、プログラミングだけでなく操作およびシステムにもあてはまるものがあります。

単独では完全に実行される 2 つのアプリケーション・システムを一緒に実行した場合によく起こることは、パフォーマンスが低下し、「ロックアウト」または待機を感じ始めるという点です。各システムの有効範囲は、まだ完全には定義されていません。

設計の優れたアプリケーション・システムで重要なことは、次の点です。

- すべてのレベルで、各機能が正しく記述された入出力で明確に定義されている
- システムが使用するリソースが正しく定義されている
- 他のシステムとの対話が明確になっている

アプリケーション・エラーからの CICS の保護

このタスクについて

アプリケーション・プログラムのエラーを最小化するために使用することができる、各種のツールおよび技法があります。一般には次のとおりです。

- ストレージ保護機能を使用すれば、CICS コードおよび制御ブロックが、ご使用のアプリケーション・プログラムによって上書きされないようにすることができます。この機能を使用するかしないかは、CICS システム初期化パラメーターを使用して選択することができます。このインターフェースについて詳しくは、「*CICS System Definition Guide*」のストレージ保護を参照してください。
- GETMAIN コマンドなどを使用する手法によって起きる問題を回避するため、標準機能の使用を検討してください。

アプリケーションのテスト

このタスクについて

アプリケーションのテストには、次の一般的な規則を適用してください。

- 実動 CICS システムではなく、テスト・システムを使用してテストしてください。そうすることで、「実動中」のデータベースに影響を与えずにエラーを切り分けることができます。
- できれば、アプリケーション開発者以外の人にテストしてもらいます。
- テストに使用するデータを文書化します。
- アプリケーションは、何回か繰り返してテストします。アプリケーションのテストの詳細については、171 ページの『第 10 章 アプリケーションのテスト』を参照してください。

- 初回のテストには、CEDF トランザクションを使用します。CEDF の使用についての詳細は、175 ページの『第 11 章 実行診断機能 (EDF)』を参照してください。
- ストレス・テストまたはボリューム・テストを行って、シングルユーザー環境では起こらない可能性がある問題を見つけてください。通信網シミュレーター (TPNS) (ライセンス・プログラム番号 5740-XT4) は、このテストを行うために優れたツールです。

  TPNS は、アプリケーションのインストール前に、そのテストと評価を可能にする通信テスト・パッケージです。組織内で、アプリケーション・プログラムが論理、ユーザー出口ルーチン、メッセージ・ロギング、データ暗号化、および装置依存要素を使用している場合には、TPNS を使用して、これらをテストすることができます。TPNS は、システム・パフォーマンスと応答時間、ストレス・テストの検査、および TP ネットワーク設計の評価を行う場合に有用です。詳しくは、「*TPNS General Information*」マニュアルを参照してください。
- アプリケーションで、***正しい*** データおよび***無効な*** データを処理できるかどうかをテストします。
- 関連するデータベースの完全なコピーに対してテストします。
- 複数領域操作の使用を考慮します。(詳細については、「*CICS 相互通信ガイド*」の複数領域操作を参照してください。)
- アプリケーションを実動システムに移す前に、実動データベースのコピーを使って***最終*** テストを実施して、エラーがないか調べるのはいい方法です。

特に、破棄されたストレージのチェーンを検索してください。

アセンブラー言語プログラムは (データ域のアドレッシングが正しくない場合)、プログラムが変更したものによって、別のトランザクションに影響を与えて (異常終了させて) いる可能性があるので、識別するのはさらに難しくなります。

問題の解決の詳細については、「*CICS Problem Determination Guide*」の問題判別に対するアプローチを参照してください。

端末を持たないトランザクションのセキュリティー

CICS は、未許可の使用に対し、端末とは関係のないトランザクションで使用されるリソースを保護することができます。このようなトランザクションには、以下の 3 タイプがあります。

- START コマンドによって開始され、端末 ID を指定していないトランザクション。
- 区画内一時データ・キューでトリガー・レベルに達した結果として、端末なしで開始されたトランザクション。
- プログラム・リスト・テーブル (PLT) で指定されたプログラムを実行する CICS 内部トランザクション (CPLT)。これは、CICS 始動処理中に稼働するものです。このトランザクションは、第 1 フェーズおよび第 2 フェーズの両方の PLT を実行します。

また、CICS シャットダウン処理中に実行される PLT プログラムに対するリソース保護検査を、実行することもできます。シャットダウン PLT シャットダウン・プロ

グラムは、シャットダウンを要求するトランザクションの一部として実行されるので、シャットダウン・コマンドを発行したユーザーの権限の下で実行されます。

START コマンドは、それによって開始される、端末を持たないトランザクションに対するセキュリティーを扱います。

別のユーザーのためにトランザクションの生成が許可されているか、トランザクションが生成されるようにするか、そのトランザクションのリソース・アクセス権限をすべて継承しているサロゲート・ユーザーは、本物のユーザーに代わって活動することができます。

CICS は、その環境によって、単一の START コマンドについてのサロゲート・ユーザー・セキュリティー検査を、3 つまで発行することができます。

1. USERID が指定されている場合には、START コマンドを発行するトランザクションのユーザー ID。
2. START コマンドを発行するトランザクションが CEDF 二重画面モードで実行中である場合には、CEDF トランザクションのユーザー ID。
3. START コマンドが別の CICS システムに機能伝送され、リンク・セキュリティーが有効である場合には、リモート・システムの CICS 領域ユーザー ID。

個別のサロゲート・ユーザー・セキュリティー検査は、トランザクションが生成される前に、必要に応じて、ユーザー ID ごとに行われます。

USERID オプション、USERIDERR 条件、INVREQ 条件、および NOTAUTH 条件のプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

第 18 章 パフォーマンスの設計

このセクションでは、アプリケーション・プログラムそのものを大幅に変更することなくパフォーマンスと効率を改良できる、設計上の変更点について説明します。

- 『プログラム・サイズ』
- 272 ページの『仮想記憶域』
- 276 ページの『リソースの排他制御』
- 277 ページの『操作のコントロール』
- 278 ページの『オペレーティング・システム待機』
- 278 ページの『NOSUSPEND オプション』
- 279 ページの『効果的な順次データ・セットのアクセス』
- 280 ページの『効率的なロギング』

アプリケーション設計のその他の局面については、241 ページの『第 17 章 アプリケーション設計』で取り上げています。

特定の状態で適用されるパフォーマンス上の問題がある場合には、変更の効果がその状況だけに適用されるように、内容を切り分けて変更するようにしてください。問題を修正し、変更内容をテストした後で、それを最も共通して使用されるプログラムおよびトランザクションで使用すると、パフォーマンスへの効果が最も著しく現れます。

プログラム・サイズ

以前は、小さなプログラムに重点がおかれ、CICS プログラマーは、プログラムをできるだけ小さな単位に分割し、それらの間で XCTL コマンドを使用して制御権を移動したり、LINK コマンドを使用してリンクするようにしていました。

しかし、現在のシステムでは、プログラムを小さな単位に分割することが必ずしもいい方法というわけではありません。これは、制御権の移動や LINK コマンドのたびに CICS 処理のオーバーヘッドがあり、レジスター保管域 (RSA) のストレージ・オーバーヘッドも生じるためです。

コードのブロック・サイズがそれほど大きくなく、順次処理される場合は、インライン・コードが最も効率的です。この規則に対する例外は、次のようなコードのブロックです。

- かなり長く、アプリケーションの複数の異なる時点で独立して使用されるコード
- 頻繁に変更されるコード (この場合は、LINK コマンドまたは XCTL コマンドのオーバーヘッドと保守の容易さとを比較検討してください)
- 使用頻度の少ないコード、例えば、エラー・リカバリー・ロジックや一般的でないデータの組み合わせを処理するコードなど

上記のいずれかの理由でサブルーチンとして書く必要があるコードのブロックがある場合には、パフォーマンスの観点からこれを取り扱う最善の方法は、呼び出し側

© Copyright IBM Corp. 1989, 2009

プログラム内で閉じたサブルーチン (例えば、COBOL で PERFORM コマンドによって処理するコード) を使用することです。このサブルーチンが他のプログラムにも必要な場合には、サブルーチンを独立したプログラムとする必要があります。独立したプログラムは CALL ステートメント (マクロ) を使用して呼び出すことができるか、あるいは独立したままで保持し、XCTL コマンドまたは LINK コマンドを使用して処理することができます。CALL の実行オーバーヘッドは、CICS サービスが呼び出されないために、小さくなります。例えば、呼び出されているプログラムの作業用ストレージは、コピー***されません***。しかし、呼び出されるプログラムは呼び出し側プログラムにリンクする必要があるので、COBOL、C、または PL/I の特殊機能を使用しない限り、呼び出されるプログラムを必要とする他のプログラムと共用することはできません。呼び出されるサブルーチンはそれを呼び出す各プログラムの一部としてロードされ、そのためにさらに多くのストレージを使用します。したがって、このプログラムを使用する後続のトランザクションは、呼び出されるプログラムに対して行われる変更内容を作業用ストレージに持っていることも、持っていないこともあります。これは、CICS がそのプログラムの新規コピーをストレージにロードしたかどうかによって、まったく異なります。

XCTL および LINK コマンドを使用すると、オーバーヘッドが (しかしまた柔軟性も) 最も高くなります。XCTL コマンドより LINK コマンドの方が、プロセッサーおよびストレージの両方の要件がはるかに大きくなります。したがって、呼び出されるプログラムの処理が終わったあとに、呼び出し側プログラムが制御を受け取る必要がない場合には、XCTL コマンドを使用してください。

アプリケーション・プログラムからのロード・モジュールは、最大 2GB までの主記憶装置を占有することができます。明らかに、非常に大きなロード・モジュールのロードおよび初期化には、余分なコストがかかります。そこで、CICS 動的ストレージの限界 (EDSA) を、それに応じて高く設定する必要があります。大きなロード・モジュールの使用は、できれば避けてください。しかし、C++ のようなオブジェクト指向言語で作成した大規模アプリケーションは、サイズが 16M を楽に超えてしまうこともあります。C++ のクラスを単一の DLL にバインドすると、その単一 DLL が複数の DLL に再編成された場合に、クラスのパフォーマンスが低下します。これは、複数の DLL 間の関数参照を解決する際に必要な処理のためです。

プログラムが動的ストレージ域 (DSA) 内で利用可能なすべてのストレージ域を占有した場合には、異常終了コード APCG が発生することがあります。

仮想記憶域

真の会話型 CICS タスクといえるのは、いくつかの対話またはたくさんの対話の間に、各端末書き込みの後で端末読み取り要求を発行する (例えば、SEND コマンドと RECEIVE コマンドを続けて使用するか、CONVERSE コマンドを使用する) ことによって端末ユーザーと会話するタスクのことです。これは、タスクの持続時間のほとんどが端末ユーザーからの次の入力を待つために費やされていることを意味します。

すべての CICS タスクはその存続期間において仮想記憶域を必要とし、会話型タスクにおいてはそのタスクが端末 I/O を待機する場合に、この仮想記憶域の一部がその期間を超えて持ち越されます。持ち越されるストレージ域には、TCA および関連するタスク制御ブロック (EIS または EIB など)、および任意の端末読み取り要求が

発行されたときに使用中であるすべてのプログラムに必要なストレージ域が含まれます。また、このタスクによるそれらのプログラムの使用と関連した作業域 (COBOL 作業用ストレージのコピーなど) も含まれます。

注意深い設計によって、会話の期間を通じて保存される、非常に小さいプログラムを 1 つだけ配置することができることもあります。必要なストレージは、他のユーザーと共用することができます。残りの仮想記憶域の所要量には、そのコードを使用する並行会話型セッションの数を乗ずる必要があります。

これに対して、疑似会話型タスクのシーケンスは、メッセージの対の処理に費やされる期間だけ、その仮想記憶域のほとんどすべてを必要とします。一般に、これには毎分 1 から 3 秒かかります (残りはオペレーターの入力を待つ時間です)。したがって、複数の並行ユーザー全体の所要量は、会話型タスクに必要な所要量の 5% 程度です。しかし、各タスクから次のタスクに渡されるデータ域の分を考慮に入れて追加しておく必要があります。これは数バイトの COMMAREA のこともあれば、一時記憶域の大きな区域のこともあります。後者の場合には、通常、主記憶装置の中ではなくディスク上の一時記憶域を使用することをお勧めしますが、これは疑似会話型のセットアップで、会話型処理には必要のない、余分な一時記憶域 I/O オーバーヘッドが増えることを意味します。

会話型アプリケーションに余分な仮想記憶域が必要であるということは、通常、それに比例した大きさの実記憶域が必要になることを意味します。ストレージを制御するために必要なページングによって、追加のオーバーヘッドおよび仮想記憶域がもたらされます。ページングの不利益面の影響は、トランザクション率が高まるにつれて増加するので、ページングの使用はできる限り最小化する必要があります。この点の詳細については、『ページングの影響の軽減』を参照してください。

ページングの影響の軽減

このタスクについて

ページングの影響の軽減は、CICS が仮想記憶域環境で使用する手法です。この環境でのプログラミングの主目的はページ不在の削減です。ページ不在が起こるのは、プログラムが実記憶域内に常駐していない命令、またはデータを参照する場合です。この場合には、参照された命令またはデータが入っている仮想記憶域内のページを、実記憶域にページインする必要があります。多くのページングが必要になると、システム全体のパフォーマンスが低下します。

アプリケーション・プログラムはオペレーティング・システムと直接連絡することもできますが、そのような処置を取った結果は予測できないばかりでなく、パフォーマンスを低下させることになります。

仮想記憶域環境で動くアプリケーション・プログラムを作成するに当たっては、次の用語を理解しておく必要があります。

参照の局所性
: アプリケーション・プログラムの実行時の、比較的長い期間の、比較的少数 (プログラム内の合計ページ数との比較で) のページ内の、命令およびデータへの一貫した参照。

作業セット
: 一定の期間中に必要となるプログラムのページの数とその組み合わせ。

参照セット

不要なデータを検索する中間ストレージ参照を使用しないで、必要なページを直接参照すること。

参照の局所性

このタスクについて

プログラムで処理される命令および使用されるデータを、比較的少数のページ (4096 バイト・セグメント) 内に保持します。プログラムにおけるこの特性は、「参照の局所性」と呼ばれます。これは、次のように行うことができます。

- プログラムはできるだけ直線的に実行する。
- 通常の実行シーケンスで使用するサブルーチンは、サブルーチンを呼び出すコードのできるだけ近くに置く。
- 数個所からしか呼び出されない短いサブルーチンがある場合には、繰り返しになっても、コードをインライン化する。
- エラー処理コード、および頻繁には処理されないその他のコードは、プログラムの本体から切り離す。
- 上記のコードが使用するデータは、通常の実行で使用するデータと分ける。
- データ項目 (特に、配列およびその他の大きな構造) は、参照する順に定義する。
- データ構造内のエレメントを、おおよそそれが参照される順序にしたがって定義する。例えば、PL/I では、1 行のすべてのエレメントを保管してから、次の行に移り、以下同様に保管されます。配列は、列ごとではなく行ごとに処理できるように定義する必要があります。
- データは、最初に使用する場所のできるだけ近くで初期設定する。
- COBOL 変数の MOVE 操作は、サブルーチン呼び出しに展開されるので使用を避ける。
- GETMAIN コマンドの発行をできるだけ少なくする。一般に、必要な期間が著しく変化しない限り、少ない所要量の GETMAIN コマンドを何回も実行するより、その所要量を合計して 1 回実行する方がプログラムには良好です。
- 可能な場合、GETMAIN コマンドでの INITIMG オプションの使用を避ける。これは、獲得されているストレージへの即時ページ参照をともないます。これは、プログラムではるかに後になってから同じ区域へのその他の参照があるまで、他では起こることがありません。

注: ご使用のシステムのプログラミング標準が、仮想記憶域環境での実行速度より、むしろコードの読みやすさおよび保守容易性を目的としている場合には、上記の示唆の中には、その標準と対立するものがある可能性があります。一部の構造化プログラミング方式、特に、モジュラー・プログラミング手法は、プログラムの構造を明瞭にするために、COBOL における PERFORM (C、PL/I、およびアセンブラー言語ではそれと同等のもの) を多用します。これもまた、非構造化プログラムで見付かるものより多くの、順次処理への例外となります。それにもかかわらず、構造化されたコードと関連した生産性は、参照の局所性が損なわれる可能性よりはるかに大きな価値がある場合があります。

作業セット

このタスクについて

作業セットは、一定の期間中に必要となるプログラムのページの数とその組み合わせです。作業セットのサイズを最小に抑えるには、プログラムが一定期間に参照するストレージの量を、できるだけ小さくすることです。これは、次のように行うことができます。

- モジュラー・プログラムを書き、参照の頻度および参照の予想時点に従ってモジュールを構成する。単にサイズの問題でモジュール化してはなりません。サブルーチンまたは別個のモジュールにするのではなく、コードが重複することになっても、インラインにすることを検討してください。
- プログラムの流れから見て順次に実行されないような場合は必ず、独立したサブプログラムを使用する。
- 端末ユーザーからの応答を待っている主記憶装置を占有しない。
- 移動モードではなく、コマンド・レベル・ファイル制御位置指定モードの入出力を使用する。
- COBOL プログラムでは、定数は、WORKING STORAGE セクション内のデータ変数としてではなく、PROCEDURE DIVISION 内のリテラルとして指定する。
- C、C++、および PL/I プログラムでは、定数データに静的ストレージを使用する。
- LINK コマンドは、主記憶装置を要求することになるので、使用をできるだけ避ける。

参照セット

このタスクについて

プログラムが通常の操作中に使用するページ総数は、できる限り少なくするようにしてください。これらのページは**参照セット**と呼ばれ、プログラムの実記憶域所要量を示します。参照セットは次のように減らすことができます。

- COBOL プログラムでは、定数は、WORKING STORAGE SECTION のデータ変数としてではなく、PROCEDURE DIVISION のリテラルとして指定する。この理由は、リテラルはプログラムそのものの一部と見なされ、CICS においてコピーが 1 つしか使用されないのに対して、作業用ストレージはプログラムを実行中のタスクごとに個別のコピーがあるためです。
- C、C++、および PL/I では前述と同じ理由で、純粋に定数であるデータに対し静的ストレージを使用する。
- プログラム内のデータ域はできるだけ再利用する。これは、COBOL では REDEFINES 文節、C および C++ では共用体文節、PL/I では基本ストレージ、さらにアセンブラー言語では、ORG またはそれと同様のものを使用して、行うことができます。特に、一度に 1 つしかマップを使用しないマップ・セットがある場合には、STORAGE=AUTO または BASE のいずれのオペランドも指定しないで、DFHMSD マップ・セットの定義をコーディングしてください。これにより、マップ・セット内のマップを別のマップ・セットに再定義することができます。

データは、次のようにして直接参照します。

- テーブルのデータの長時間探索を避ける。

- 直接アドレッシング可能な配列などのデータ構造を使用し、探索が必要となるチェーンなどの構造は使用しない。
- 間接アドレッシングをシミュレートする方式を避ける。

アプリケーション・プログラムでは、オーバーレイ (ページング手法) を使用しないようにしてください。システム・ページングが自動的に提供され、このページングのパフォーマンスは優れています。仮想記憶域環境用のアプリケーション・プログラムの設計は、実記憶域環境用の設計に似ています。システムは、参照されないページ上のコードをページインする必要がなくなるように、すべてのモジュールを常駐させる必要があります。

プログラムが動的になっている場合には、実行を始める前に、隣接するページにまたがるプログラム全体をロードする必要があります。動的プログラムが使用中でなく、満たされないストレージ要求が存在する場合に、動的プログラムがストレージから除去されることがあります。動的プログラムはページ上の未使用スペースを他のプログラムと共用しないので、動的区域を十分に使用できるようにしてプログラムが除去されないようにしておくことは、そのプログラムを常駐させておくことより不経済です。

リソースの排他制御

CICS が提供する、非常に基本的かつ強力なリカバリー機能は、パフォーマンスに影響を与えます。

CICS はリカバリー可能リソースへの更新を逐次化するので、トランザクションが失敗した場合には、リカバリー可能リソースへの更新は、他のトランザクションによって行われる変更とは独立しており、バックアウトすることができます。したがって、リカバリー可能リソースを更新するトランザクションは、それが終了するか、あるいは SYNCPOINT コマンドを使用して変更内容をコミットしたいことを指示するまで、そのリソースの制御を獲得しています。同一リソースを必要としている他のトランザクションは、最初のトランザクションがそれを使い終えるまで待つ必要があります。

このようなロック遅延を生成する 1 次リソースは、データ・セット、DL/I データベース、一時記憶域、および一時データ・キューです。保護の基礎となる単位は、データ・セットの場合は個別レコード (キー)、DL/I データベースの場合はプログラム仕様ブロック (PSB)、さらに一時ストレージの場合はキュー名です。一時データの場合には、キューの「読み取り」終了が、「書き込み」終了とは別個のリソースと見なされます (すなわち、あるトランザクションがキューに書き込んでいる間に、別のトランザクションがそこから読み取ることができます)。

リソース所有権に関する競合からのトランザクション遅延を削減するためには、リソースの請求とリソースの解放 (UOW の終了) の間の時間の長さを最小化する必要があります。特に、会話型トランザクションが端末読み取りにまたがってクリティカル・リソースを所有してはいけません。

注: 非リカバリー可能データ・セットの場合でも、VSAM は、2 つのトランザクションが同時に同じレコードを更新のために読み取らないようにします。しかし、このエンキューは、UOW の終了時ではなく、更新が完了するとすぐに終了します。

BDAM データ・セットに関するこの保護でも、ファイル管理テーブルに「排他制御なし」(SERVREQ=NOEXCTL) と定義すると解放することができます。

特定の規則を守らない限り、この保護スキーマは、遅延の他にデッドロックを起こすことがあります。2 つのトランザクションが複数のリカバリー可能リソースを更新する場合には、常に同じ順序でリソースを更新する必要があります。例えば、それぞれが 2 つのデータ・セットを更新する場合には、すべてのトランザクションで、データ・セット「A」はデータ・セット「B」の前に更新する必要があります。同様に、トランザクションが単一データ・セット内の複数レコードを更新する場合には、常になんらかの予測可能な順序 (低いキーから高いキー、またはその逆) で更新する必要があります。READ UPDATE コマンドを使用するたびに TOKEN キーワードを組み込むことを検討するのもいいことです。TOKEN キーワードについての詳細は、577 ページの『TOKEN オプション』を参照してください。一時データ、一時ストレージ、およびユーザー・ジャーナルは、そのようなリソースに含まれている必要があります。「*CICS Recovery and Restart Guide*」のアプリケーション・プログラム内のリソースのロックには、リソース保護の範囲に関する詳細情報が示されています。

ここでは、VSAM 制御インターバルに対する CICS データ・セットと、データ・セットに対する VSAM 内部ロックとの相違点に注目するのが適切です。 CICS は VSAM エンキューについての情報を持っていないので、バッチと CICS の両方から同時に更新される SHARE OPTION 4 の制御インターバルを使用すると、最善の場合でもパフォーマンスが低下し、さらに最悪の場合には、バッチと CICS の間で予期しないデッドロック状態になることがあります。このようなバッチと CICS の間の同時更新は避ける必要があります。どの場合でも、データ・セットがバッチおよび CICS の両方によって更新されると、CICS はデータ保全性を確保できません。

操作のコントロール

以下の操作手法を使用すると、CICS システムのパフォーマンスと効率を改善することができます。

MXT

CICS システムの初期設定パラメーター MXT は、CICS システムに同時に存在できるユーザー・タスクの最大数を指定します。MXT は、CICS システムにおいて、ストレージ不足 (SOS) 状況を回避するため、およびリソースに対する競合を制御するために非常に重要です。このパラメーターは、CICS システム内にあるアクティビティーが既に多過ぎる場合に、入力メッセージを処理するユーザー・タスクの作成を遅らせます。特に、処理を待っているメッセージが占有する仮想記憶域は、通常、メッセージを処理するタスクに必要な仮想記憶域よりはるかに少ないので、速やかに処理できるようになるまで、メッセージの処理を延期することによって仮想記憶域が節約されます。

トランザクション・クラスは、ユーザー定義の特定のタイプ、またはクラスのタスクの数を制限するのに有用です。特に、これらが大量のリソースを使用する場合に有用です。

ランナウェイ・タスク

タスクが中断したとき、CICS はタスクのランナウェイ・タイム (ICVR) だけをリセットします。 CICS の各実行方法が固有の性質を持っているため、EXEC

CICS コマンドが、処理中にタスクを中断させることは保証できません。ランナウェイ・タイムが超過して、タスクを異常終了 AICA にすることがあります。この異常終了は、アプリケーションに EXEC CICS SUSPEND コマンドをコーディングすることによって防止できます。これにより、ディスパッチャーが、要求を発行したタスクを中断し、より高い優先順位を持つタスクを実行することができるようになります。実行する準備ができているタスクがない場合は、中断を発行したプログラムが再開されます。異常終了 AICA の詳細については、「*CICS Problem Determination Guide*」のCICS によって検出されないループの調査を参照してください。

補助トレース

補助トレースを使用してユーザー・アプリケーション・プログラムを検討します。例えば、SETL の後ではなくデータ・セットの先頭からのデータ・セット・ブラウズ、回数が多すぎるかあるいは量が多すぎる GETMAIN コマンド、もはや必要なくなった時にストレージを解放する障害、意図していない論理ループ、およびもはや必要ない排他制御用に保留されているレコードのアンロックの障害などの、明らかに不要なコーディングをはっきりさせることができます。

オペレーティング・システム待機

オペレーティング・システムが待機する原因になるような機能の使用は避ける必要があります。これらの待機の 1 つが起こった場合には、すべての CICS アクティビティーが停止し、すべてのトランザクションに応答遅延が生じます。このような待機の主な原因には、次のものがあります。

- 区画外一時データ・セット (279 ページの『効果的な順次データ・セットのアクセス』を参照してください)。
- COBOL、C、C++、および PL/I の各言語機能のうち、CICS で使用できない機能、および一般に CICS が代替機能を提供している機能。言語の制約事項についての詳細は、25 ページの『第 3 章 COBOL でのプログラミング』、53 ページの『第 4 章 C および C++ でのプログラミング』、および 65 ページの『第 5 章 PL/I でのプログラミング』を参照してください。
- オペレーター宛メッセージ (WTO) など、オペレーティング・システム・サービスを呼び出す SVC およびアセンブラー言語マクロ。

NOSUSPEND オプション

ENQBUSY、NOJBUFSP、NOSPACE、NOSTG、QBUSY、SESSBUSY、および SYSBUSY 条件のデフォルトのアクションは、必要なリソース (例えばストレージ) が利用可能になってコマンドの処理が再開されるまで、アプリケーションの実行を延期することです。これらの条件を発生させることがあるコマンドは ALLOCATE、ENQ、GETMAIN、WRITE JOURNALNAME、WRITE JOURNALNUM、READQ TD、および WRITEQ TS です。

上記のコマンドでは、NOSUSPEND オプション (ALLOCATE コマンドの場合は NOQUEUE オプション) を使用して、このような待ちを禁止し、アプリケーション・プログラムのコマンドの後に続く命令に、即時に戻るようにすることができます。

CICS は、COBOL アプリケーション・プログラムにおける HANDLE CONDITION、および IGNORE CONDITION コマンドによって参照される条件のテーブルを保守します。

制約事項: HANDLE CONDITION コマンドおよび IGNORE CONDITION コマンドは、C および C++ プログラムではサポートされていません。
条件がまだそのようなコマンドの対象でない場合は、これらのコマンドを実行すると、既存の項目を更新するか、または新規の項目を発生させます。各項目は、下記の 3 つの状態のいずれかを示します。

- ラベルは現在、次のように指定されています。

  ```
  HANDLE CONDITION condition(label)
  ```

- 条件は、次のように無視されます。

  ```
  IGNORE CONDITION
  ```

- 現在、次のようにどのラベルも指定されていません。

  ```
  HANDLE CONDITION
  ```

条件が発生すると、次のテストが行われます。

1. コマンドに NOHANDLE オプションまたは RESP オプションがある場合、アプリケーション・プログラムの次の命令に制御権を返します。そうでない場合には、処置をどうするかを決めるために条件テーブルがスキャンされます。
2. 条件についての項目が存在する場合には、その項目によって処置が決まります。
3. 項目が存在せず、その状態に関するデフォルトのアクションが実行の中断である場合は、次のように処理されます。
   - コマンドに NOSUSPEND オプションまたは NOQUEUE オプションが指定されている場合には、制御は次の命令に返される。
   - コマンドにこれらのオプションのいずれも指定されていない場合には、タスクが中断される。
4. 条件についての項目が存在せず、デフォルトのアクションが異常終了させることであれば、ERROR 条件を探す二度目の探索が行われ、次のように処理されます。
   - ERROR が見つかった場合は、この項目によって処置が判別される。
   - ERROR が探索され、検出されなかった場合は、タスクは異常終了する。

効果的な順次データ・セットのアクセス

CICS は順次処理オプションをいくつか提供します。一時記憶域および区画内一時データ・キュー (既に、 263 ページの『一時記憶域』および 265 ページの『区画内一時データ』で説明しました) が最も効率的な使用ですが、それらはすべて CICS 内で作成および処理しなければなりません。

区画外一時データは、標準順次 (QSAM/BSAM) データ・セットを処理する CICS の手段です。区画外一時データは、BDAM を処理するときと同様に、CICS がオペレーティング・システム待機を発行してデータ・セットを処理する必要があるので、リストされている順次サポートの 3 つの形式の中では最も効率が悪くなります。そのうえ、区画外一時データ・セットはリカバリー可能ではありません。一方、VSAM ESDS は制限付きでリカバリー可能で、処理はもっと効率的です。リカ

バリーの制限は、未完了の UOW の間に ESDS に追加されるレコードは、VSAM の制約事項のために、バックアウト処理中に物理的に除去することはできないという点です。しかし、このレコードにはユーザー出口ルーチンによって削除済みのフラグを付けることができます。

出力データ・セットに対してだけですが、CICS ジャーナルは、区画外一時データに代わる良好な手段を提供します。ジャーナルは MVS システム・ロガーによって管理されますが、柔軟性のある処理オプションによって非常に効率的な処理を行うことができます。区画外操作は、一時データ・キュー定義内のパラメーターによって全体が制御されるのに対して、各ジャーナル・コマンドは、例えば、同期または非同期などの操作特性を指定します。

可能な場合には、ジャーナリングに関するタスク待機を最小に抑えるために、トランザクションは非同期にジャーナル処理する必要があります。しかし、保全性の考慮事項として、タスクを終了する前にジャーナル・レコードを物理的に書き込むことが必要な場合には、同期書き込みを使用する必要があります。ジャーナル書き込みが複数ある場合には、タスクの論理レコードが最小数の物理 I/O および 1 回の待機で書き込まれるように、トランザクションは、最後の論理レコードを除いたすべてに非同期書き込みを使用する必要があります。

ジャーナルは、CICS が実行中、出力 (オンラインで) だけでなく入力 (バッチで) に対しても使用できます。提供されているバッチ・ユーティリティー DFHJUP は、例えば、印刷またはコピーによってジャーナル・データにアクセスするために使用できます。バッチ内でのジャーナルの読み取りには、次の制約事項があることに注意してください。

- MVS システム・ロガーのログ・ストリーム・データへのアクセスは、サブシステム・インターフェース LOGR を通して提供されます。
- ジャーナルからのレコード読み取りは、バッチ・ジョブ方法によってのみ、オフラインで可能です。

効率的なロギング

CICS は、データ・セットに対する一部のタイプまたはすべてのタイプのアクティビティーのログをとるための、オプションを提供します。必要な場合には、ロギング更新によって、バックアップ・コピーからデータ・セットを再構成することができます。また、セキュリティー上の理由から読み取りのログをとりたい場合もあります。データ保全性およびセキュリティーの必要性と、ロギングによるパフォーマンスへの影響のバランスをとる必要があります。これらは、ロギングするために必要な実際の操作であり、ロギングが暗黙指定する排他制御によって引き起こされることがある遅延です。

第 19 章 トランザクション間のデータの共用

トランザクション間でデータを共用するための CICS 機能は、次のとおりです。

- 共通作業域 (CWA)
- TCTTE ユーザー域 (TCTUA)
- COMMAREA
- 表示画面
- チャネルおよびコンテナー

TCTUA および CWA に保管されているデータは、システム内のどのトランザクションでも利用できます。リソース保護およびストレージ保護の制約事項はあるものの、すべてのトランザクションがそこに書き込むことができ、すべてのトランザクションがそこから読み取ることができます。

これらの機能のいくつかを使用すると、トランザクション間に類縁性が生じる場合があります。トランザクションの類縁性の詳細については、339 ページの『第 22 章 類縁性』を参照してください。

このセクションでは、以下について説明します。

- 『共通作業域 (CWA) の使用』
- 285 ページの『TCTTE ユーザー域 (TCTUA) の使用』
- 286 ページの『RETURN コマンドでの COMMAREA の使用』
- 286 ページの『RETURN コマンドにおけるチャネルの使用』
- 287 ページの『データを共用する表示画面の使用』

共通作業域 (CWA) の使用

共通作業域 (CWA) は、システムの起動時に割り振られ、その CICS セッションの間存在する単一の制御ブロックです。サイズは、システム初期設定パラメーター WRKAREA に指定されているとおりに固定されています。CWA は、次の特性を持っています。

- CWA へのデータの保管または CWA からのデータの検索では、ほとんどオーバーヘッドがありません。コマンド・レベル・プログラムは、1 つの ADDRESS コマンドを発行して区域のアドレスを入手する必要があり、その後でその区域を直接アクセスすることができます。
- トランザクションまたはシステムが障害を起こすと、CWA のデータはリカバリーされません。
- リソース保護は適用されません。
- CICS は CWA の使用を規制しません。CWA を使用するすべてのアプリケーション内のすべてのプログラムは、共用使用の場合と同じ規則に従う必要があります。通常、これらはアプリケーションの開発者と協力してシステム・プログラマーが設定し、区域のレイアウトを記述するために同一の「コピー」モジュールを使用する、すべてのプログラムに必要です。

© Copyright IBM Corp. 1989, 2009

CWA の長さを超えると記憶保護違反になるので、これが起こらないようにしてください。また、1 つのトランザクションで使用するデータが、他のトランザクションで使用するデータを上書きしないように、十分に注意してください。CWA データを保護する 1 つの方法は、ユーザー・キー・アプリケーションによる CWA への書き込みを防止する、ストレージ保護機能を使用することです。詳しくは、『CWA の保護』を参照してください。

- CWA は、アプリケーション・プログラムの複数のプログラムによって、頻繁に読み取りまたは更新が行われる少量のデータ (状況情報など) に、特に適しています。
- CWA は常に割り振られるので、大量のデータまたは存続期間の短いデータには適していません。

CWA の保護

CWAKEY システム初期設定パラメーターを使用して、CICS キー・ストレージまたはユーザー・キー・ストレージのどちらから CWA を割り振るのかを、指定することができます。CWAKEY パラメーターについて詳しくは、「*CICS System Definition Guide*」のCWAKEY パラメーターを参照してください。

CWA に対する書き込みアクセスを制限したい場合は、CWAKEY=CICS を指定することができます。つまり、CICS は CICS キー・ストレージから CWA を割り振り、EXECKEY(USER) で定義されたアプリケーション・プログラムの CWA に対するアクセスを、読み取り専用アクセスに制限するということです。CICS キー・ストレージから割り振られた CWA に書き込み可能なプログラムは、EXECKEY(CICS) で定義されたものだけです。

CICS キーで実行されるプログラムはどれも CICS ストレージに書き込み可能なので、それらのプログラムを十分にテストし、CICS ストレージを上書きしないことを確認する必要があります。

CWA より CICS 保護を優先したい場合は、CWA に対して CWAKEY=USER を指定し、CWA を上書きするすべてのプログラムに対して EXECKEY(USER) を指定してください。これにより、プログラムが CWA の長さを超えた場合でも、プログラムが CICS ストレージを上書きしないようにします。ストレージ保護の詳細については、653 ページの『ストレージ制御』を参照してください。

283 ページの図 63 は、CWA の特定の使用方法を示すものです。ここでは CWA そのものが、CWAKEY=CICS により、ユーザー・キー・アプリケーション・プログラムから保護されています。

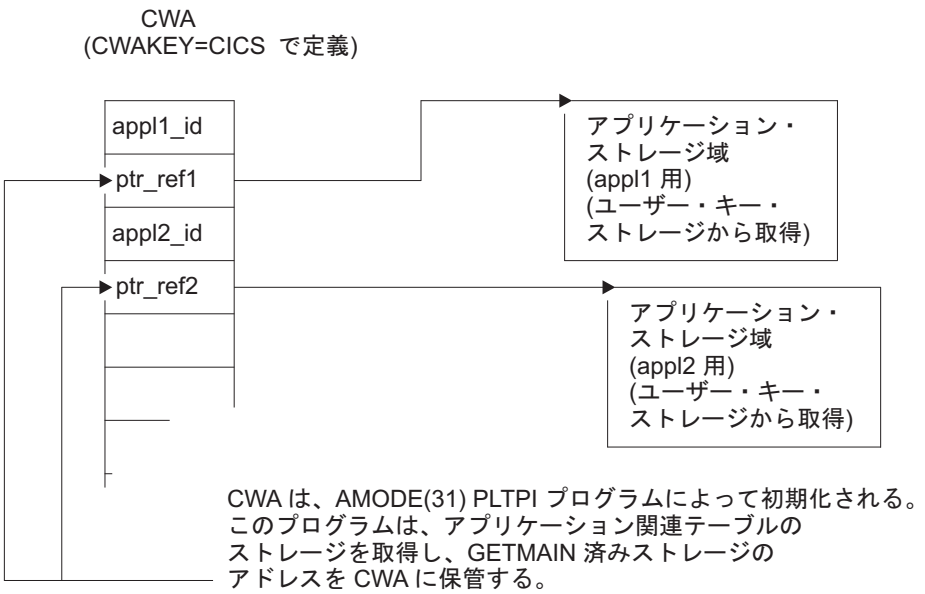


図 63. CICS キー・ストレージにおける CWA の使用例: この図は、ユーザー・キーまたは CICS キー・ストレージにおいて、アプリケーション・プログラムが使用するために確保されているストレージを参照するための、CWA の使用方法を示しています。CWA 自体は、CICS キー・ストレージに入れられているため、保護されています。

この図では、CWA はアプリケーション・データおよび定数の保管に、直接には使用されていません。CWA には、対になったアプリケーション ID、およびアプリケーションに関連したアドレスがあり、アプリケーションに関連したデータを保持するデータ域のアドレスが入った、アドレス・フィールドがあります。保護のために CWA は CWAKEY=CICS で定義されています。したがって、この図においてプログラム・リスト・テーブル事後初期化 (PLTPI) リストで定義され、CWA にアドレスとアプリケーション ID をロードするプログラムは、EXECKEY(CICS) で定義されなければなりません。CWA にアクセスする必要のあるすべてのアプリケーションは、EXECKEY(USER) で定義されているので、それをもとに、CWA がアプリケーション・プログラムで上書きされないように保護されているか、確認してください。図 63 では、データ域の 1 つが CICS キー・ストレージから獲得され、他のデータ域はユーザー・キー・ストレージから獲得されます。

284 ページの図 64 に示すサンプル・コードにおいて、プログラム・リスト・テーブル事後初期化 (PLTPI) プログラムは、CWA に保管されるデータに対するポインターを含む、アプリケーション・データ域を設定します。

このサンプルは、アプリケーション・プログラムが使用するグローバル・データの作成方法を、CWA に保管されるデータのアドレスとともに (例えば PLTPI プログラムによって) 示しています。最初のデータ域は、PLTPI プログラムが発行する GETMAIN コマンドのデフォルトにより CICS キー・ストレージから獲得され、2 番目のデータ域は USERDATAKEY オプションを指定することによりユーザー・キー・ストレージから獲得されます。CWA 自体は CICS キー・ストレージにあり、PLTPROG は EXECKEY(CICS) により定義されます。

```
ID DIVISION.
PROGRAM-ID. PLTPROG.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
77  APPLID                  PIC X(8)     VALUE SPACES.
77  SYSID                   PIC X(4)     VALUE SPACES.
01  COMM-DATA.
    03  AREA-PTR            USAGE IS POINTER.
    03  AREA-LENGTH         PIC S9(8)   COMP.
LINKAGE SECTION.
01  COMMON-WORK-AREA.
    03  APPL-1-ID           PIC X(4).
    03  APPL-1-PTR          USAGE IS POINTER.
    03  APPL-2-ID           PIC X(4).
    03  APPL-2-PTR          USAGE IS POINTER.
PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.
*  Obtain APPLID and SYSID values
    EXEC CICS ASSIGN APPLID(APPLID)
                     SYSID(SYSID)
    END-EXEC.
* Set up addressability to the CWA
    EXEC CICS ADDRESS
              CWA(ADDRESS OF COMMON-WORK-AREA)
    END-EXEC.
* Get 12KB of CICS-key storage for the first application ('APP1')
    MOVE 12288     TO   AREA-LENGTH.
    EXEC CICS GETMAIN  SET(AREA-PTR)
              FLENGTH(AREA-LENGTH)
              SHARED
    END-EXEC.
* Initialize CWA fields and link to load program
* for storage area 1.
    MOVE 'APP1'      TO  APPL-1-ID.
    SET APPL-1-PTR   TO  AREA-PTR.
    EXEC CICS LINK PROGRAM('LOADTAB1')
              COMMAREA(COMM-DATA)
    END-EXEC.
* Get 2KB of user-key storage for the second application ('APP2')
    MOVE 2048      TO   AREA-LENGTH.
    EXEC CICS GETMAIN SET(AREA-PTR)
              FLENGTH(AREA-LENGTH)
              SHARED
              USERDATAKEY
    END-EXEC.
* Initialize CWA fields and link to load program
* for storage area 2.
    MOVE 'APP2'    TO  APPL-2-ID.
    SET APPL-2-PTR TO  AREA-PTR.
    EXEC CICS LINK PROGRAM('LOADTAB2')
              COMMAREA(COMM-DATA)
    END-EXEC.
    EXEC CICS RETURN
    END-EXEC.
  MAIN-PROCESSING-EXIT.
      GOBACK.
```

図 64. CWA のロードのサンプル・コード

TCTTE ユーザー域 (TCTUA) の使用

TCT ユーザー域 (TCTUA) は、端末管理テーブル項目 (TCTTE) に対するオプションの拡張です。TCT の各項目は、この拡張が現在存在しているかどうか、および存在している場合には、その長さはどのくらいかを指定します (端末用の TYPETERM リソース定義の USERAREALEN 属性を使用して)。

TYPETERM リソース定義について詳しくは、「*CICS Resource Definition Guide*」のモデル端末定義の自動インストールを参照してください。

システム初期化パラメーター TCTUALOC および TCTUAKEY は、すべての TCTUA のロケーションおよびストレージ・キーを指定します。

- TCTUALOC=BELOW または ANY は、TCTUA に対するアドレッシング可能度を、24 ビットにするのか 31 ビットにするのかを指定します。さらに、TCTUA が 16 MB 境界より下に保管されるようにするのか、16MB 境界より上下どちらにでも保管できるようにするのかを指定します。
- TCTUAKEY=USER または CICS は、TCTUA をユーザー・キー・ストレージから割り振るのか、CICS キー・ストレージから割り振るのかを指定します。

TCTUA は、CWA と共通する次の特性を持っています。

- プロセッサー・オーバーヘッドがきわめて小さい (必要な ADDRESS コマンドは 1 だけ)
- リカバリーされない
- リソース保護がない
- CICS による使用制限がない
- 固定長
- 大量のデータまたは存続期間の短いデータには適していない

しかし、CWA とは違い、通常、特定の端末の TCTUA は、その端末を使用するトランザクションの間だけで共用されます。したがって、一連の疑似会話型シーケンスのトランザクションの間の、かなり標準的な長さの少量のデータには有用です。もう 1 つの違いは、TCTUA は TCTTE がセットアップされる間だけ存在しているので、永続的に割り振る必要がない点です。TCTUA は、非自動インストール端末に対してはシステム始動から、自動インストール端末に対しては TCTTE が生成される時点で、割り振られます。

この方法で TCTUA を使用すると、常に、データを書き込んだトランザクションのすぐ後のトランザクションがそのデータを読み取るので、トランザクションの使用に関する特別の規律は必要ありません。しかし、TCTUA を使用して長期間にわたるデータ (例えば、アプリケーション全体に必要な端末情報またはオペレーター情報など) を保管する場合には、あるトランザクションで使用するデータが別のトランザクションで使用するデータと重ならないように、CWA の場合と同じ注意が必要です。割り振られた TCTUA の長さを超えると記憶保護違反になるので、これが起こらないようにしてください。

RETURN コマンドでの COMMAREA の使用

RETURN コマンドの COMMAREA オプションは、疑似会話型シーケンスの連続したトランザクションの間でデータを受け渡しするために、特別に設計されています。アプリケーション・プログラムではなく EXEC インターフェースが、GETMAIN 要求および FREEMAIN 要求を発行するので、これはユーザー・ストレージの特殊な形式として組み込まれます。

COMMAREA は、疑似会話型アプリケーションのタスクの間で、主記憶装置の CICS 共有サブプールから割り振られ、TCTTE によってアドレッシングされます。COMMAREA は次のタスクに渡されない限り解放されます。

次のタスクの最初のプログラムは、プログラムが LINK コマンド、または XCTL コマンドによって呼び出されたかのように、渡された COMMAREA に対して、自動的にアドレッシング可能になります (262 ページの『LINK コマンドおよび XCTL コマンドにおける COMMAREA』を参照してください)。また、ADDRESS コマンドの COMMAREA オプションを使用して、COMMAREA のアドレスを入手することもできます。

分散環境で疑似会話型シーケンスの連続したトランザクションの間で受け渡しされる COMMAREA の場合、VTAM の合計最大データ長は 32KB です。この制限は、VTAM によって追加される制御データを含む伝送パッケージ全体に適用されます。伝送が中間リンクを使用する場合には、制御データ容量が増加します。

要約すると次のとおりです。

- プロセッサーのオーバーヘッドは大きくない (XCTL コマンドで COMMAREA を使用するのと同等で、主一時記憶域を使用するのとほぼ同じです)。
- リカバリー可能ではない。
- リソース保護はない。
- 非常に大量のデータには適さない (主記憶装置が使用され、端末ユーザーが応答するまで保留されているため)。
- プログラム間でデータを転送するために COMMAREA を使用するのと同様に、トランザクション内の最初のプログラムが、データまたはそのアドレスを後続のプログラムに明示的に渡さない限り、そのプログラムに対してのみ利用可能です。

RETURN コマンドにおけるチャネルの使用

連絡域 (COMMAREA) を使用する代わりに、疑似会話型で次のプログラムにデータを受け渡す最新の方法では、**チャネル**を使用します。チャネルには、COMMAREA に対するいくつかの利点があります。 319 ページの『チャネルの利点』を参照してください。RETURN コマンドでチャネルを受け渡すには、COMMAREA オプションの代わりに CHANNEL オプションを使用します。

チャネルについては、 289 ページの『第 20 章 拡張プログラム間データ転送: 新しい COMMAREA としてのチャネル』で説明しています。

データを共用する表示画面の使用

3270 ディスプレイ端末からの疑似会話型トランザクションの間で、データを表示画面そのものに保管することもできます。例えば、ユーザーが画面に入力することを要求されたデータでエラーを起こした場合には、入力のトランザクション処理は、通常、画面上のエラーを (強調表示またはメッセージによって) 指示し、(訂正後の入力を処理するように) 次のトランザクション ID を、そのトランザクション自身のものに設定して、CICS に戻ります。

トランザクションが***有効*** データを使用する方法は 2 つあります。トランザクションがデータ (例えば、COMMAREA 内のデータ) を保管し、次に実行するときにそれを渡します。この場合には、トランザクションは、画面上で変更済みのデータを直前の入力からのデータと組み合わせる必要があります。あるいは、キー付きフィールドの修正データ・タグをオフにしないで、データを画面上に保管することができます。

画面上にデータを保管しておくためのコーディングは非常に簡単ですが、次の 2 つの制約があります。まず第一に、エラーが発生する可能性が高い場合には、大量のデータを含む画面を保管することはお勧め**できません**。それは、未変更データを再送するために追加のネットワーク・トラフィックが必要になるためです。(この制約事項は、ローカル接続された端末には適用されません。)

第二に、ユーザーが CLEAR キーを押した場合には、画面データは失われるので、トランザクションはこれからリカバリーできる必要があります。CLEAR キーが CANCEL または QUIT を意味するように定義して (関連のアプリケーションに適切な場合)、これを回避することができます。

キー付きデータ以外のデータも画面上に保管することができます。このデータは変更 (CLEAR によって引き起こされるもの以外) から保護し、非表示とすることができます (必要な場合)。

第 20 章 拡張プログラム間データ転送: 新しい COMMAREA としてのチャネル

従来、CICS プログラムは連絡域 (COMMAREA) を使用してデータを交換してきました。このセクションでは、COMMAREA に適用される、32KB 制限をはるかに上回るデータ転送をプログラム間で行うための改善された方法について説明します。

このセクションには、次のものが含まれています。

- 『チャネル: クイック・スタート』
- 292 ページの『チャネルの使用: いくつかの典型的なシナリオ』
- 295 ページの『チャネルの作成』
- 296 ページの『現行チャネル』
- 301 ページの『チャネルの有効範囲』
- 305 ページの『プログラムに渡されたコンテナーの検出』
- 305 ページの『リンクから返されたコンテナーの検出』
- 306 ページの『CICS 読み取り専用コンテナー』
- 306 ページの『チャネルの設計: 最良事例』
- 308 ページの『チャネルの構成および使用: 例』
- 309 ページの『チャネルと BTS アクティビティー』
- 311 ページの『JCICS からのチャネルの使用』
- 312 ページの『チャネルを使用した動的ルーティング』
- 313 ページの『データ変換』
- 319 ページの『チャネルの利点』
- 321 ページの『COMMAREA からチャネルへのマイグレーション』

チャネル: クイック・スタート

コンテナーおよびチャネル

コンテナー とは、プログラム間で情報を受け渡すために設計されたデータのブロックのことです。コンテナーは、「指定された連絡域 (COMMAREA)」と考えることができます。プログラムは、任意の数のコンテナーを相互に受け渡すことができます。コンテナーは、チャネルと呼ばれるセットにグループ化されます。チャネルはパラメーター・リストに類似しています。

名前付きコンテナーを作成してチャネルに割り当てるために、プログラムは EXEC CICS PUT CONTAINER(コンテナー名) CHANNEL(チャネル名) コマンドを使用します。次に、EXEC CICS LINK、XCTL、START、または RETURN の各コマンドで CHANNEL(チャネル名) オプションを使用して、チャネル (およびそのコンテナー) を 2 番目のプログラムに渡すことができます。

2 番目のプログラムは、EXEC CICS GET CONTAINER(コンテナー名) コマンドを使用して、渡されたコンテナーを読み取ることができます。このコマンドは、プログラムを起動したチャネルに属する名前付きコンテナーを読み取ります。

© Copyright IBM Corp. 1989, 2009

2 番目のプログラムが LINK コマンドにより起動された場合、コンテナーを呼び出し側プログラムに戻すこともできます。この処理は、新しいコンテナーを作成するか、または既存のコンテナーを再利用することにより実行できます。

チャネルおよびコンテナーは、それらを作成したプログラム、およびそれらが渡されるプログラムでのみ認識できます。これらのプログラムが終了した場合、CICS は自動的にコンテナーおよびそのストレージを破棄します。

チャネル・コンテナーはリカバリー可能ではありません。リカバリー可能なコンテナーを使用する必要がある場合は、CICS Business Transaction Services (BTS) コンテナーを使用します。

基本的な例

291 ページの図 65では、以下のような COBOL プログラム CLIENT1 について説明します。

1. PUT CONTAINER(コンテナー名) CHANNEL(チャネル名) コマンドを使用して、inqcustrec と呼ばれるチャネルを作成し、2 つのコンテナー custno および branchno を追加します。これらには、顧客番号と支店番号がそれぞれ含まれます。
2. LINK PROGRAM(プログラム名) CHANNEL(チャネル名) コマンドを使用して、プログラム SERVER1 にリンクし、inqcustrec チャネルを渡します。
3. GET CONTAINER(コンテナー名) CHANNEL(チャネル名) コマンドを使用して、SERVER1 により返される顧客レコードを取得します。顧客レコードは、inqcustrec チャネルの custrec コンテナーにあります。

クライアント・プログラムとサーバー・プログラムの両方により、同じ COBOL コピーブック INQINTC が使用されることに注意してください。コピーブックの 3 行目と、5 行目から 7 行目までは、INQUIRY-CHANNEL およびそのコンテナーを表しています。コンテナーおよびチャネルは (例えば、PUT CONTAINER コマンド上で) *名前指定* されることで作成されるため、上記の行はプログラムの動作に必ずしも必要というわけではありません。これらの行は定義しなくても差し支えありません。ただし、両方のプログラムで使用されるコピーブックにこれらの行を含めておくと、保守が容易になります。これらの行で、使用されるコンテナーの名前を記録しておくことができます。

推奨

チャネルを使用するクライアント/サーバー・アプリケーションの保守を容易にするには、使用するコンテナーの名前を記録して、そのコンテナーにマップするデータ・フィールドを定義するコピーブックを作成してください。クライアント・プログラムとサーバー・プログラムの両方に、そのコピーブックを含めます。

注: この例では、2 つの COBOL プログラムを示します。これと同じ技法を、CICS がサポートする他の言語でも使用できます。ただし、*COBOL プログラムの場合*のみ、サーバー・プログラムが EXEC CICS GET CONTAINER コマンドで (INTO の代わりに) SET オプションを使用する場合は、SET で指すストレージの構造をプログラムの LINKAGE SECTION で定義する必要があります。つまり、コピーブックが 1 つではなく 2 つ必要であることを意味します。まず、プログラムの

WORKING-STORAGE SECTION で、使用するチャネルおよびコンテナーを指定します。次に、LINKAGE SECTION で、ストレージ構造を定義します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CLIENT1.

WORKING-STORAGE SECTION.

  COPY INQINTC
*                 copybook INQINTC
*  Channel name
*  01 INQUIRY-CHANNEL  PIC X(16) VALUE 'inqcustrec'.
*  Container names
*  01 CUSTOMER-NO        PIC X(16) VALUE 'custno'.
*  01 BRANCH-NO          PIC X(16) VALUE 'branchno'.
*  01 CUSTOMER-RECORD    PIC X(16) VALUE 'custrec'.
*  Define the data fields used by the program
*  01 CUSTNO     PIC X(8).
*  01 BRANCHNO   PIC X(5).
*  01 CREC.
*    02 CUSTNAME   PIC X(80).
*    02 CUSTADDR1  PIC X(80).
*    02 CUSTADDR2  PIC X(80).
*    02 CUSTADDR3  PIC X(80).

PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.

*
* INITIALISE CUSTOMER RECORD
*
     ... CREATE CUSTNO and BRANCHNO
*
* GET CUSTOMER RECORD
*
    EXEC CICS PUT CONTAINER(CUSTOMER-NO) CHANNEL(INQUIRY-CHANNEL)
                  FROM(CUSTNO) FLENGTH(LENGTH OF CUSTNO)
                  END-EXEC
    EXEC CICS PUT CONTAINER(BRANCH-NO) CHANNEL(INQUIRY-CHANNEL)
                  FROM(BRANCHNO) FLENGTH(LENGTH OF BRANCHNO)
                  END-EXEC

    EXEC CICS LINK PROGRAM('SERVER1') CHANNEL(INQUIRY-CHANNEL) END-EXEC

    EXEC CICS GET CONTAINER(CUSTOMER-RECORD) CHANNEL(INQUIRY-CHANNEL)
                  INTO(CREC) END-EXEC

*
* PROCESS CUSTOMER RECORD
*
    ... FURTHER PROCESSING USING CUSTNAME and CUSTADDR1 etc...

    EXEC CICS RETURN END-EXEC

    EXIT.
```

図 65. チャネルを作成して 2 番目のプログラムに渡すプログラムの簡単な例

292 ページの図 66 では、CLIENT1 がリンクする SERVER1 プログラムを示します。SERVER1 は、渡された custno および branchno の各コンテナーからデータを取得し、そのデータを使用してデータベース内で完全な顧客レコードを見つけます。次に、新しいコンテナー custrec を同じチャネル上に作成して、顧客レコードをその中に返します。

プログラマーは SERVER1 の GET コマンドおよび PUT コマンドに CHANNEL キーワードを指定していないことに注意してください。チャネルが明示的に指定されていない場合、現行チャネル (つまり、プログラムを起動したチャネル) が使用されます。

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SERVER1.

WORKING-STORAGE SECTION.

  COPY INQINTC
*                 copybook INQINTC
*  Channel name
*  01 INQUIRY-CHANNEL  PIC X(16) VALUE 'inqcustrec'.
*  Container names
*  01 CUSTOMER-NO        PIC X(16) VALUE 'custno'.
*  01 BRANCH-NO          PIC X(16) VALUE 'branchno'.
*  01 CUSTOMER-RECORD    PIC X(16) VALUE 'custrec'.
*  Define the data fields used by the program
*  01 CUSTNO     PIC X(8).
*  01 BRANCHNO   PIC X(5).
*  01 CREC.
*    02 CUSTNAME   PIC X(80).
*    02 CUSTADDR1  PIC X(80).
*    02 CUSTADDR2  PIC X(80).
*    02 CUSTADDR3  PIC X(80).

PROCEDURE DIVISION.
MAIN-PROCESSING SECTION.

    EXEC CICS GET CONTAINER(CUSTOMER-NO)
                        INTO(CUSTNO) END-EXEC
    EXEC CICS GET CONTAINER(BRANCH-NO)
                        INTO(BRANCHNO) END-EXEC

       ... USE CUSTNO AND BRANCHNO TO FIND CREC IN A DATABASE

    EXEC CICS PUT CONTAINER(CUSTOMER-RECORD)
                  FROM(CREC)
                  FLENGTH(LENGTH OF CREC) END-EXEC

    EXEC CICS RETURN END-EXEC

    EXIT.
```

図 66. データが渡されたチャネルからそのデータを取得するリンク・プログラムの簡単な例 : このプログラムは、 291 ページの図 65 で示すように、プログラム CLIENT1 からリンクされています。

チャネルの使用: いくつかの典型的なシナリオ

チャネルおよびコンテナーは、プログラム間でのデータの受け渡しで非常に役立ちます。このセクションでは、チャネルの使用方法について、いくつか例を示します。

1 つのチャネルに 1 つのプログラム

図 67 では、もっとも簡単なシナリオである、単一チャネルを使用して、このチャネルから起動できる「独立型」プログラムについて説明します。

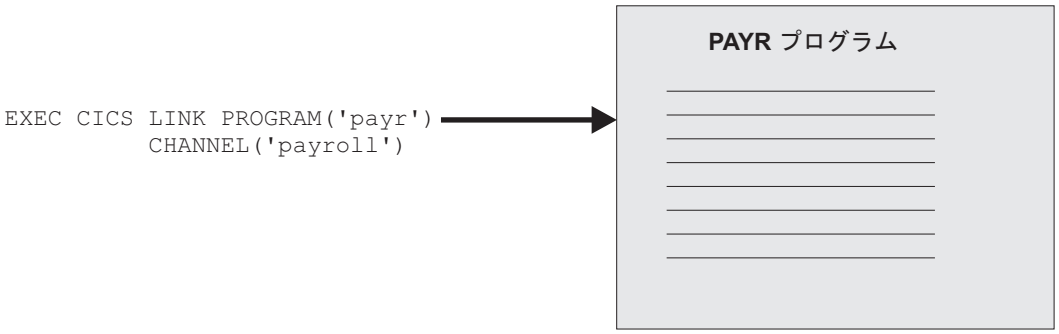


図 67. 単一チャネルを使用した独立型プログラム

1 つのチャネルに複数のプログラム (1 つのコンポーネント)

図 68 では、相互に関連するプログラム・セットの最上位プログラムに対して、単一のチャネルが存在します。陰影の領域内のプログラムのセットを、「コンポーネント」と見なすことができます。クライアント・プログラムは外部チャネルのみを「認識」し、発生する処理も、バックエンド・プログラムの存在も認識しません。

コンポーネント内では、複数プログラムが、それらのプログラム間でチャネルを受け渡すことができます。あるいは、例えば、コンポーネント・プログラムが新しいチャネルを作成し、元のチャネルから 1 つ以上のコンテナーを追加することによって、元のチャネルのサブセットを渡すことができます。

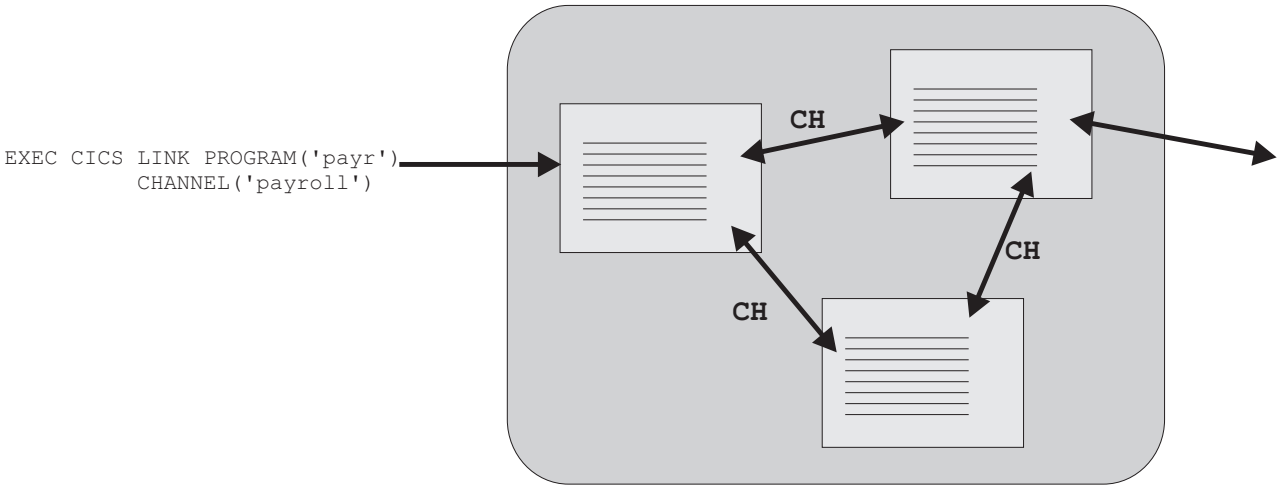


図 68. 『コンポーネント』 - 単一外部チャネルを経由して起動される関連プログラムのセット： 『CH』 は、コンポーネント内の複数プログラムが、それらのプログラム間でチャネルを受け渡すことができることを示します。

複数のチャネルに 1 つのコンポーネント

前の例では、コンポーネントと見なすことができる、相互に関連するプログラムのセットについて説明しました。このセクションでは、コンポーネントを起動できる 2 つの代替外部チャネルについて説明します。コンポーネントの最上位プログラムは、EXEC CICS ASSIGN CHANNEL コマンドを実行して、起動に使用されたチャネルを判別し、それに合わせて処理を調整します。

クライアント・プログラムとコンポーネントとの間の「緩やかな結合」により、改善が容易になりました。つまり、クライアントとコンポーネントを、異なる時期にアップグレードできます。例えば、まずコンポーネントをアップグレードして、1 番目または 2 番目のチャネルの異なるコンテナー・セットから構成される、3 番目のチャネルを処理するようにします。次に、3 番目のチャネルを受け渡すようにクライアント・プログラムをアップグレード (または新しいクライアントを作成) します。

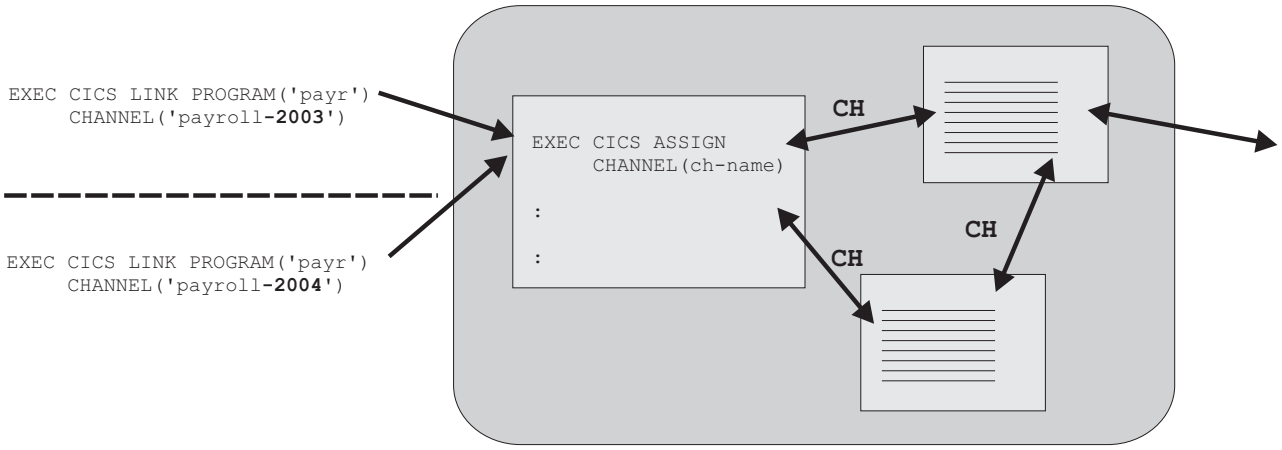


図 69. 同一コンポーネントへの複数の外部チャネル : 『CH』 は、コンポーネント内の複数プログラムが、それらのプログラム間でチャネルを受け渡す場合があることを示します。

複数の対話式コンポーネント

295 ページの図 70 では、「人的要員」コンポーネントおよび「給与計算」コンポーネントについて説明します。どちらのコンポーネントにも、起動元となるチャネルがあります。給与計算コンポーネントは、独立型プログラムと人的要員コンポーネントの両方から起動されます。

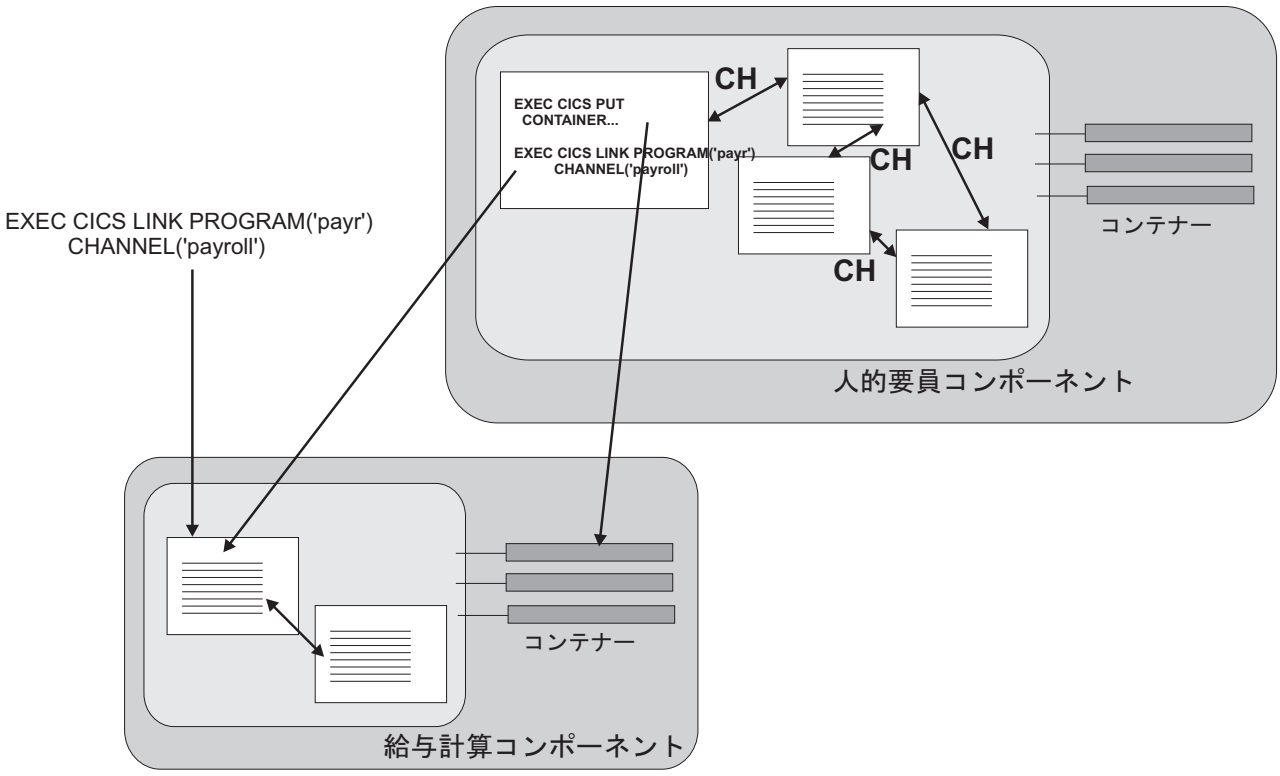


図 70. チャネルを経由して対話する複数のコンポーネント

チャネルの作成

このタスクについて

チャネルを作成するには、以下のコマンドのいずれかでそのチャネルを指名します。

EXEC CICS LINK PROGRAM CHANNEL
EXEC CICS MOVE CONTAINER CHANNEL TOCHANNEL
EXEC CICS PUT CONTAINER CHANNEL
EXEC CICS RETURN TRANSID CHANNEL
EXEC CICS START TRANSID CHANNEL
EXEC CICS XCTL PROGRAM CHANNEL

そのチャネルが現行プログラムのスコープ内に存在していなければ、作成されます。

チャネルを作成し、そのチャネルにデータのコンテナーを移植する最も単純な方法は、一連の EXEC CICS PUT CONTAINER(*container-name*) CHANNEL(*channel-name*) FROM(*data_area*) コマンドを発行することです。最初の PUT コマンドは、チャネルを (存在していなければ) 作成し、そのチャネルにコンテナーを追加します。その後に続くコマンドは、チャネルにさらにコンテナーを追加します。そのコンテナーが既に存在している場合は、その内容が新しいデータで上書きされます。

チャネルにコンテナーを追加するもう 1 つの方法は、追加するコンテナーを別のチャネルから移動する方法です。これを行うには、以下のコマンドを使用します。

```
EXEC CICS MOVE CONTAINER(container-name) AS(container-new-name)
     CHANNEL(channel-name1) TOCHANNEL(channel-name2)
```

注:

1. CHANNEL または TOCHANNEL オプションが指定されない場合は、現行チャネルであることを意味する。
2. ソース・チャネルはプログラムのスコープ内にある必要がある。
3. ターゲットのチャネルが、現行プログラムのスコープ内に存在していない場合は、作成される。
4. ソース・コンテナーが存在していない場合は、エラーが発生する。
5. ターゲットのコンテナーが存在していない場合は作成され、既に存在している場合は内容が上書きされる。

チャネル間でのデータ移動をより効果的に行う方法として、GET CONTAINER および PUT CONTAINER の代わりに MOVE CONTAINER を使用することもできます。

以下のコマンドで指定されたチャネルが現行プログラムのスコープ内に存在していない場合は、空の チャネルが作成されます。

- EXEC CICS LINK PROGRAM CHANNEL(*channel-name*)
- EXEC CICS RETURN TRANSID CHANNEL(*channel-name*)
- EXEC CICS START TRANSID CHANNEL(*channel-name*)
- EXEC CICS XCTL PROGRAM CHANNEL(*channel-name*)

現行チャネル

プログラムの**現行チャネル**とは、そのプログラムの呼び出しに使用されたチャネル (もしあれば) です。プログラムはその他のチャネルを作成することができます。しかし、特定のプログラムの特定の呼び出しに使用される*現行* チャネルは変更されません。これはパラメーター・リストと類似しています。

LINK コマンドを使用した現行チャネルの例

以下の図は、プログラムの現行チャネルの起点を示しています。この図には、5 つの対話式プログラムがあります。プログラム A は、例えば端末のエンド・ユーザーによって始動された、トップレベルのプログラムです。このプログラムは、プログラムからは始動されず、現行チャネルも持っていません。

B、C、D、および E は、それぞれ第 2、第 3、第 4、および第 5 レベルのプログラムです。

プログラム B の現行チャネルは、プログラム A によって発行された EXEC CICS LINK コマンドの CHANNEL オプションによって渡される X です。プログラム B は、チャネル X にコンテナーを 1 つ追加し、別のコンテナーを削除するという変更を加えます。

プログラム C の現行チャネルも、プログラム B によって発行された EXEC CICS LINK コマンドの CHANNEL オプションによって渡される X です。

プログラム C から現行チャネルが渡されていないため、プログラム D には現行チャネルはありません。

プログラム E の現行チャネルは、プログラム D によって発行された EXEC CICS LINK コマンドの CHANNEL オプションによって渡される Y です。

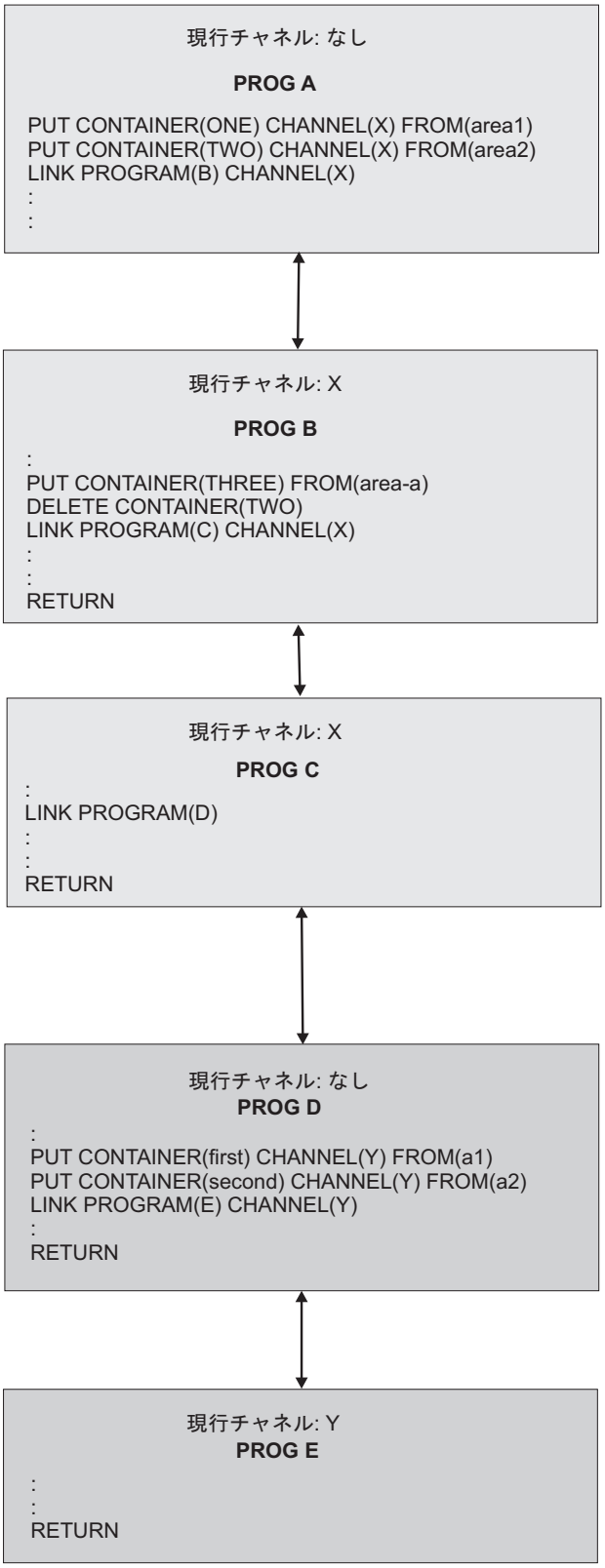


図 71. 現行チャネル: LINK コマンドを使用した例

以下のテーブルには、前の図で示された 5 つのプログラムそれぞれの現行チャネル (もしあれば) の名前がリストされています。

表 12. 対話式プログラムの現行チャネル - LINK コマンドを使用した例

プログラム	現行チャネル	コマンドの発行	コメント
A	なし	. EXEC CICS PUT CONTAINER(ONE) CHANNEL(X) FROM(area1) EXEC CICS PUT CONTAINER(TWO) CHANNEL(X) FROM(area2) EXEC CICS LINK PROGRAM(B) CHANNEL(X) .	プログラム A はチャネル X を作成し、プログラム B に渡します。 制御がプログラム B によりプログラム A に戻されるまでに、X チャネルは変更される場合があることに注意してください。プログラム A によって作成されたときと同じコンテナー・セットが X チャネルに含まれているとは限りません (プログラム B により、コンテナー TWO が削除され、コンテナー THREE が追加されました)。
B	X	. EXEC CICS PUT CONTAINER(THREE) FROM(area-a) EXEC CICS DELETE CONTAINER(TWO) EXEC CICS LINK PROGRAM(C) CHANNEL(X) . . EXEC CICS RETURN	プログラム B は、コンテナーの追加および削除によってチャネル X (そのプログラムの現行チャネル) に変更を加え、変更済みのチャネルをプログラム C に渡します。 プログラム B の現行チャネルは暗黙に決まっているため、PUT CONTAINER コマンドおよび DELETE CONTAINER コマンドで CHANNEL オプションを指定する必要はありません。
C	X	. EXEC CICS LINK PROGRAM(D) . . EXEC CICS RETURN	プログラム C はプログラム D とリンクしますが、チャネルは渡しません。
D	なし	. EXEC CICS PUT CONTAINER(first) CHANNEL(Y) FROM(a1) EXEC CICS PUT CONTAINER(second) CHANNEL(Y) FROM(a2) EXEC CICS LINK PROGRAM(E) CHANNEL(Y) . . EXEC CICS RETURN	プログラム D はチャネル Y を新規作成し、プログラム E に渡します。
E	Y	. RETURN .	プログラム E は、渡されたデータに対して幾つかの処理を行ってから返します。

XCTL コマンドを使用した現行チャネルの例

300 ページの図 72 では、4 つの対話式プログラムを示しています。A1 は、端末のエンド・ユーザーなどによって始動されるトップレベルのプログラムです。このプログラムは、プログラムからは始動されず、現行チャネルも持っていません。B1、B2、および B3 は、すべて第 2 レベルのプログラムです。

B1 の現行チャネルは、A1 によって発行された EXEC CICS LINK コマンドの CHANNEL オプションによって渡される X です。

B1 から現行チャネルが渡されていないため、B2 には現行チャネルはありません。

B3 の現行チャネルは、B2 によって発行された EXEC CICS XCTL コマンドの CHANNEL オプションによって渡される Y です。

B3 が戻ると、チャネル Y およびそのコンテナーは、CICS によって削除されます。

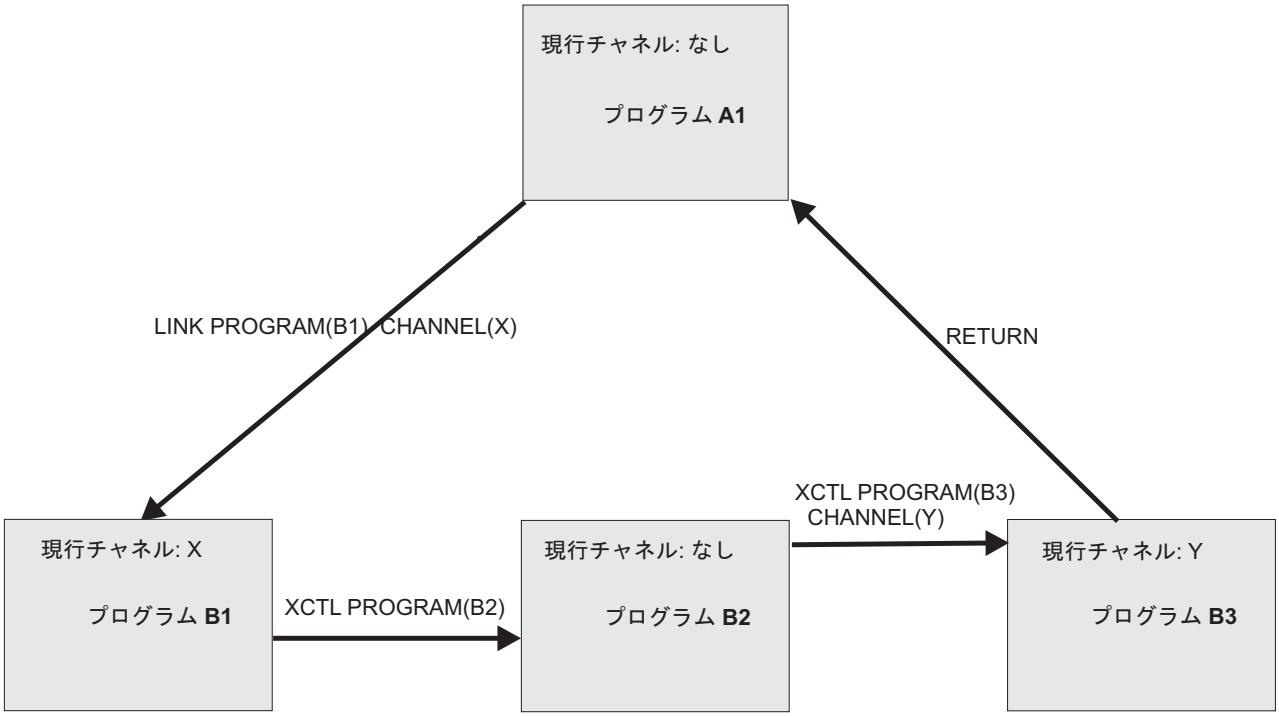


図 72. XCTL コマンドを使用した現行チャネルの例

以下のテーブルには、図 72 で示された 4 つのプログラムそれぞれの現行チャネル (もしあれば) の名前がリストされています。

表 13. 対話式プログラムの現行チャネルの例

プログラム	現行チャネル	発行コマンド
A1	なし	. EXEC CICS LINK PROGRAM(B1) CHANNEL(X) .
B1	X	. EXEC CICS XCTL PROGRAM(B2) .
B2	なし	. EXEC CICS XCTL PROGRAM(B3) CHANNEL(Y) .
B3	Y	. EXEC CICS RETURN .

現行チャネル: START および RETURN コマンド

プログラムを呼び出してチャネルに渡すには、EXEC CICS LINK および XCTL の他に、2 つのコマンドを使用することができます。

EXEC CICS START TRANSID(*tranid***) CHANNEL(***channel-name***)**
開始済みのトランザクションをインプリメントするプログラム (複数存在する場合は最初のプログラム) がチャネルに渡され、そのチャネルがそのプログラムの現行チャネルになります。

EXEC CICS RETURN TRANSID(*tranid***) CHANNEL(***channel-name***)**
CHANNEL オプションは以下の場合にのみ有効です。
- 疑似会話型 RETURN、つまり、ユーザー端末で実行される次のトランザクションを TRANSID オプションで指定する RETURN コマンドの場合。
- 最高論理レベルのプログラム、つまり、CICS に制御を返すプログラムによって発行される場合。

次のトランザクションをインプリメントするプログラムがチャネルに渡され、そのチャネルがそのプログラムの現行チャネルになります。

チャネルの有効範囲

チャネルの**有効範囲**とは、チャネルにアクセスできるコードのことです。

LINK コマンドを使用した有効範囲の例

以下の図に、『LINK コマンドを使用した現行チャネルの例』で説明した 5 つの同一の対話式プログラムを示します。

X チャネルの有効範囲 (つまり、そのチャネルにアクセスできるコード) は、プログラム A、B、および C です。

Y チャネルの有効範囲は、プログラム D および E です。

制御がプログラム B によりプログラム A に返されるまでに、X チャネルは変更される場合があることに注意してください。プログラム A によって作成されたときと同じコンテナー・セットが X チャネルに含まれているとは限りません。

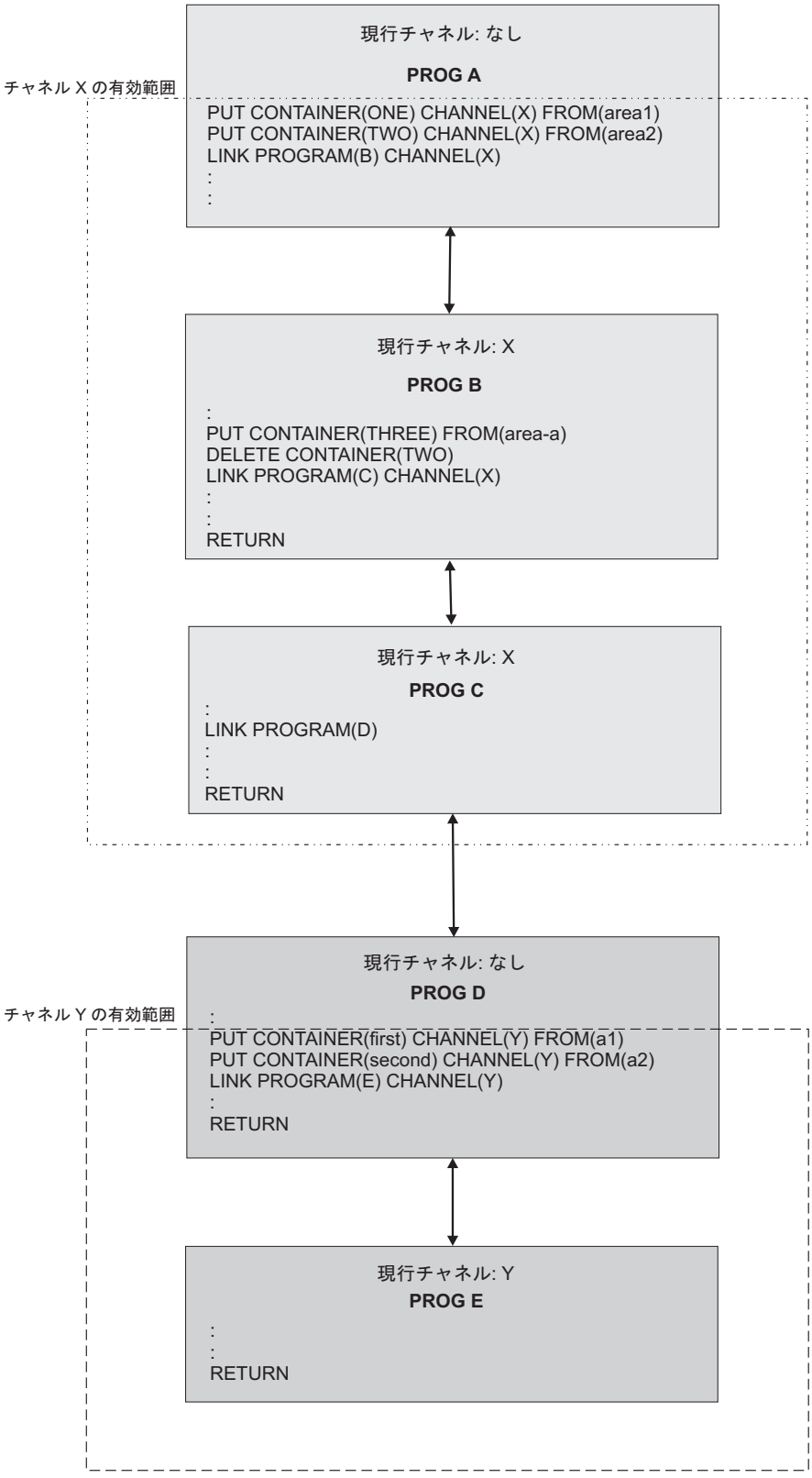


図 73. チャネルの有効範囲 - LINK コマンドの例

以下のテーブルに、前の図で示した 5 つの各プログラムの現行チャネル (ある場合) の、名前と有効範囲を示します。

表 14. チャネルの有効範囲 - LINK コマンドを示す例

プログラム	現行チャネル	チャネルの有効範囲
A	なし	適用されない
B	X	A、B、C
C	X	A、B、C
D	なし	適用されない
E	Y	D、E

LINK コマンドおよび XCTL コマンドを使用した有効範囲の例

304 ページの図 74 に、『XCTL コマンドを使用した現行チャネルの例』で説明した 4 つの同一の対話式プログラム、およびプログラム B1 から EXEC CICS LINK コマンドにより起動された 第 3 レベルのプログラム C1 を示します。

X チャネルの有効範囲は、A1 と B1 に制限されます。

Y チャネルの有効範囲は、B2 と B3 です。

Z チャネルの有効範囲は、B1 と C1 です。

制御がプログラム B3 によりプログラム A1 に返されるまでに、X チャネルはプログラム B1—it により変更される場合があることに注意してください。A1 によって作成されたときと同じコンテナー・セットが X チャネルに含まれているとは限りません。

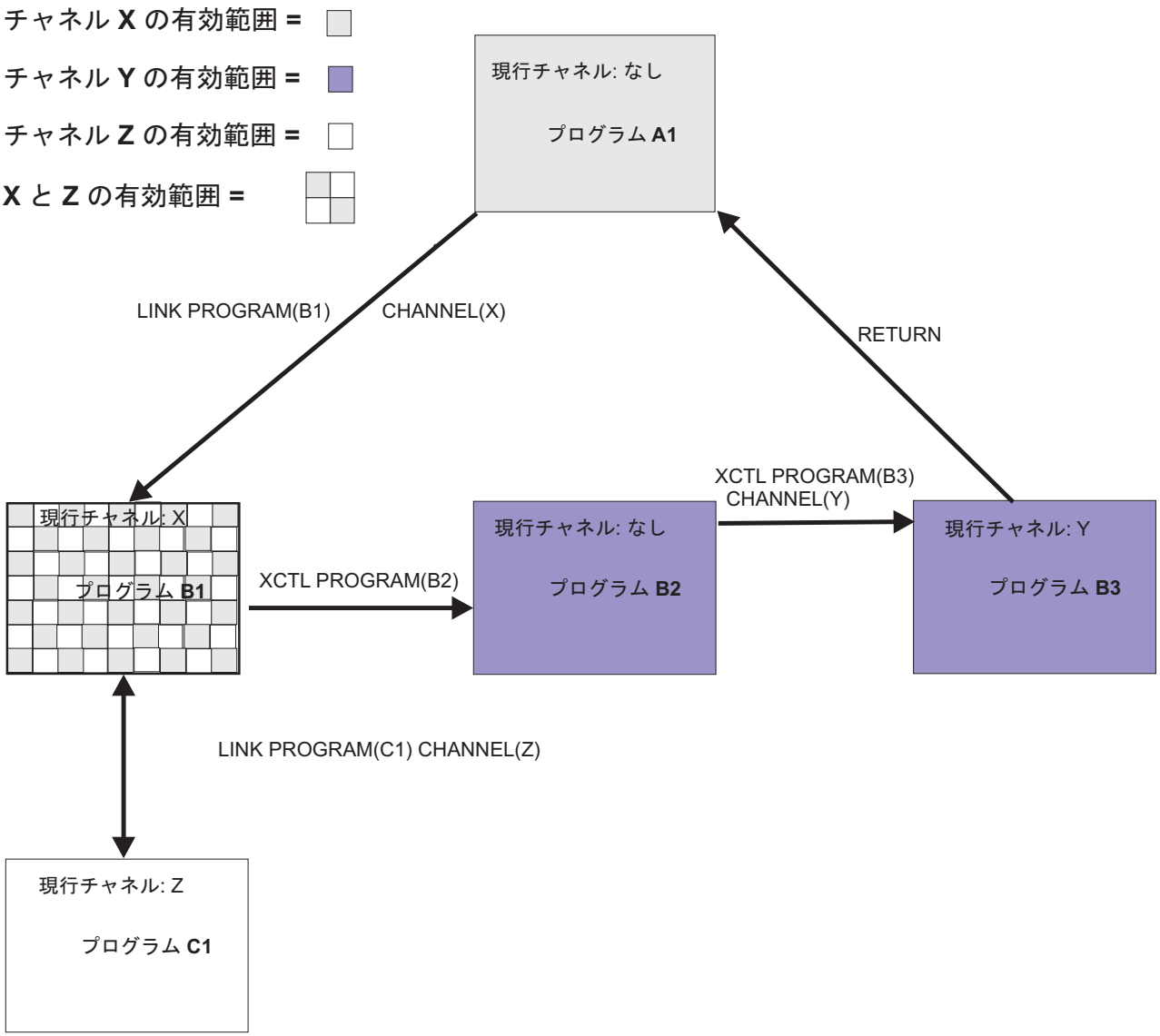


図 74. チャネルの有効範囲 - LINK と XCTL コマンドを示す例

以下のテーブルに、図 74 で示した 5 つの各プログラムの現行チャネル (ある場合) の名前と有効範囲をリストします。

表 15. チャネルの有効範囲 - LINK と XCTL コマンドの例

プログラム	現行チャネル	チャネルの有効範囲
A1	なし	適用されない
B1	X	A1 および B1
B2	なし	適用されない
B3	Y	B2 および B3
C1	Z	B1 および C1

プログラムに渡されたコンテナーの検出

通常、チャネルを交換するプログラムは、そのチャネルを処理するよう作成されます。つまり、クライアント・プログラムとサーバー・プログラムの両方が、チャネルの名前、およびチャネル内のコンテナーの名前と数を認識しています。ただし、例えば、サーバー・プログラムまたはコンポーネントが複数のチャネルを処理するように作成されている場合、それらのチャネルのうちどれが渡されたかを、起動時に検出する必要があります。

プログラムは現行チャネル (つまり、起動に使用したチャネル) を、EXEC CICS ASSIGN CHANNEL コマンドを実行することにより検出できます (現行チャネルがない場合、コマンドはブランクを返します)。

プログラムは (必要に応じて)、ブラウズにより現行チャネル内のコンテナーの名前を取得することもできます。通常、この処理は必要ありません。複数のチャネルを処理するよう作成されたプログラムは、それらの各チャネル内のコンテナーの名前と数を認識するようにコーディングされている場合が多いです。

現行チャネル内のコンテナーの名前を取得するには、以下のようなブラウズ・コマンドを使用します。

- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) .
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

サーバー・プログラムは、現行チャネルの名前、および必要に応じてチャネル内のコンテナーの名前を取得して、渡されるデータの種類に合うよう処理を調整できます。

リンクから返されたコンテナーの検出

プログラムはチャネルを作成し、そのチャネルを EXEC CICS LINK PROGRAM(*program-name*) CHANNEL(*channel-name*) コマンドにより 2 番目のプログラムに渡します。2 番目のプログラムは、渡されたデータを一部処理し、結果を同じチャネル (現行チャネル) に返します。

次に、最初のプログラムは返されたチャネルの名前を認識しますが、チャネル内のコンテナーの名前は必ずしも認識するわけではありません (返されたチャネルには、渡されたチャネルと同じコンテナーが含まれるか、あるいは、2 番目のプログラムが一部を削除または作成しました)。最初のプログラムは、コンテナー名をブラウズにより検出できます。そのためには、以下のコマンドを使用します。

- EXEC CICS STARTBROWSE CONTAINER BROWSETOKEN(*data-area*) CHANNEL(*channel-name*).
- EXEC CICS GETNEXT CONTAINER(*data-area*) BROWSETOKEN(*token*).
- EXEC CICS ENDBROWSE CONTAINER BROWSETOKEN(*token*).

CICS 読み取り専用コンテナー

CICS は、自分で使用するためにチャネルおよびコンテナーを作成し、それらをユーザー・プログラムに渡すことができます。CICS はこれらのコンテナーに読み取り専用のマークを付けることがあります。そのような場合、ユーザー・プログラムから戻る時点で CICS が特定のデータを必要としても、ユーザー・プログラムはコンテナーを変更できません。

ユーザー・プログラムは読み取り専用コンテナーを作成できません。

読み取り専用コンテナーは、上書き、移動、または削除できません。このため、PUT CONTAINER、MOVE CONTAINER、または DELETE CONTAINER の各コマンドで読み取り専用コンテナーを指定した場合、INVREQ 状態が発生します。

チャネルの設計: 最良事例

連絡域 (COMMAREA) を使用していた従来の方法と同じ方法で、コンテナーを使用してデータを渡すことができます。しかし、COMMAREA と比べてチャネルには幾つかの利点があるため、チャネルを設計して、そうした改善を行うことは有益です。

このタスクについて

サーバー・プログラムによって変更された入力コンテナーは、DPL 呼び出しの最後にクライアントに戻されません。サーバー・プログラムによって内容を変更された入力コンテナー、および、サーバー・プログラムによって作成されたコンテナーは、戻されます。したがって、DPL のパフォーマンスを最適にするには、以下のようにします。

- データの入力と出力で別々のコンテナーを使用する。
- 出力コンテナーは、クライアントではなくサーバー・プログラムによって作成する。
- 読み取り専用のデータと読み取り/書き込み可能なデータで別々のコンテナーを使用する。
- 構造がオプションの場合は、別のコンテナーにする。
- エラー情報には専用のコンテナーを使用する。

チャネルの設計に関する一般的なヒントは以下のとおりです。ここには、DPL のパフォーマンスを最適化するための推奨事項を含むとともに、さらに詳しく説明しています。

- データの入力と出力で別々のコンテナーを使用する。これにより、以下のようになります。
  - データのカプセル化が改善され、プログラムの保守が容易になる。
  - 各方向へのコンテナーの流れが小さくなるため、DPL 呼び出しでチャネルが渡される際の効率が大きく向上する。
- 出力コンテナーは、クライアントではなくサーバー・プログラムによって作成する。クライアントで作成すると、空のコンテナーがサーバー領域に送信されます。
- 読み取り専用のデータと読み取り/書き込み可能なデータで別々のコンテナーを使用する。これにより、以下のようになります。

  - コピーブックの構造が単純化され、プログラムが理解しやすくなる。
  - REORDER オーバーレイでの問題を回避する。
  - サーバー領域に送信された読み取り専用のコンテナーが戻されないため、CICS 領域間の送信効率が大きく向上する。
- 各構造体ごとに別々のコンテナーを使用する。これにより、以下のようになります。
  - データのカプセル化が改善され、プログラムの理解と保守が容易になる。
  - コンポーネント全体を再コンパイルする必要がないため、構造体の 1 つの変更が非常に容易になる。
  - MOVE CONTAINER コマンドを使用してコンテナーをチャネル間で移動することによって、チャネルのサブセットをサブコンポーネントに渡すことができる。
- 構造がオプションの場合は、別のコンテナーにする。これにより、コンテナーが存在する場合にのみ構造体が渡されるため、効率が大きく向上します。
- エラー情報には専用のコンテナーを使用する。これにより、以下のようになります。
  - エラー情報の文書化が向上する。
  - 以下の理由で、効率が向上する。
    1. エラー情報を含む構造体は、エラーが発生した場合にのみ戻される。
    2. チャネル内のコンテナーの表示を開始するより、GET CONTAINER (*known-error-container-name*) コマンドを発行する (さらに場合によっては NOTFOUND 状態を返す) ことによって、エラー・コンテナーの存在を確認する方が効率的である。
- さまざまなタイプのデータ (例えば、`codepage1` の文字データと `codepage2` の文字データ) を渡す必要がある場合は、複雑な構造を持つ単一のコンテナーを使用するのではなく、各タイプごとに別のコンテナーを使用する。これにより、さまざまなコード・ページ間を移動する能力が向上します。
- 大量のデータを渡す必要がある場合は、そのデータをすべて単一のコンテナーに入れるのではなく、複数のコンテナーに分割する。

  チャネルがリモート・プログラムまたはトランザクションに渡される場合、大量のデータを渡すことによってパフォーマンスに影響が出る可能性があります。これは特に、ローカル領域とリモート領域が MRO 接続ではなく ISC 接続によって接続されている場合にあてはまります。

  注意:
  大きなコンテナーを多数作成したために、他のアプリケーションが使用できるストレージの量が制限されることがないよう注意してください。

  関連概念

  321 ページの『COMMAREA からチャネルへのマイグレーション』
  従来の連絡域 (COMMAREA) を使用してデータを交換する CICS アプリケーション・プログラムは、以前と同様に動作することができます。この例は、新しい機能にマイグレーションする場合に、複数のタイプの既存アプリケーションをマイグレーションして、COMMAREA ではなくチャネルおよびコンテナーを使用できるようにする方法を示しています。

関連資料

319 ページの『チャネルの利点』

チャネルの構成および使用: 例

図 75 では、以下のような CICS クライアント・プログラムについて説明します。

1. EXEC CICS PUT CONTAINER コマンドを使用して、コンテナーのセットを構成 (およびコンテナーのセットにデータを追加) します。すべてのコンテナーが、同じ名前のチャネル 『payroll-2004』 に含まれています。
2. EXEC CICS LINK コマンドを実行して、PAYR サーバー・プログラムを起動し、そのプログラムに payroll-2004 チャネルを渡します。
3. EXEC CICS GET CONTAINER コマンドを実行して、サーバーのプログラムの出力を取得します。この出力は、payroll-2004 チャネルの status コンテナー内に入っていると認識されます。

```
* create the employee container on the payroll-2004 channel
EXEC CICS PUT CONTAINER('employee') CHANNEL('payroll-2004') FROM('John Doe')

* create the wage container on the payroll-2004 channel
EXEC CICS PUT CONTAINER('wage') CHANNEL('payroll-2004') FROM('100')

* invoke the payroll service, passing the payroll-2004 channel
EXEC CICS LINK PROGRAM('PAYR') CHANNEL('payroll-2004')

* examine the status returned on the payroll-2004 channel
EXEC CICS GET CONTAINER('status') CHANNEL('payroll-2004') INTO(stat)
```

図 75. クライアント・プログラムがチャネルを構成して、サーバー・プログラムに渡し、サーバーの出力を取得する方法

309 ページの図 76 では、クライアントが起動する PAYR サーバー・プログラムの一部を示します。サーバー・プログラム:

1. サーバー・プログラムを起動するチャネルを照会します。
2. EXEC CICS GET CONTAINER コマンドを実行して、payroll-2004 チャネルの employee および wage の各コンテナーから入力を取得します。
3. 入力データを処理します。
4. EXEC CICS PUT CONTAINER コマンドを実行して、payroll-2004 チャネルの status コンテナー内に出力を返します。

「PAYR」、CICS COBOL サーバー・プログラム

```
* discover which channel I've been invoked with
EXEC CICS ASSIGN CHANNEL(ch_name)
:
WHEN ch_name 'payroll-2004'
    * my current channel is "payroll-2004"
    * get the employee passed into this program
    EXEC CICS GET CONTAINER('employee') INTO(emp)
    * get the wage for this employee
    EXEC CICS GET CONTAINER('wage') INTO(wge)
    :
    * process the input data
    :
    :
    * return the status to the caller by creating the status container
    * on the payroll channel and putting a value in it
    EXEC CICS PUT CONTAINER('status') FROM('OK')
    :
    :
WHEN ch_name 'payroll-2005'
    * my current channel is "payroll-2005"
    :
    :
    :
```

図 76. サーバー・プログラムが渡されたチャネルを照会して、データをチャネルのコンテナーから取得し、出力を呼び出し元に戻す方法

チャネルと BTS アクティビティー

構築およびチャネルとの対話に使用される PUT、GET、MOVE、および DELETE CONTAINER コマンドは、CICS business transaction services (BTS) アプリケーションで使用されるコマンドと類似しています (BTS については、「*CICS Business Transaction Services*」マニュアルを参照してください)。そのため、BTS に関して経験のあるプログラマーにとっては、非 BTS アプリケーションでも容易にコンテナーが使用できます。さらに、コンテナーを使用するサーバー・プログラムは、チャネルと BTS アプリケーションの両方から呼び出すことができます。この例は 310 ページの図 77 に示されています。

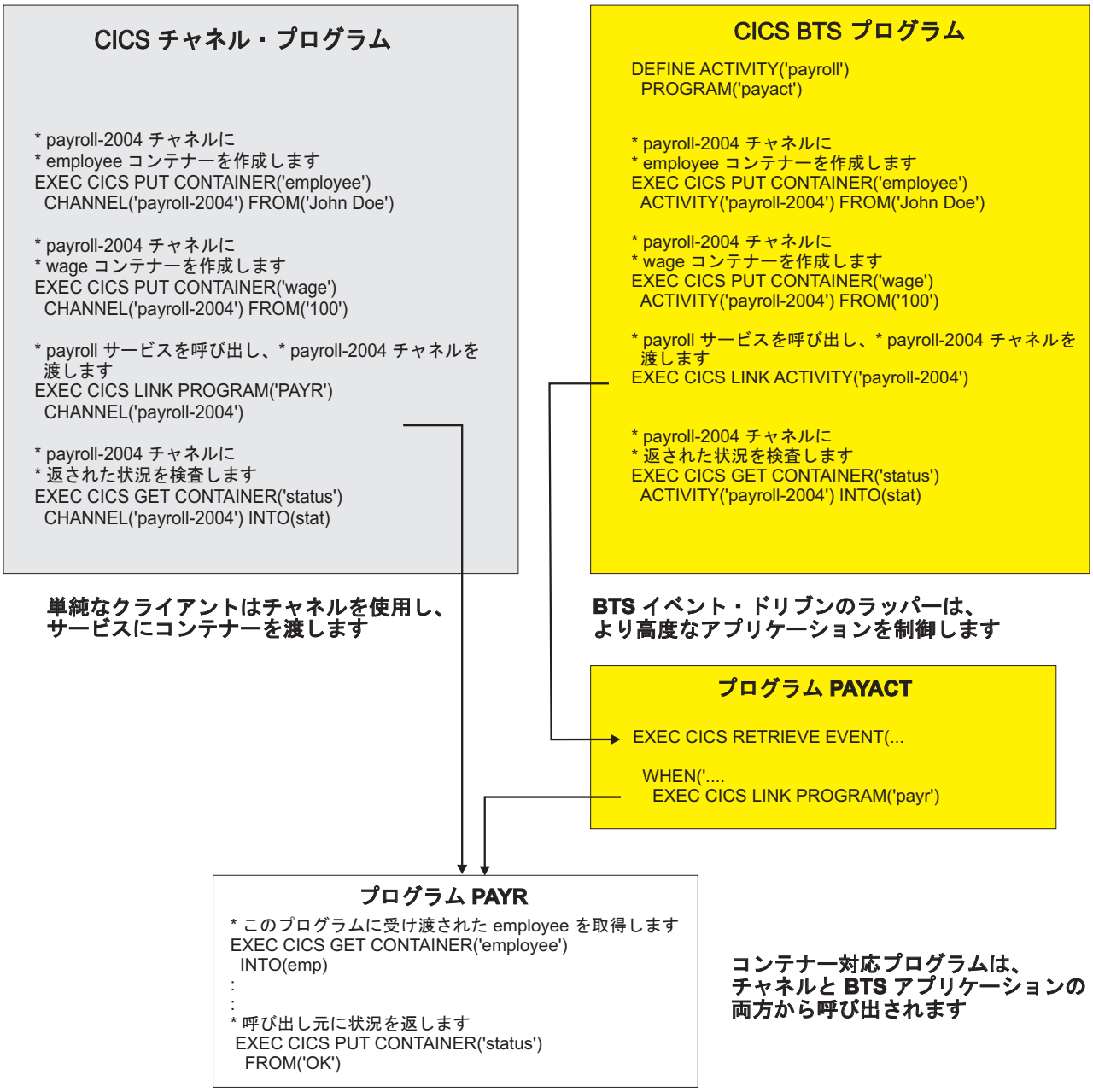


図 77. チャネルと BTS アクティビティー

コンテキスト

図 77 で示すように、コンテナー・コマンドを実行するプログラムを、変更せずに、チャネル・アプリケーションの一部、または BTS アクティビティーの一部として使用できます。

チャネルおよび BTS コンテキストの両方で使用されるプログラムの場合、実行するコンテナー・コマンドでは、それらのコマンドをチャネルまたは BTS コマンドのいずれかとして識別するオプションを指定しないでください。各コンテナー・コマンドで回避するオプションは、以下のとおりです。

DELETE CONTAINER

ACQACTIVITY (BTS 固有)
ACQPROCESS (BTS 固有)
ACTIVITY (BTS 固有)
CHANNEL (チャネル固有)
PROCESS (BTS 固有)

GET CONTAINER

ACQACTIVITY (BTS 固有)
ACQPROCESS (BTS 固有)
ACTIVITY (BTS 固有)
CHANNEL (チャネル固有)
INTOCCSID (チャネル固有)
PROCESS (BTS 固有)

MOVE CONTAINER

FROMACTIVITY (BTS 固有)
CHANNEL (チャネル固有)
FROMPROCESS (BTS 固有)
TOACTIVITY (BTS 固有)
TOCHANNEL (チャネル固有)
TOPROCESS (BTS 固有)

PUT CONTAINER

ACQACTIVITY (BTS 固有)
ACQPROCESS (BTS 固有)
ACTIVITY (BTS 固有)
CHANNEL (チャネル固有)
DATATYPE (チャネル固有)
FROMCCSID (チャネル固有)
PROCESS (BTS 固有)

コンテナー・コマンドが実行されると、CICS はコマンドが実行されたコンテキスト (チャネル、BTS、またはそれ以外) を分析して、コマンドの処理方法を決定します。コンテキストを判別するために、CICS は以下の順序で検査を行います。

1. チャネル: プログラムが現行チャネルを持っているかどうか。
2. BTS: プログラムが BTS アクティビティーの一部であるかどうか。
3. それ以外: プログラムは現行チャネルを持たず、BTS アクティビティーの一部でもありません。このため、コンテナー・コマンドを実行するコンテキストはありません。コマンドは INVREQ 状態および RESP2 値 4 でリジェクトされます。

JCICS からのチャネルの使用

JCICS を利用したチャネルの使用については、「*Java Applications in CICS*」のチャネルおよびコンテナーを参照してください。

チャネルを使用した動的ルーティング

EXEC CICS LINK および EXEC CICS START コマンドは、COMMAREA またはチャネルのいずれかを受け渡し、動的にルーティングされます。したがって、以下のタイプのチャネルに関連する要求が動的にルーティングできます。

- プログラム・リンク (DPL) の要求
- 端末関連の START 要求によって開始されたトランザクション
- 非端末関連の START 要求

ルーティング・プログラムは、その通信域であるチャネル名の DYRCHANL フィールドに受け渡され、プログラム・リンクまたは START コマンドがある場合は、それに関連付けられます。DYRCHANL フィールドは、上記の 3 つのタイプの要求に対してのみ適用されます。その他のタイプの要求の場合、またはその要求に関連付けられたチャネルがない場合、そのフィールドにはブランクが入ります。

注: ルーティング・プログラムの通信域は、DFHDYPDS DSECT によりマップされます。

ルーティング・プログラムにはチャネルのアドレスではなく*名前* が指定されることに注意してください。そのため、DYRCHANL フィールドを使用してそのコンテナーの内容を検査または変更することはできません。

LINK または START コマンドがチャネルではなく COMMAREA を渡す場合、ルーティング・プログラムは、要求のタイプに応じて COMMAREA の内容を検査または変更します。端末関連の START 要求 (*動的* ルーティング・プログラムにより処理される) によって開始された LINK 要求およびトランザクションの場合には、アプリケーションの COMMAREA *のアドレス* である DFHDYPDS の DYRACMAA フィールドにルーティング・プログラムが指定され、その内容の検査および変更が可能ですが、非端末関連の START 要求 (*分散* ルーティング・プログラムで処理される) の場合はこれが可能ではありません。

ルーティング・プログラムにチャネルを使用した場合と同じ種類の機能を与えるために、チャネルを使用するアプリケーションは、チャネル内で DFHROUTE という名前の特殊なコンテナーを作成できます。アプリケーションが LINK 要求、または動的にルーティングされる端末関連の START 要求 (しかし、非端末関連の START 要求ではない) 場合、動的ルーティング・プログラムに、DFHDYPDS の DYRACMAA フィールドで DFHROUTE コンテナーのアドレスが指定され、内容を検査および変更できます。

COMMAREA ではなくチャネルを受け渡すプログラムをマイグレーションする場合は、その既存の COMMAREA 構造を使用して DFHROUTE にマップできます。

関連情報

CICS 動的ルーティングの概要

動的ルーティング・プログラムの書き込み

データ変換

データ変換が必要な理由

データ変換が必要なケースは、以下のとおりです。

- 文字データが、EBCDIC と ASCII など、異なるエンコード規格を使用するプラットフォーム間で受け渡される場合。
- いくつかの文字データのエンコードを、あるコード化文字セット ID (CCSID) から別のコード化文字セット ID (CCSID) に変更したい場合。CCSID の説明、および CICS のサポートする CCSID のリストについては、「*CICS Family: Communicating from CICS on zSeries*」を参照してください。

チャネルを使用したコード・ページ変換の準備

始める前に

Unicode データが含まれるコード・ページ変換を準備している場合、「*CICS Transaction Server for z/OS インストール・ガイド*」の Unicode データ変換の使用可能化に関する情報を参照することもできます。

このタスクについて

UTF-8 または UTF-16 のいずれかと EBCDIC および ASCII のコード・ページ間のデータの変換は、適切な変換イメージの選択に応じて異なります。Unicode の UTF-8 と UTF-16 の形式での間の変換もサポートされます。

これらのサービスを使用してサポートされるこれらの変換について詳しくは、「*z/OS Support for Unicode: 変換サービスの使用*」(SA88-8813) の付録 F を参照してください。これらの変換は、Unicode に制限されるものではなく、EBCDIC、ASCII および Unicode などの広い範囲の文字のエンコード間での変換が可能です。

注:

1. 例えば、WebSphere MQ トランスポートと共に使用されている、037 と 500 との間の変換は、CICS および MQ により使用される文字エンコードで小さな差異により引き起こされる EBCDIC から EBCDIC への変換です。
2. 各コード・ページの一部のポイントで、他のコード・ページに直接対応するものがないものがあることに注意する必要があります。EBCDIC 文字 NL がそれに当てはまります。Java と z/OS の変換サービスは、実行する変換で異なる場合があります。特定のポイントのガイダンスについて詳しくは、『技術情報』、および他のインターネットのディスカッションを参照してください。特定の環境でのより適切な変換について、プログラミング・コミュニティーで意見が分かれていることも、注目に値します。

CICS は、現在、z/OS 変換サービスを利用することにより、これらの文字変換をサポートしています。ただし、テーブルのセットを使用して以前のリリースの CICS が実行したこれらの変換は、同じ方法でサポートが継続されます。CICS TS 3.1 が、これらのテーブル経由ではサポートされない CCSID のペア間での変換を実行するよう命令された場合にのみ、z/OS サービスを使用して変換が試行されます。

必要な変換イメージが使用可能であることの保証

CICS アプリケーションの一部として使用される CCSID は、z/OS 変換イメージの保守を担当しているシステム・プログラマーが認識している必要があり、これにより、これらのアプリケーションが実行される CICS 領域で特定の変換が使用可能になります。

CCSID 1200 の処理

CICS は、CCSID 1200、1201、および 1202 を使用した UTF-16 データを含む変換をサポートしています。z/OS 変換サービスは、CCSID 1200 のビッグ・エンディアン・フォームでの使用を許可しますが、CCSID 1201 または 1202 のリトル・エンディアン・フォームのサポートは含まれません。CICS は、これらのサポートされないフォームで認識されるソース・データを、1200 のビッグ・エンディアン・フォームに変換してから、データを変換のために z/OS に渡します。ターゲット・データがサポートされないフォームの 1 つである場合、CICS はデータを 1200 のビッグ・エンディアン・フォームとして受け取り、必要な CCSID に変換します。ターゲット CCSID が 1200 である場合、CICS はエンコードがビッグ・エンディアン・フォームであると見なします。変換がこれらの CCSID のいずれかの間で行われる場合、CICS は z/OS 変換サービスを呼び出さずに変換を実行します。

UTF-16 のこれらのフォームを含む変換に対して z/OS 変換イメージを設定する場合、CCSID 1200 を指定する必要があります。変換イメージを作成しようとする際に、CCSID 1201 および 1202 は z/OS により認識されません。

CICS は、インバウンド変換に対してバイト・オーダー・マーカーを尊重しますが、関連アウトバウンド変換を処理するときに情報を保存できません。CCSID 1200 のすべてのアウトバウンド・データは、UTF16-BE です。アプリケーション・プログラマーはこのことについて確認し、必要に応じて BE から LE への変換を実行する必要があります。

変換イメージの共用

- APAR OA05744 の PTF が適用されていない場合は、CICS で使用することを意図した、z/OS イメージにインストールされたこれらの変換に対して検索順序を指定しないでください。
- COBOL に対して同じ変換が必要な場合は、以下の 2 つの別のステートメントを使用して、変換イメージを定義する必要があります。
  - 検索順序を指定しないステートメント、および
  - 「RECLM」の検索順序を明示的に指定したステートメント

  例:

```
CONVERSION 850,037;
CONVERSION 850,037,RECLM;
```

APAR がインストールされていると、CICS および COBOL は、暗黙的または明示的にデフォルトの検索順序を指定したこれらのサポートされる変換を利用できるようになるので、イメージ生成ファイルで 2 つの制御ステートメントを指定する必要がなくなります。

JAVA プログラム
コード・ページ変換機能は Java 内に存在するため、CICS でこれらを重複させる必要はありません。ここで説明した変換機能は、Java プログラムを拡張しません。例については、「*Java Applications in CICS*」の コンテナーへのデータの入力のトピックを参照してください。

チャネルを使用したデータ変換

チャネルを使用してデータを交換するアプリケーションでは、簡単なデータ変換モデルが使用されます。一般的に変換は必要ありませんが、必要な場合は、単一のプログラミング命令を使用して、それを自動的に処理するよう CICS に指示できます。

以下のことに注意してください。

- 通常、(非 Java の) CICS TS プログラムが別の (非 Java) CICS TS プログラムを呼び出す場合、両方のプログラムで EBCDIC エンコードを使用しているため、データ変換は必要ありません。例えば、CICS TS C 言語プログラムが CICS TS の COBOL プログラムを呼び出して、文字データを含む一部のコンテナーを渡す場合、データ変換を使用する唯一の理由は、データの CCSID を変更するという一般的でない処理のためです。
- チャネルのアプリケーションで使用されるデータ変換モデルは、COMMAREA のアプリケーションによって使用されるモデルより単純です。データの交換に COMMAREA を使用するアプリケーションが従来のデータ変換モデルを使用する方法については、「*CICS Family: Communicating from CICS on System/390®*」資料で説明されています。変換は、システム・プログラマーによる制御のもと、DFHCNV 変換テーブル、DFHCCNV 変換プログラム、およびオプションで DFHUCNV ユーザー置換可能変換プログラムを使用して実行されます。

  これに対して、チャネル・コンテナー内のデータは、アプリケーション ・プログラマーの制御のもとで、API コマンドを使用して変換されます。
- アプリケーション・プログラマーは、ユーザー・データ (すなわち、そのアプリケーション・プログラムによって作成されたコンテナー内のデータ) の変換のみを担当します。システム・データは、必要に応じて CICS により自動的に変換されます。
- アプリケーション・プログラマーは、文字データの変換のみを考慮します。バイナリー・データの変換 (ビッグ・エンディアンとリトル・エンディアンの間の変換) は、サポートされていません。
- アプリケーションでは、文字データを 1 つのコード・ページから別のコード・ページに変換するための簡単な手段として、コンテナー API を使用することができます。

データ変換を目的として、CICS は 2 つのタイプのデータを識別します。

CHAR 文字データ、つまりテキスト・ストリング。コンテナーのデータは、(必要に応じて) 取得元のアプリケーションのコード・ページに変換されます。データを取得するアプリケーションが ASCII ベース・システムのクライアントである場合、ASCII コード・ページになります。CICS Transaction Server for z/OS アプリケーションである場合、EBCDIC コード・ページになります。

コンテナー内のすべてのデータが、単一の文字ストリングとして変換されます。1 バイト文字セット (SBCS) コード・ページでは、複数の文字フィールドから構成される構造が、1 つの 1 バイト文字ストリングに相当します。ただし、2 バイト文字セット (DBCS) コード・ページでは異なります。DBCS コード・ページを使用して、データ変換が必ず正常に動作するようにするには、各文字ストリングを別のコンテナーに格納する必要があります。

BIT すべての非文字のデータ、つまり、CHAR 型として指定されないすべてのデータ。コンテナー内のデータは変換できません。これはデフォルト値です。

コンテナー内のデータのデータ変換に対してコード・ページを指定するには、以下の 2 つの方法があります。

- コード化文字セット ID (CCSID) として指定する。CCSID は特定のコード・ページを識別する 10 進数です。例えば、ASCII 文字セット ISO 8859-1 の CCSID は 819 です。
- コード・ページの IANA 登録文字セット名として指定する。この名前は、HTTP ヘッダーの charset= values で指定できる英数字の名前です。例えば、ISO 8859-1 に対して CICS がサポートする IANA 文字セット名は、`iso-8859-1` および `iso_8859-1` です。

アプリケーション・プログラマーがデータ変換用のコード・ページを指定しなかった場合、CICS は、ローカルの CICS 領域全体に対してデフォルト・コード・ページを使用します。デフォルトのコード・ページは、LOCALCCSID システム初期設定パラメーターで指定されています。

データ変換で使用される API コマンドは、以下のとおりです。

- ```
  EXEC CICS PUT CONTAINER [CHANNEL] [DATATYPE] [FROMCCSID | FROMCODEPAGE]
  ```
- ```
  EXEC CICS GET CONTAINER [CHANNEL] [INTOCCSID | INTOCODEPAGE]
  ```

CICS が自動的に文字データを変換する方法

このタスクについて

1. **クライアント・プログラム**では、PUT CONTAINER コマンドの DATATYPE(DFHVALUE(CHAR)) オプションを使用して、コンテナーが文字データを保持していること、また、データが変換に対して適格であることを指定します。 例:

   ```
   EXEC CICS PUT CONTAINER(cont_name) CHANNEL('payroll')
                FROM(data1) DATATYPE(DFHVALUE(CHAR))
   ```

   データがクライアント・プラットフォームのデフォルトの CCSID である場合は、FROMCCSID または FROMCODEPAGE オプションを指定する必要はありません。デフォルトの CCSID ではない場合に、このオプションを指定してください (CICS TS 領域では、デフォルトの CCSID は LOCALCCSID システム初期化パラメーターで指定されます)。デフォルトの CCSID は暗黙指定されます。
2. **サーバー・プログラム**では、GET CONTAINER コマンドを実行して、データをプログラムの現行チャネルから取得します。

   ```
   EXEC CICS GET CONTAINER(cont_name) INTO(data_area1)
   ```

データはサーバー・プラットフォームのデフォルトの CCSID で返されます。データをデフォルト以外の CCSID に変換する場合のみ、INTOCCSID または INTOCODEPAGE オプションを指定してください。クライアントとサーバー・プラットフォームが異なる場合、データ変換は自動的に実行されます。

3. **サーバー・プログラム**では、PUT CONTAINER コマンドを実行して、値をクライアントに返します。

```
EXEC CICS PUT CONTAINER(status) FROM(data_area2)
              DATATYPE(DFHVALUE(CHAR))
```

DATATYPE(DFHVALUE(CHAR)) オプションは、コンテナーが文字データを保持していること、また、データが変換に対して適格であることを指定します。データがサーバー・プラットフォームのデフォルトの CCSID である場合は、FROMCCSID または FROMCODEPAGE オプションを指定する必要はありません。デフォルトの CCSID ではない場合に、このオプションを指定してください。

4. **クライアント・プログラム**では、GET CONTAINER コマンドを実行して、サーバー・プログラムにより返された状況を取得します。

```
EXEC CICS GET CONTAINER(status) CHANNEL('payroll')
              INTO(status_area)
```

状況はクライアント・プラットフォームのデフォルトの CCSID で返されます。データをデフォルト以外の CCSID に変換する場合のみ、INTOCCSID または INTOCODEPAGE オプションを指定してください。クライアントとサーバー・プラットフォームが異なる場合、データ変換は自動的に実行されます。

タスクの結果

コンテナーを使用したコード・ページ変換

このタスクについて

アプリケーションでは、文字データを 1 つのコード・ページから別のコード・ページに変換するための簡単な手段として、コンテナー API を使用することができます。以下の例は、データを codepage1 から codepage2 に変換します。

```
EXEC CICS PUT CONTAINER(temp) DATATYPE(DFHVALUE(CHAR))
              FROMCCSID(codepage1) FROM(input-data)
EXEC CICS GET CONTAINER(temp) INTOCCSID(codepage2)
              SET(data-ptr) FLENGTH(data-len)
```

以下の例は、データをその領域のデフォルトの EBCDIC コード・ページから、指定された UTF8 コード・ページに変換します。

```
EXEC CICS PUT CONTAINER(temp) DATATYPE(DFHVALUE(CHAR))
              FROM(ebcdic-data)
EXEC CICS GET CONTAINER(temp) INTOCCSID(utf8_ccsid)
              SET(utf8-data-ptr) FLENGTH(utf8-data-len)
```

この方法でコンテナー API を使用する場合は、以下の点に注意してください。

- GET CONTAINER コマンドでは、変換される長さが明白でない限り、INTO ではなく SET オプションを使用する (変換されるデータの長さは、GET CONTAINER(*cont_name*) NODATA FLENGTH(*len*) コマンドを発行することによって取得することができます)。

- 10 進数の CCSID ではなくサポートされているコード・ページ用 IANA 文字セット名を指定する場合や、CCSID 英数字を指定する場合は、FROMCCSID オプションと INTOCCSID オプションの代わりに、FROMCODEPAGE オプションと INTOCODEPAGE オプションを使用する。
- ストレージのオーバーヘッドをなくすために、変換の終了後は、変換されたデータをコピーし、コンテナーを削除する。
- チャネルのシップを避けるために、一時チャネルを使用する。
- 全体から全体への変換はできません。つまり、指定されたコード・ページとそのチャネルのコード・ページがサポートされていない組み合わせだった場合は、コード・ページ変換エラーが発生します。

SOAP の例

CICS TS SOAP アプリケーションでは、以下のことを実行できます。

1. ソケットまたは WebSphere MQ メッセージ・キューから UTF-8 または UTF-16 メッセージを取得します。
2. メッセージをコンテナーに UTF-8 フォーマットで挿入します。
3. EBCDIC データ構造を同一チャネルの他のコンテナーに挿入します。
4. バックエンド AOR 上のハンドラー・プログラムへの分散プログラム・リンク (DPL) 呼び出しを実行して、チャネルを渡します。

CICS TS 上でも実行しているバックエンド・ハンドラー・プログラムは、EXEC CICS GET CONTAINER コマンドを使用して EBCDIC データ構造またはメッセージを取得できます。メッセージは UTF-8 または UTF-16 で取得するか、独自または領域の EBCDIC コード・ページで取得できます。同様に、EXEC CICS PUT CONTAINER コマンドを使用して、データをコンテナーに UTF-8、UTF-16、または EBCDIC で配置します。

領域の EBCDIC コード・ページのメッセージの 1 つを取得するために、ハンドラーは以下のコマンドを実行できます。

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg)
```

INTOCCSID および INTOCODEPAGE オプションが指定されていないため、メッセージ・データは領域の EBCDIC コード・ページに自動的に変換されます (このことは、チャネルのメッセージ・データを格納するために使用した PUT CONTAINER コマンドが、CHAR の DATATYPE を指定したことを前提としています。デフォルトである BIT の DATATYPE を指定した場合、変換は不可能です)。

一部の出力を領域の EBCDIC コード・ページで戻すために、ハンドラーは以下のコマンドを実行できます。

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg)
```

CHAR が指定されていないため、データ変換は許可されません。FROMCCSID および FROMCODEPAGE オプションが指定されていないため、メッセージ・データは領域の EBCDIC コード・ページで取得されます。

UTF-8 のメッセージの 1 つを取得するには、INTOCCSID または INTOCODEPAGE オプションを指定する必要があります。それにより、コード・ページを識別し、デ

ータが領域の EBCDIC コード・ページに自動変換されるのを回避する必要があります。ハンドラーは、以下のコマンドを実行できます。

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg) INTOCCSID(utf8)
```

この場合、utf8 は、フルワードとして定義された変数で、UTF-8 用のコード化文字セット ID (CCSID) である 1208 に初期設定されます。コード・ページに IANA 文字セット名を使用する場合は、INTOCCSID オプションの代わりに INTOCODEPAGE オプションを使用できます。

```
EXEC CICS GET CONTAINER(input_msg) INTO(msg) INTOCODEPAGE(utf8)
```

この場合、utf8 は長さが 56 の文字ストリングとして定義された変数で、「utf-8」に初期設定されます。

一部の出力を UTF-8 で戻すために、サーバー・プログラムは以下のコマンドを実行できます。

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg) FROMCCSID(utf8)
```

または、代わりに以下のコマンドを実行できます。

```
EXEC CICS PUT CONTAINER(output) FROM(output_msg) FROMCODEPAGE(utf8)
```

ここで、変数 utf8 は、INTOCCSID および INTOCODEPAGE と同じ方法で定義および初期設定されます。FROMCCSID または FROMCODEPAGE オプションは、メッセージ・データが現在 UTF-8 フォーマットであることを指定します。FROMCCSID または FROMCODEPAGE が指定されているため、CHAR の DATATYPE は暗黙指定され、データ変換が許可されます。

チャネルの利点

チャネル / コンテナー・モデルには、従来の CICS プログラムがデータの交換に使用していた通信域 (COMMAREA) と比べて、幾つかの利点があります。例:

- COMMAREA と違い、チャネルのサイズは 32KB に制限されない。チャネルに追加できるコンテナーの数に制限はなく、個々のコンテナーのサイズにも、使用可能なストレージの量以外に制限はありません。

  大きなコンテナーを多数作成したために、他のアプリケーションが使用できるストレージの量が制限されることがないよう注意してください。
- チャネルには複数のコンテナーを含めることができるため、チャネルを使用することで、より構造化された方法でデータを渡すことができる。これに対して、COMMAREA は一体構造のデータ・ブロックです。
- COMMAREA とは異なり、チャネルは、戻されるデータの正確なサイズを知るためのプログラムを必要としない。
- チャネルは CICS プログラム間でのデータ交換の標準メカニズムである。チャネルは、LINK、START、XCTL、および RETURN コマンドで渡すことができます。分散プログラム・リンク (DPL) がサポートされているため、START CHANNEL および RETURN TRANSID コマンドによって開始されるトランザクションはリモートであっても構いません。
- チャネルは、CICS がサポートしているどの言語で作成された CICS アプリケーション・プログラムからでも使用できる。例えば、ある CICS 領域上の Java ク

ライアント・プログラムが、チャネルを使用して、バックエンド AOR 上の COBOL サーバー・プログラムとデータを交換することができます。

- 複数のチャネルを処理するサーバー・プログラムを作成することができる。このプログラムは、以下を行うことができます。
  1. 一緒に呼び出されたチャネルの動的な検出。
  2. チャネル内のコンテナーの参照。
  3. 渡されたチャネルに応じた処理の変更。
- 1 つ以上のチャネルを通じて呼び出された関連プログラムのセットから、「コンポーネント」を作成することができる。
- クライアントとコンポーネントの疎結合により、容易な展開が可能である。クライアントとコンポーネントは別々にアップグレードすることができます。例えば、まず先にコンポーネントをアップグレードして新規チャネルを処理してから、クライアント・プログラムをアップグレードして (または新規クライアント・プログラムを作成して) 新規チャネルを使用することが可能です。
- プログラマーはストレージ管理の悩みから解放される。CICS では、有効範囲から外れたコンテナー (およびそのストレージ) は自動的に破棄されます。
- チャネルのアプリケーションで使用されるデータ変換モデルは、COMMAREA のアプリケーションによって使用されるモデルより単純です。また、COMMAREA アプリケーションのデータ変換はシステム・プログラマーによって制御されますが、チャネル・アプリケーションのデータ変換は、アプリケーション・プログラマーが単純な API コマンドを使用して制御します。
- CICS Business Transaction Services (BTS) に関して経験のあるプログラマーにとっては、非 BTS アプリケーションでのコンテナーの使用も容易である。
- コンテナーを使用するプログラムは、チャネルと BTS アプリケーションの両方から呼び出し可能である。
- コンテナーを使用する非 BTS アプリケーションは、完全な BTS アプリケーションへのマイグレーションが可能である (それらのアプリケーションは BTS へのマイグレーション経路を形成します)。

このトピックでリストしているのは、数多くあるチャネルの利点の一部にすぎません。しかし、チャネルはあらゆる事例で最良のソリューションというわけではありません。アプリケーションを設計する際は、チャネルを使用することによって、以下のような影響が 1 つか 2 つ程度あることを理解しておく必要があります。

- チャネルがリモート・プログラムまたはトランザクションに渡される場合、大量のデータを渡すことによってパフォーマンスに影響が出る可能性がある。これは特に、ローカル領域とリモート領域が MRO 接続ではなく ISC 接続によって接続されている場合にあてはまります。
- チャネルは、同等のデータを渡すように設計された COMMAREA に比べて、より多くのストレージを使用する可能性がある。この理由は以下のとおりです。
  1. コンテナー・データは複数の場所に保持されている可能性がある。
  2. COMMAREA はポインターによってアクセスされるが、コンテナー内のデータはプログラム間でコピーされる。

COMMAREA からチャネルへのマイグレーション

従来の連絡域 (COMMAREA) を使用してデータを交換する CICS アプリケーション・プログラムは、以前と同様に動作することができます。この例は、新しい機能にマイグレーションする場合に、複数のタイプの既存アプリケーションをマイグレーションして、COMMAREA ではなくチャネルおよびコンテナーを使用できるようにする方法を示しています。

COMMAREA を単一のコンテナーを備えたチャネルに置き換えることができます。この置き換えは、COMMAREA からチャネルおよびコンテナーに移行する最も簡単な方法に見えますが、実際にはいい方法とは言えません。

また、同じデータを渡すために設計された場合でも、COMMAREA よりチャネルの方が、使用するストレージが大きい場合があることに注意してください (319 ページの『チャネルの利点』を参照してください)。

この新しい機能を活用するためにアプリケーション・プログラムを変更するには、時間と労力を必要とするため、チャネルおよびコンテナーに対して「最良事例」を実施する必要があります。詳しくは、 306 ページの『チャネルの設計: 最良事例』を参照してください。チャネルは COMMAREA よりいくつかの利点があり (319 ページの『チャネルの利点』を参照)、これらの改善点を最大限に利用できるようにチャネルを設計する価値があります。

チャネルを使用するためにアプリケーション・プログラムをマイグレーションしない場合でも、ワークロード管理のために CICSPlexSM ではなく、ユーザー作成の動的または分散ルーティング・プログラムを使用するには、DFHDYPDS 連絡域の DYRTYPE フィールドで渡されることのある新しい値を処理するようにプログラムを変更する必要があります。詳細については、「*CICS Customization Guide*」の動的ルーティング・プログラムに渡されるパラメーターを参照してください。

COMMAREA を渡す LINK コマンドのマイグレーション

このタスクについて

構造を交換するために LINK コマンドで COMMAREA を使用する 2 つのプログラムをマイグレーションするには、表 16 で説明する指示を変更します。

表 16. COMMAREA を渡す LINK コマンドのマイグレーション

プログラム	変更前	変更後

PROG1	`EXEC CICS LINK PROGRAM(PROG2)`
	`COMMAREA(structure)`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`CHANNEL(channel-name) FROM(structure)`
		`EXEC CICS LINK PROGRAM(PROG2)`
		`CHANNEL(channel-name)`
		⋮
		`EXEC CICS GET CONTAINER(structure-name)`
		`CHANNEL(channel-name) INTO(structure)`

表 16. COMMAREA を渡す LINK コマンドのマイグレーション (続き)

プログラム	変更前	変更後
PROG2	EXEC CICS ADDRESS COMMAREA(structure-ptr) ... RETURN	EXEC CICS GET CONTAINER(structure-name) INTO(structure) ... EXEC CICS PUT CONTAINER(structure-name) FROM(structure) RETURN

注: COMMAREA の例の PROG2 では、データを COMMAREA に挿入すると、RETURN コマンドを実行してデータを PROG1 に返すだけで済みます。チャネルの例では、データを返すには、*PROG2 は RETURN の前に PUT CONTAINER コマンドを実行する必要があります。*

COMMAREA を渡す XCTL コマンドのマイグレーション

このタスクについて

構造を渡すために XCTL コマンドで COMMAREA を使用する 2 つのプログラムをマイグレーションするには、表 17 で説明する指示を変更します。

表 17. COMMAREA を渡す XCTL コマンドのマイグレーション

プログラム	変更前	変更後
PROG1	EXEC CICS XCTL PROGRAM(PROG2) COMMAREA(structure)	EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS XCTL PROGRAM(PROG2) CHANNEL(channel-name) ⋮
PROG2	EXEC CICS ADDRESS COMMAREA(structure-ptr) ...	EXEC CICS GET CONTAINER(structure-name) INTO(structure) ...

RETURN コマンドでの疑似会話型 COMMAREA のマイグレーション

このタスクについて

疑似会話型の一部として構造を交換するために COMMAREA を使用する 2 つのプログラムをマイグレーションするには、表 18 で説明する指示を変更します。

表 18. RETURN コマンドでの疑似会話型 COMMAREA のマイグレーション

プログラム	変更前	変更後
PROG1	EXEC CICS RETURN TRANSID(PROG2) COMMAREA(structure)	EXEC CICS PUT CONTAINER(structure-name) CHANNEL(channel-name) FROM(structure) EXEC CICS RETURN TRANSID(TRAN2) CHANNEL(channel-name)

表 18. RETURN コマンドでの疑似会話型 COMMAREA のマイグレーション (続き)

プログラム	変更前	変更後
PROG2	`EXEC CICS ADDRESS COMMAREA(structure-ptr)`	`EXEC CICS GET CONTAINER(structure-name)` `INTO(structure)`

START データのマイグレーション

このタスクについて

構造を交換するために START データを使用する 2 つのプログラムをマイグレーションするには、表 19 で説明する指示を変更します。

表 19. START データのマイグレーション

プログラム	変更前	変更後

PROG1	`EXEC CICS START TRANSID(TRAN2)`
	`FROM(structure)`
		`EXEC CICS PUT CONTAINER(structure-name)`
		`CHANNEL(channel-name) FROM(structure)`
		`EXEC CICS START TRANSID(TRAN2)`
		`CHANNEL(channel-name)`

PROG2	`EXEC CICS RETRIEVE INTO(structure)`	`EXEC CICS GET CONTAINER(structure-name)` `INTO(structure)`

PROG2 の新しいバージョンは、疑似会話型の例と同じであることに注意してください。

データを渡すために一時ストレージを使用するプログラムのマイグレーション

このタスクについて

以前のリリースでは、COMMAREA のサイズが 32K に制限され、チャネルが利用不可だったため、一部のアプリケーションで一時ストレージ・キュー (TSQ) を使用して 32K を超えるデータをプログラム間で受け渡していました。通常、この処理では TSQ から複数の読み書きが行われます。

チャネルを使用するためにこれらのアプリケーションの 1 つをマイグレーションする場合は、以下のことに注意してください。

- 既存のアプリケーションにより使用される TS キューが主ストレージにある場合、新しくマイグレーションされたアプリケーションのストレージ要件は、既存のアプリケーションの要件に似たものになります。
- 既存のアプリケーションにより使用される TS キューが補助ストレージにある場合、マイグレーションされたアプリケーションのストレージ要件は、既存のアプリケーションの要件より大きくなります。これは、コンテナー・データがディスクに書き込まれるのではなく、ストレージに保持されるためです。

動的にルーティングされたアプリケーションのマイグレーション

EXEC CICS LINK および EXEC CICS START コマンドは、COMMAREA またはチャネルのいずれかを受け渡し、動的にルーティングされます。

LINK または START コマンドがチャネルではなく COMMAREA を渡す場合、ルーティング・プログラムは、要求のタイプに応じて COMMAREA の内容を検査または変更します。非端末関連の START 要求 (*分散* ルーティング・プログラムにより処理されるもの) ではなく、LINK 要求および、端末関連の START 要求 (*動的* ルーティング・プログラムにより処理されるもの) により開始されるトランザクションの場合、ルーティング・プログラムに、連絡域の DYRACMAA フィールドでアプリケーションの COMMAREA *のアドレス* が指定され、内容を検査および変更できます。

注: ルーティング・プログラムの通信域は、DFHDYPDS DSECT によりマップされます。

動的にルーティングされた EXEC CICS LINK または START コマンドをマイグレーションして、COMMAREA ではなくチャネルを使用する場合、ルーティング・プログラムに、DFHDYPDS の DYRCHANL フィールドでチャネルの名前が渡されます。ルーティング・プログラムに渡されるのは、アドレスではなくチャネルの*名前*であるため、DYRCHANL フィールドを使用して、チャネルのコンテナーの内容を検査または変更できないことに注意してください。

ルーティング・プログラムにチャネルを使用した場合と同じ種類の機能を与えるために、チャネルを使用するアプリケーションは、チャネル内で DFHROUTE という名前の特殊なコンテナーを作成できます。アプリケーションが LINK 要求、または動的にルーティングされる端末関連の START 要求 (しかし、非端末関連の START 要求ではない) 場合、動的ルーティング・プログラムに、DFHDYPDS の DYRACMAA フィールドで DFHROUTE コンテナーのアドレスが指定され、内容を検査および変更できます。

COMMAREA ではなくチャネルを受け渡すプログラムをマイグレーションする場合は、その既存の COMMAREA 構造を使用して DFHROUTE にマップできます。

動的ルーティングおよび分散ルーティングの概要については、「*CICS 相互通信ガイド*」のCICS 動的ルーティングの概説を参照してください。動的および分散ルーティング・プログラムの作成方法については、「*CICS Customization Guide*」の動的ルーティング・プログラムの作成を参照してください。

第 21 章 プログラム制御

CICS プログラム制御機能は、CICS システムにおけるアプリケーション・プログラム間の制御の流れを管理します。

Java および C++

ここで説明するアプリケーション・プログラミング・インターフェースは、Java プログラムでは使用されない CICS API です。JCICS クラスを使用してプログラム制御サービスにアクセスする Java プログラムについて詳しくは、「*Java Applications in CICS*」および JCICS Javadoc 文書を参照してください。CICS C++ クラスを使用した C++ プログラムについて詳しくは、「*CICS C++ OO Class Libraries*」を参照してください。

プログラム制御コマンドで参照するアプリケーション・プログラムの名前は、CICS にプログラムとして定義しておかなければなりません。プログラム制御コマンドを使用すると、以下のことができます。

- 後で要求を出した側のプログラムに戻ることを前提として、アプリケーション・プログラムの 1 つを別のプログラムにリンクする (LINK コマンド)。このコマンドの COMMAREA、CHANNEL、および INPUTMSG オプションを使用すると、要求された側のアプリケーション・プログラムにデータを渡すことができます。
- 後で要求を出した側のプログラムに戻ることを前提として、アプリケーション・プログラムの 1 つを別の CICS 領域の別のプログラムにリンクする (LINK コマンド)。このコマンドの COMMAREA または CHANNEL オプションを使用すると、要求された側のアプリケーション・プログラムにデータを渡すことができます。これは、分散プログラム・リンク (DPL) と呼ばれています。 (DPL を使用する場合、LINK コマンドの INPUTMSG および INPUTMSGLEN オプションは使用できません。DPL について詳しくは、407 ページの『第 27 章 CICS の相互通信』を参照してください。
- 要求を出した側のプログラムに戻らないで、アプリケーション・プログラムの 1 つから別のプログラムに制御権を移動する (XCTL コマンド)。このコマンドの COMMAREA、CHANNEL、および INPUTMSG オプションを使用すると、要求された側のアプリケーション・プログラムにデータを渡すことができます。(DPL を使用する場合、XCTL コマンドの INPUTMSG および INPUTMSGLEN オプションは使用できません。DPL について詳しくは、407 ページの『第 27 章 CICS の相互通信』を参照してください。
- アプリケーション・プログラムの 1 つから別のプログラムまたは CICS に制御権を返す (RETURN コマンド)。このコマンドの COMMAREA、CHANNEL、および INPUTMSG オプションを使用すると、新たに開始されるトランザクションにデータを渡すことができます。 (DPL を使用する場合、RETURN コマンドの INPUTMSG および INPUTMSGLEN オプションは使用できません。DPL についての詳細は、407 ページの『第 27 章 CICS の相互通信』を参照してください。)
- 指定したアプリケーション・プログラム、テーブル、またはマップを、主記憶装置にロードする (LOAD コマンド)。

© Copyright IBM Corp. 1989, 2009

プログラム、テーブル、または読み取り専用ではないマップをロードするために、LOAD および RELEASE コマンドと共に HOLD オプションを使用すると、動的トランザクション・ルーティングを実行する能力に不都合な影響を及ぼすトランザクション間の類縁性が生じることがあります。

これらのコマンドを発行するプログラムにおいて発生する可能性のある問題の識別を容易にするため、CICS Interdependency Analyzer を使用できます。このユーティリティーについて詳しくは「*CICS Interdependency Analyzer for z/OS User's Guide and Reference*」を参照し、トランザクションの類縁性について詳しくは 339 ページの『第 22 章 類縁性』を参照してください。

- 既にロードされているアプリケーション・プログラム、テーブル、またはマップを、主記憶装置から削除する (RELEASE コマンド)。

異常終了は、RESP オプションを使用して処理することができます。

プログラムのリンク

LINK コマンドは、ある論理レベルのアプリケーション・プログラムから次に低い論理レベルのアプリケーション・プログラムに制御を渡す場合に使用されます。

アプリケーション・プログラムの論理レベル

CICS の下で実行中のアプリケーション・プログラムは、各種の論理レベルで実行されます。タスク内で最初に制御を受け取るプログラムは、最高位の論理レベルにあります。アプリケーション・プログラムが別のプログラムにリンクした時に、最終的に制御を取り戻すことにしている場合には、リンクされるプログラムは次に低い論理レベルにあるものと見なされます。1 つのアプリケーション・プログラムから別のプログラムに制御を移動し、制御を返すことにしていない場合には、2 つのプログラムが同じ論理レベルにあるものと見なされます。

戻り操作が見込まれる別のプログラムへのリンク

制御を受けるプログラムがまだ主記憶装置にない場合には、プログラムがロードされます。リンク先プログラムにおいて RETURN コマンドが処理されると、次の順番の処理命令でそのリンクを開始したプログラムに制御が返されます。

リンクされるプログラムは、例外条件、アテンション ID、および異常終了の処理に関して、LINK コマンドを発行したプログラムとは独立して動作します。例えば、リンク元のプログラムの HANDLE コマンドの効果は、リンクされるプログラムに継承されませんが、リンク元のプログラムに戻った時に元の HANDLE コマンドが復元されます。HANDLE ABEND コマンドを使用すると、他のリンク・レベルの異常終了を処理することができます。 327 ページの図 78 は、論理レベルでの概念を示しています。

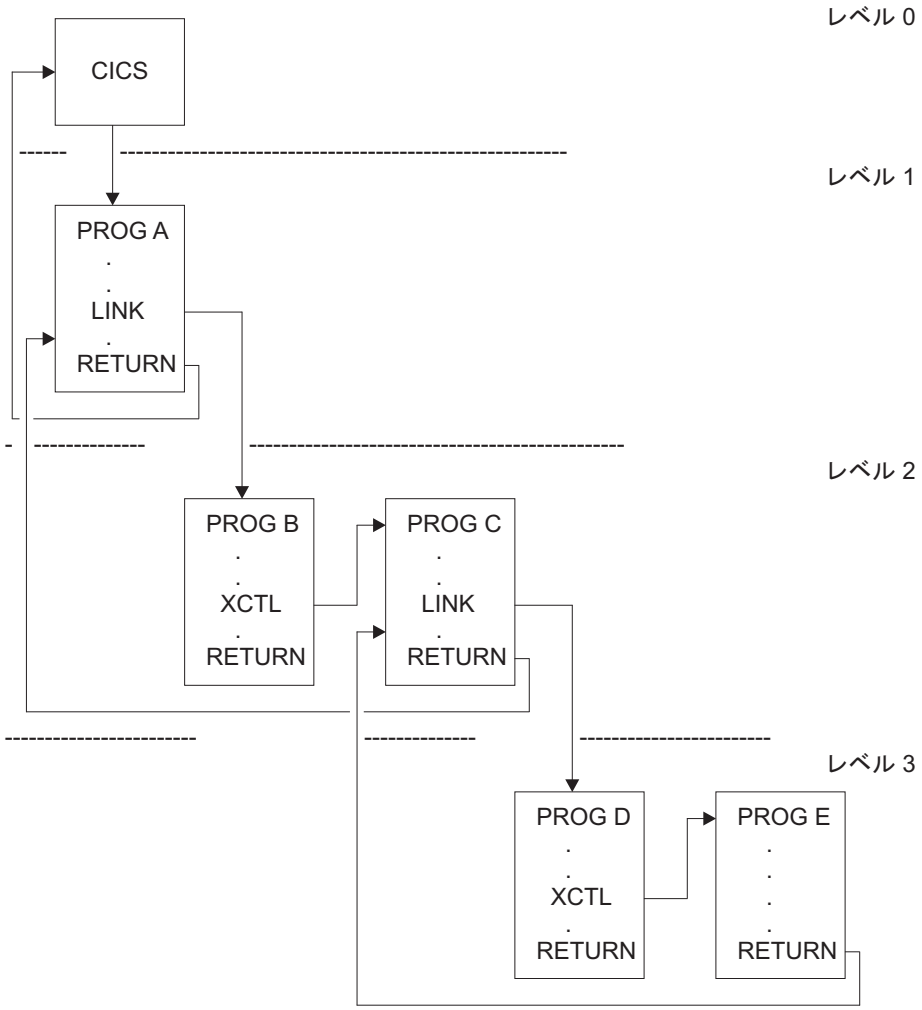


図 78. アプリケーション・プログラムの論理レベル

他のプログラムへのデータの受け渡し

このタスクについて

EXEC CICS プログラム制御コマンドである LINK、XCTL、および RETURN を使用し、そのコマンドに COMMAREA、CHANNEL、および INPUTMSG の各オプションを指定することにより、別のプログラムにデータを受け渡すことができます。COMMAREA と CHANNEL を同時に指定することはできません。

COMMAREA

LINK および XCTL コマンドの COMMAREA オプションは、呼び出されるプログラムにデータを渡すためのデータ域 (**連絡域**して知られている) の名前を指定します。

同様に、RETURN コマンドの COMMAREA オプションには、TRANSID オプションで識別されるトランザクションに渡すデータが入っている連絡域の名前を指定し

ます。 (TRANSID オプションには、タスクに関連付けられた端末から次の入力を受け取った時に開始されるトランザクションを指定します。)

呼び出されたプログラムは、データをパラメーターとして受け取ります。このプログラムには、渡されたデータにアクセスするためのデータ域の定義を含めなければなりません。

受取側の COBOL プログラムでは、データ域に DFHCOMMAREA という名前を付ける必要があります。この COBOL プログラムでは、あるプログラムが LINK、XCTL、または RETURN コマンドの一部として COMMAREA を渡す場合に、作業用ストレージと LINKAGE SECTION のどちらにデータ域を含めてもかまいません。COMMAREA を受け取るプログラムは、LINKAGE SECTION にデータを指定する必要があります。これは、以下のプログラムに適用されます。

- LINK コマンドまたは XCTL コマンドにおいて COMMAREA が渡される場合の受取側のプログラム
- 前に呼び出されたタスクの RETURN コマンドが COMMAREA および TRANSID を指定していた場合の、最初に呼び出されるプログラム

COMMAREA を受け取る C または C++ プログラムでは、COMMAREA は構造体に対するポインターとして定義しなければなりません。次に、プログラムは、ADDRESS COMMAREA コマンドを出して、渡されたデータをアドレッシング可能にしなければなりません。

PL/I プログラムでは、データ域に名前を持つことができますが、プログラムに渡されるパラメーターに基づいて、基底付き変数として宣言しなければなりません。この基底付き変数に対するポインターは、区域の宣言中に現れるときのコンテキストによる宣言ではなく、ポインターとして明示的に宣言しなければなりません。これは、PL/I エラー・メッセージの生成を防止します。このポインターに基づいて変数用に受取プログラム内で ALLOCATE ステートメントを処理することはできません。このポインターは、アプリケーション・プログラムによって更新してはなりません。

アセンブラー言語プログラムでは、データ域を DSECT にする必要があります。この DSECT をアドレッシングするのに使用されるレジスターは、DFHEISTG DSECT にある DFHEICAP からロードしなければなりません。

受取データ域は、元の連絡域と同じ長さである必要はありません。データの最初の部分だけにアクセスが必要である場合には、新しいデータ域は短くすることができます。ただし、新しいデータ域を渡される連絡域の長さより長くしてはなりません。それを超えた場合には、ユーザー・トランザクションは渡された区域の外側からデータを誤って読み取る可能性があります。さらに、区域の外側のデータに上書きする可能性もあり、CICS を異常終了させることにもなります。

このような事態を避けるためには、ユーザー・プログラムは、タスクの EIB の EIBCALEN フィールドをアクセスすることによって、プログラムに渡されている連絡域の長さが予定どおりであるかどうかを検査してください。連絡域が渡されていない場合には、EIBCALEN の値はゼロです。渡されている場合には、呼び出されたプログラムのデータ域のサイズに関係なく、EIBCALEN には、常に LINK、XCTL、または RETURN コマンドの LENGTH オプションに指定された値

が入ります。確実に EIBCALEN の値をユーザー・プログラムの DSECT の値と一致させて、ユーザー・トランザクションがその区域内のデータをアクセスしていることを確認してください。

渡されるデータの追加検査として、COMMAREA に ID を追加することもできます。この ID は、送出側のトランザクションで送られ、受取側のトランザクションによって検査されます。

LINK コマンドを使用して連絡域を渡す場合は、呼び出されたプログラムには連絡域自身を指すポインターが渡されます。呼び出されたプログラムがデータ域の内容を変更した場合、呼び出し側プログラムは、制御権が戻ってきた時点で、変更後の内容を使えるようになります。変更後の内容にアクセスするためには、呼び出し側プログラムは、元の COMMAREA オプションに指定していたデータ域の名前を指定します。

連絡域が XCTL コマンドを使用して渡される場合は、渡す区域のアドレスおよび長さがコマンドを発行したプログラムに渡された区域と同じ場合を除き、その区域のコピーが作成されます。例えば、プログラム A がプログラム B への LINK コマンドを発行し、次にプログラム B がプログラム C への XCTL コマンドを発行した場合に、A が B に渡した連絡域と同じ連絡域を B が C に渡すと、プログラム C は A に属する連絡域 (そのコピーではなく) へアドレッシング可能となり、C が行った変更は、制御権が A に戻ると A で利用可能になります。

LINK コマンドによってアクセスされていた下位レベル・プログラムが RETURN コマンドを発行した場合には、制御を返すプログラムより 1 つ論理レベルの高いレベルに制御が渡されます。タスクが端末と関連している場合には、TRANSID オプションを低位レベルで使用して、その端末と関連付ける次のトランザクションのトランザクション ID を指定することができます。そのトランザクション ID が有効になるのは、RETURN コマンドを使用して最高位の論理レベルが CICS に対する制御を解放し、入力が端末から受信された後のみです。端末から入力された入力は、アテンション・キー以外はすべて、全体がデータとして解釈されます。任意のリンク・レベルから戻る場合に COMMAREA なしで TRANSID オプションを使用できますが、その後の RETURN コマンドで指定変更されることがあります。無効な COMMAREA が原因で RETURN コマンドが最上位レベルで失敗した場合、TRANSID はヌルになります。また、最高位のレベルでは COMMAREA または IMMEDIATE だけを指定することができます。指定しない場合には、RESP2=2 のある INVREQ が入手されます。

さらに、COMMAREA オプションを使用して、新たに開始するタスクにデータを渡すこともできます。

呼び出されたプログラムは、EIB のフィールド EIBFN にアクセスすることによって、どのタイプのコマンドがそれを呼び出したのかを判別することができます。このフィールドは、CICS コマンドが発行される前にテストしなければなりません。このプログラムが LINK または XCTL コマンドによって呼び出された場合、該当するコードは EIBFN フィールドにあります。RETURN コマンドによって呼び出された場合は、このタスク内で CICS コマンドは発行されておらず、フィールドにはゼロが入ります。

チャネル

CICS プログラム間のデータ転送の最新の方法として、連絡域 (COMMAREA) を使用する代わりに、**チャネル**を使用します。チャネルには、COMMAREA に対するいくつかの利点があります。 319 ページの『チャネルの利点』を参照してください。 COMMAREA の代わりに、チャネルを LINK、XCTL、および RETURN の各コマンドで渡すことができます。

チャネルについては、 289 ページの『第 20 章 拡張プログラム間データ転送: 新しい COMMAREA としてのチャネル』で説明しています。

INPUTMSG

LINK、XCTL、および RETURN コマンドの INPUTMSG オプションは、呼び出されるプログラムに渡すデータ域の名前を指定するためのもう 1 つの方法です。この場合には、呼び出されたプログラムは RECEIVE コマンドを処理することによってデータを入手します。このオプションにより、端末から直接呼び出されるように作成されている (「フロントエンド」) アプリケーション・プログラムを呼び出し、RECEIVE コマンド含まれているプログラムで初期の端末入力を取得できます。

LINK コマンドを使用してアクセスされたプログラムが端末からの初期入力を取得するため RECEIVE コマンドを発行しても、最初の RECEIVE 要求が高水準プログラムによって既に発行されていた場合、プログラムが受け取るデータはありません。この場合には、アプリケーションは端末からの入力を待機します。INPUTMSG オプションでリンクされるプログラムを呼び出すことによって、元の端末入力を、リンクされるプログラムに対して継続して利用可能にすることができます。

アプリケーション・プログラムが別のプログラムを呼び出す場合は、LINK (または XCTL または RETURN) コマンドに INPUTMSG を指定すると、リンクされたプログラム自身が RECEIVE コマンドを出さずに、さらに別のアプリケーション・プログラムを呼び出していたとしても、INPUTMSG に指定したデータはそのプログラムで引き続き利用できます。 331 ページの図 79 には、INPUTMSG が図示されています。

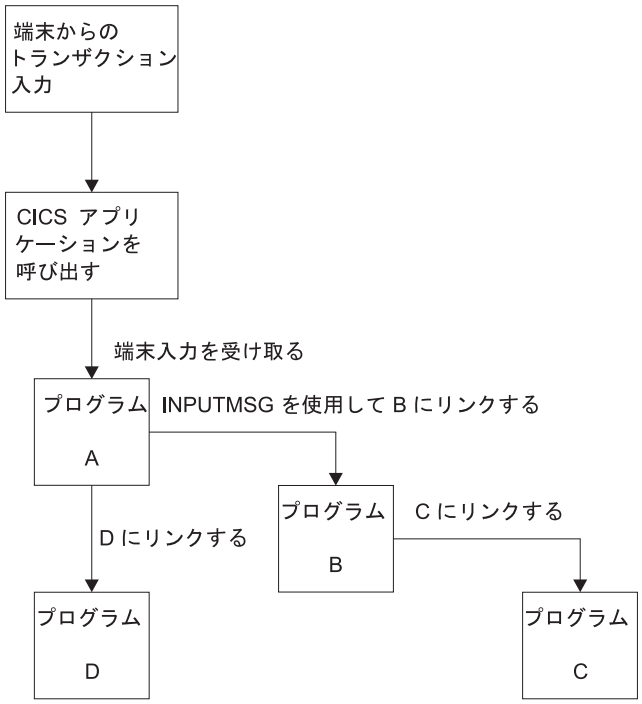


図 79. リンクされたチェーンでの INPUTMSG の使用

注:

1. この例では、「実際の」最初の RECEIVE コマンドがプログラム A によって発行されています。INPUTMSG オプションでプログラム B をリンクすることによって、次に RECEIVE 要求を発行するプログラムも端末入力を受け取ることができます。これはプログラム B またはプログラム C のいずれかとすることができます。
2. プログラム A は、受け取った端末入力を変更しないでそのまま渡す場合には、INPUTMSG オプションに、RECEIVE コマンドに使用したのと同じデータ域を指定することができます。例:

```
EXEC CICS RECEIVE
          INTO(TERMINAL-INPUT)

EXEC CICS LINK
          PROGRAM(PROGRAMB)
          INPUTMSG(TERMINAL-INPUT)
```

3. LINK チェーンの 1 つのプログラムが RECEIVE コマンドを発行すると同時に、INPUTMSG データはそれ以降の RECEIVE コマンドに対して使用可能ではなくなります。つまり、ここに示した例では、B が C にリンクする前に RECEIVE 要求を発行した場合には、INPUTMSG データ域が C に対して利用不能になります。
4. 1 つのプログラムから別のプログラムにデータを連絡していくこの方式は、どのような種類のデータにも使用できます。データがユーザー端末から発信されていなくてもかまいません。この例では、プログラム A が、名前の付いたデータ域にデータを移動させて、そのデータを参照する INPUTMSG を指定してプログラム B を呼び出す形になります。
5. INPUTMSG で渡される「端末データ」も、INPUTMSG を指定してリンクを発行したプログラムに最終的に制御が返されたときに利用できなくなります。この例

では、C が B に戻り、B が A に戻った場合には、B も C も RECEIVE コマンドを発行していなければ、データは A が受け取ったものと見なされます。その後で A が別のプログラム (例えば、D) を呼び出した場合には、INPUTMSG オプションを指定しない限り、元の INPUTMSG データを D で利用することはできません。

6. INPUTMSG データは、SEND または CONVERSE コマンドが発行された時に利用できなくなります。

RETURN コマンドでの INPUTMSG オプションの使用

INPUTMSG を指定すると、TRANSID オプションを使用した RETURN コマンドで指定した次のトランザクションにデータを受け渡すことができます。これを実行するには、RETURN は最高位の論理レベルで発行して CICS に制御を返す必要があり、コマンドでは IMMEDIATE オプションも指定する必要があります。TRANSID と一緒に INPUTMSG を指定して、IMMEDIATE を指定していない場合には、端末からの次の実際の入力が INPUTMSG データを指定変更するので、このデータが失われます。

SEND コマンド後少し間をおいて TRANSID と一緒に INPUTMSG を指定すると、SEND メッセージは即時に端末に表示されます。

TRANSID オプションを使用しない RETURN コマンドにおける INPUTMSG のその他の使用目的として、動的トランザクション・ルーティング・プログラムでの使用があります。ユーザーが変更可能な動的トランザクション・ルーティング・プログラムに関するプログラミング情報については、「*CICS Customization Guide*」の動的ルーティング・プログラムの作成を参照してください。

混合アドレッシング・モードの使用

CICS では、アドレッシング・モードが異なるプログラム間、およびアドレッシング・モードが同じプログラム間で、LINK、XCTL、および RETURN コマンドの使用をサポートしています。

COMMAREA オプションに指定した連絡域を使用してデータを渡すプログラムには、以下の制約があります。

- 連絡域内で AMODE(31) のプログラムに渡されるアドレスは、31 ビットの長さでなければなりません。フラグ・データを先頭バイトに入れた形の 3 バイト・アドレスは、呼び出されるプログラムが最上位バイトを無視するように特別に設計されている場合以外は、使用しないでください。
- AMODE(24) のプログラムにデータとして渡すアドレスは、呼び出されたプログラムで正しく解釈するためには、16MB の境界より下のアドレスでなければなりません。

これらの制約事項は、連絡域のアドレスにも、またその中のアドレスにも適用されます。しかし、16MB の境界より上の連絡域を、AMODE(24) のサブプログラムに渡すことができます。CICS は処理のために、16MB の境界より下の区域に連絡域をコピーします。CICS はリンク元プログラムに制御が返されるときに、連絡域を再びコピーして返します。CICS キーのストレージのコピーの詳細については、653 ページの『ストレージ制御』を参照してください。

CICS は、アドレッシング・モードが異なるプログラム間で連絡域に入れて渡されるデータ・アドレスの妥当性検査は行いません。

LINK を使用したデータの受け渡し

このタスクについて

図 80 から 335 ページの図 83 は、COBOL、C、C++、PL/I、およびアセンブラー言語において、LINK コマンドによりデータがリンク先のプログラムに受け渡される方法を示しています。XCTL コマンドの場合も同様の方法でコーディングできます。

COMMAREA を使用してデータが受け渡される例が以下に示されています。データの受け渡しにチャネルを使用する LINK コマンドの例については、 289 ページの『第 20 章 拡張プログラム間データ転送: 新しい COMMAREA としてのチャネル』を参照してください。

```
                    呼び出し側プログラム
IDENTIFICATION DIVISION.
PROGRAM ID. 'PROG1'.
.
WORKING-STORAGE SECTION.
01 COM-REGION.
    02 FIELD PICTURE X(3).
.
PROCEDURE DIVISION.
    MOVE 'ABC' TO FIELD.
    EXEC CICS LINK PROGRAM('PROG2')
         COMMAREA(COM-REGION)
         LENGTH(3) END-EXEC.
.
                     呼び出されるプログラム
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG2'.
.
LINKAGE SECTION.
01 DFHCOMMAREA.
    02 FIELD PICTURE X(3).
    .
    PROCEDURE DIVISION.
    IF EIBCALEN GREATER ZERO
    THEN
        IF FIELD EQUALS 'ABC' ...
```

図 80. COBOL の例 — LINK コマンド

```
                呼び出し側プログラム
main()
{
    unsigned char field[3];
    memcpy(field, "ABC", 3);
    EXEC CICS LINK PROGRAM("PROG2")
              COMMAREA(field)
              LENGTH(sizeof(field));
}
              呼び出されるプログラム
main()
{
    unsigned char *commarea;
    EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
    if (dfheiptr->eibcalen > 0)
    {
        if (memcmp(commarea, "ABC", 3) == 0)
        {
.
```

図 81. C の例 — LINK コマンド： この例では、COMMAREA は文字ストリングを含んでいます。構造体を含んでいる COMMAREA の例は、337 ページの図 85を参照してください。

```
                  呼び出し側プログラム
PROG1: PROC OPTIONS(MAIN);
DCL 1 COM_REGION AUTOMATIC,
      2 FIELD CHAR(3),
.
FIELD='ABC';
EXEC CICS LINK PROGRAM('PROG2')
     COMMAREA(COM_REGION) LENGTH(3);
END;

                  呼び出されるプログラム
PROG2:
PROC(COMM_REG_PTR) OPTIONS(MAIN);
DCL COMM_REG_PTR PTR;
DCL 1 COM_REGION BASED(COMM_REG_PTR),
      2 FIELD CHAR(3),
.
IF EIBCALEN>0 THEN DO;
    IF FIELD='ABC' THEN ...
.
    END;
END;
```

図 82. PL/I の例 — LINK コマンド

```
                    呼び出し側プログラム
DFHEISTG DSECT
COMREG   DS 0CL20
FIELD    DS CL3
.
PROG1    CSECT
.
         MVC FIELD,=C'XYZ'
         EXEC CICS LINK
         PROGRAM('PROG2')
         COMMAREA(COMREG) LENGTH(3)
.
         END
                     呼び出されるプログラム
COMREG   DSECT
FIELD    DS CL3
.
PROG2    CSECT
.
         L COMPTR,DFHEICAP
         USING COMREG,COMPTR
         CLC FIELD,=C'ABC'

         END
```

図 83. ASM の例 — LINK コマンド

RETURN を使用したデータの受け渡し

このタスクについて

336 ページの図 84 から 338 ページの図 87 は、COBOL、C、C++、PL/I、およびアセンブラー言語において、RETURN コマンドを使用して新規トランザクションにデータを受け渡す方法を示しています。

COMMAREA でデータが返される例が以下に示されています。データを返す場合にチャネルを使用する RETURN コマンドの例については、 289 ページの『第 20 章 拡張プログラム間データ転送: 新しい COMMAREA としてのチャネル』を参照してください。

呼び出し側プログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'PROG1'.
.
WORKING-STORAGE SECTION.
01  TERMINAL-STORAGE.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).
.
PROCEDURE DIVISION.
    MOVE 'XYZ' TO FIELD.
    EXEC CICS RETURN TRANSID('TRN2')
         COMMAREA(TERMINAL-STORAGE)
         LENGTH(20) END-EXEC.
.
```

呼び出されるプログラム

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  'PROG2'
.
LINKAGE SECTION.
01  DFHCOMMAREA.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).
.
PROCEDURE DIVISION.
    IF EIBCALEN GREATER ZERO
    THEN
        IF FIELD EQUALS 'XYZ'
            MOVE 'ABC' TO FIELD.
    EXEC CICS RETURN END-EXEC.
```

図 84. COBOL の例 — RETURN コマンド

呼び出し側プログラム

```
struct ter_struct
{
    unsigned char field[3];
    unsigned char datafld[17];
};
main()
{
    struct ter_struct ter_stor;
    memcpy(ter_stor.field,"XYZ",3);
    EXEC CICS RETURN TRANSID("TRN2")
              COMMAREA(&ter_stor)
              LENGTH(sizeof(ter_stor));
}
```

呼び出されるプログラム

```
struct term_struct
{
    unsigned char field[3];
    unsigned char datafld[17];
};
main()
{
    struct term_struct *commarea;
    EXEC CICS ADDRESS COMMAREA(commarea) EIB(dfheiptr);
    if (dfheiptr->eibcalen > 0)
    {
        if (memcmp(commarea->field, "XYZ", 3) == 0)
            memcpy(commarea->field, "ABC", 3);
    }
    EXEC CICS RETURN;
}
```

図 85. C の例 — RETURN コマンド

呼び出し側プログラム

```
PROG1: PROC OPTIONS(MAIN);
DCL 1 TERM_STORAGE,
      2 FIELD CHAR(3),
.
FIELD='XYZ';
EXEC CICS RETURN TRANSID('TRN2')
     COMMAREA(TERM_STORAGE);
END;
```

呼び出されるプログラム

```
PROG2:
PROC(TERM_STG_PTR) OPTIONS(MAIN);
DCL TERM_STG_PTR PTR;
DCL 1 TERM_STORAGE
       BASED(TERM_STG_PTR),
      2 FIELD CHAR(3),
.
IF EIBCALEN>0 THEN DO;
    IF FIELD='XYZ' THEN FIELD='ABC';
    END;
EXEC CICS RETURN;
END;
```

図 86. PL/I の例 — RETURN コマンド

呼び出し側プログラム

```
DFHEISTG DSECT
TERMSTG  DS 0CL20
FIELD    DS CL3
DATAFLD  DS CL17
:
:
PROG1    CSECT
:
:
         MVC FIELD,=C'ABC'
         EXEC CICS RETURN
         TRANSID('TRN2')
         COMMAREA(TERMSTG)
:
:
         END
```

呼び出されるプログラム

```
TERMSTG  DSECT
FIELD    DS CL3
DATAFLD  DS CL17
:
:
PROG2    CSECT
:
:
         CLC EIBCALEN,=H'0'
         BNH LABEL2
         L COMPTR,DFHEICAP
         USING TERMSTG,COMPTR
         CLC FIELD,=C'XYZ'
         BNE LABEL1
         MVC FIELD,=C'ABC'
LABEL1   DS 0H
:
:
LABEL2   DS 0H
:
:
         END
```

図 87. ASM の例 — RETURN コマンド

第 22 章 類縁性

CICS トランザクションおよびプログラムは、数多くの多彩な手法を使用して相互間でデータを渡します。それらの手法の一部は、データを交換するトランザクションやプログラムが、すべて同じ CICS 領域で実行されていることを必要とします。この結果、トランザクションおよび分散プログラム・リンク (DPL) 要求を動的にルーティングできる領域に、制限が加わります。複数のトランザクションまたはプログラムが、そのような制限が課される方法でデータを交換する場合、それらの間には類縁性があると言います。

Java

このトランザクション間の類縁性に関するガイダンスでは、EXEC CICS API を使用して作成されたアプリケーションについて説明します。しかし、説明の大部分は CICSplex で実行される Java アプリケーションおよび Enterprise Beans の場合も同様に有効です。Java アプリケーションおよび Enterprise Beans の開発に関するガイダンスについては、「*Java Applications in CICS*」を参照してください。

トランザクション、プログラム・リンク要求、EXEC CICS START 要求、CICS Business Transaction Services (BTS) のアクティビティー、および Enterprise Bean のメソッドの呼び出しはすべて、動的にルーティングできます。

動的 ルーティング・プログラムを使用して、以下のものを動的にルーティングすることができます。

- 端末から開始されるトランザクション
- 適格な端末関連の EXEC CICS START コマンドが開始するトランザクション
- CICS から CICS への適格な DPL 要求
- CICS 外部から受け取った適格なプログラム・リンク要求

分散 ルーティング・プログラムを使用して、以下のものを動的にルーティングすることができます。

- 適格な BTS のプロセスおよびアクティビティー。(BTS については、「*CICS Business Transaction Services*」の資料に説明されています。)
- 端末関連でない、適格な EXEC CICS START 要求

動的および分散のルーティングの概要情報の詳細については、「*CICS 相互通信ガイド*」のCICS 動的ルーティングの概説を参照してください。

重要:

以下のセクションにおける説明は、トランザクション間の類縁性にのみ適用されます。

- 類縁性はプログラム間にも存在する。(ただし厳密にいえば、類縁性があるのは、プログラムに関連したトランザクションだということができます。) この結果、プログラム・リンク要求がルーティングできる領域に、制限が課せられる場合があります。

© Copyright IBM Corp. 1989, 2009

- 安全なプログラミング手法、危険なプログラミング手法、危険性のあるプログラミング手法に関する各セクションは、トランザクションのルーティングだけではなく、プログラム・リンク要求および START 要求のルーティングにも適用される。

類縁性のタイプ

動的ルーティングに影響を与える類縁性には以下の 2 つのタイプがあります。

- トランザクション間の類縁性
- トランザクションとシステム間の類縁性

トランザクション間の類縁性

複数の CICS トランザクション間のトランザクション類縁性は、お互いに情報をやりとりするための手法を使用するトランザクション間、またはお互いのアクティビティーを同期化するための手法を使用するトランザクション間で生じます。その場合トランザクションは、同一の CICS 領域で実行される必要があります。このタイプの類縁性はトランザクション間の類縁性で、トランザクションの集合は、共通リソースの共用または処理の調整あるいはその両方を行います。トランザクション間の類縁性は、トランザクションの動的ルーティングに制限を加えるもので、次のような状況で生じる可能性があります。

- 2 番目のトランザクションが最初のトランザクションと同じ CICS 領域で実行している場合にのみアクセスできる場所に、最初のトランザクションが「状態データ」を残して終了した場合。
- 最初のトランザクションが、その最初のトランザクションが実行されている間に 2 番目のトランザクションがアクセスするデータを作成する場合。この作業を問題なく実行するために、最初のトランザクションは通常、何らかのイベントを待ちます。そのイベントは、最初のトランザクションが作成したデータを 2 番目のトランザクションが読み取ったときに、2 番目のトランザクションによって通知されます。この手法では、両方のトランザクションが同一の CICS 領域にルーティングされなければなりません。
- 2 つのトランザクションが、イベント制御ブロック (ECB) 機構を使用して同期する場合。CICS では、この手法についての機能シップ・サポートは行っていないので、このタイプの類縁性では、2 つのトランザクションは、同一の CICS 領域にルーティングされなければならないということになります。

  注: 2 つのトランザクションがエンキュー (ENQ) 機構を使用して同期する場合にも、同じことが言えます。**ただし**、適切な ENQMODEL リソース定義を使用して、シスプレックス全体を ENQ の有効範囲にする場合は別です。「*CICS Resource Definition Guide*」の ENQMODEL リソース定義を参照してください。

トランザクションとシステム間の類縁性

トランザクション間の類縁性ではない別のタイプのトランザクションの類縁性があります。それは、あるトランザクションと特定の CICS 領域との間の類縁性です。そこでは、トランザクションが CICS 領域とトランザクションとシステム間の類似性の特性を問い合わせたり、変更したりします。

他のトランザクションに対してではなく、特定のシステムに対して類縁性があるトランザクションは、動的ルーティングには適していません。一般に、それらのトランザクションは、INQUIRE および SET コマンドを使用するか、または特定の CICS 領域に対して類縁性があるグローバル・ユーザー出口プログラムに依存しています。

INQUIRE および SET コマンドとグローバル・ユーザー出口の使用

INQUIRE および SET コマンドには、リモート (つまり、機能シップ) のサポートがなく、SYSID オプションもないので、これらのコマンドを使用するトランザクションは、トランザクション自体が参照するリソースを所有する CICS 領域にルーティングされなければなりません。一般に、そのようなトランザクションは、どのターゲット領域に対しても動的にルーティングできないので、INQUIRE および SET コマンドを使用するトランザクションは、静的にルーティングしなければなりません。

異なる CICS 領域で実行されるグローバル・ユーザー出口ではデータを交換することはできません。ユーザー出口を使用してデータまたはパラメーターを渡すユーザー・トランザクションはまれですが、そのようなトランザクションが存在する場合は、グローバル・ユーザー出口と同一のターゲット領域で実行されなければなりません。

プログラミング手法と類縁性

動的または分散ルーティング環境における、トランザクション間の類縁性という見地からすると、アプリケーション・プログラムで使用するプログラミング手法は、大きく次の 3 つのカテゴリーに分類することができます。

- 一般に安全で、トランザクション間の類縁性の原因とならない手法
- 本質的に危険な手法
- 組み込まれる方法によって、類縁性の原因になる場合とならない場合があるという点で、危険性のある手法

安全な手法

一般的に安全なカテゴリーのプログラミング手法は次のとおりです。

- いくつかの CICS コマンドの CICS API によりサポートされている連絡域 (COMMAREA) の使用。しかし、トランザクションの類縁性に関して、動的または分散ルーティング環境において重要なのは、CICS RETURN コマンドの COMMAREA オプションだけです。というのは、疑似会話型トランザクションにおいて次のトランザクションに渡されるのは、RETURN コマンドの COMMAREA オプションだけであるためです。
- CICS に定義されている各端末で選択的に使用可能な TCT ユーザー域 (TCTUA) の使用。
- 次の CICS コマンドを使用したタスクの同期化または逐次化。
  - ENQ / DEQ。**ただし**、適切な ENQMODEL リソース定義を使用して、シスプレックス全体を ENQ の有効範囲にしている場合。ENQMODEL の説明については、346 ページの『ENQMODEL リソース定義による ENQ および DEQ コマンドの使用』および「*CICS Resource Definition Guide*」のENQMODEL リソース定義を参照してください。

- CICS Business Transaction Services (BTS) のアクティビティー間で、データの受け渡しをするための、コンテナーの使用。コンテナー・データは RLS が使用可能な BTS VSAM ファイルに保管されます。

COMMAREA および TCTUA の詳細については、343 ページの『類縁性を防止するプログラミング手法』を参照してください。

安全ではない手法

危険なカテゴリーのプログラミング手法は次のとおりです。

- 長寿命共用ストレージの使用
  - 共通作業域 (CWA)
  - GETMAIN SHARED ストレージ
  - LOAD PROGRAM HOLD を介して得られるストレージ
- 同期タスクが共用するタスク存続時間ローカル・ストレージの使用

  1 つのタスクから別のタスクにいくつかのタスク存続時間ストレージのアドレスを渡すことができます。

  アドレスの所有側タスクが終了しない限り、受信タスクが渡されたアドレスを使用しても安全です。好ましい手法ではありませんが、CICS タスク同期化手法を使用して、受信タスクがアドレスを受け取り終えるまで送信タスクが終了しない (または何か別の方法でストレージを解放しない) ようにすることができます。しかし、このような設計には、送信タスクが外部の処理によって除去される可能性があるので、危険をともないます。

  詳細については、353 ページの『タスク存続期間ストレージの共用』を参照してください。
- 次の CICS コマンドを使用したタスクの同期化または逐次化。
  - WAIT EVENT / WAIT EXTERNAL / WAITCICS
  - ENQ / DEQ、**ただし**、適切な ENQMODEL リソース定義を使用して、システム全体を ENQ の有効範囲にする場合は別。ENQMODEL の説明については、346 ページの『ENQMODEL リソース定義による ENQ および DEQ コマンドの使用』および「*CICS Resource Definition Guide*」 を参照してください。

危険なプログラミング技法の詳細については、349 ページの『類縁性を生じさせるプログラミング手法』を参照してください。

疑いのある手法

いくつかのプログラミング技法は、組み込まれ方によって類縁性の原因となるか、ならないかが決まります。例えば一時記憶域の使用がその例です。動的または分散ルーティング環境において、制限を加えなくても動作するかどうかを判別するためには、このカテゴリーに含まれる手法を使用するアプリケーション・プログラムを、検査する必要があります。危険性のあるカテゴリーのプログラミング手法は次のとおりです。

- 限定的命名規則に従った一時記憶キューの使用
- 一時データ・キューおよびトリガー・レベルの使用

- 次の CICS コマンドを使用したタスクの同期化または逐次化。
  - RETRIEVE WAIT / START
  - START / CANCEL REQID
  - DELAY / CANCEL REQID
  - POST / CANCEL REQID
- INQUIRE および SET コマンドならびにグローバル・ユーザー出口

危険性のあるプログラミング技法の詳細については、356 ページの『類縁性を生じさせる可能性のあるプログラミング手法』を参照してください。

推奨

トランザクション間の類縁性を扱う最良の方法は、まず第一にトランザクション間に類縁性を生じさせないことです。

類縁性を防止することが不可能な場合は、

- 適切な命名規則を使用してトランザクション間の類縁性を容易に識別できるようにし、
- 類縁性の存続期間を可能な限り短くすることです。

アプリケーション・プログラムの内容を変更すれば、トランザクション間の類縁性を防止することは可能ですが、類縁性に対応できるように、動的および分散ルーティング・プログラムに論理を組み込めば、その作業は必ずしも必要ではありません。また、類縁性に影響されるトランザクションを静的にルーティングすることもできます。

類縁性を防止するプログラミング手法

トランザクション間でデータを受け渡しするいくつかの手法は、トランザクション間の類縁性を生じさせない点で一般に安全です。これらの手法は、連絡域 (COMMAREA)、端末管理テーブルのユーザー域 (TCTUA)、または BTS コンテナーのいずれかを使用します。

類縁性が生じないようにするには、アドレスを含む COMMAREA、TCTUA、および BTS コンテナーを使用しないでください。一般に、そのようなアドレスが参照するストレージは長寿命ストレージにする必要があるので、動的トランザクション・ルーティングの環境においてこうしたアドレスを使用するようなプログラミング手法は安全ではありません。

COMMAREA

RETURN コマンドの COMMAREA オプションは、CICS 疑似会話型トランザクションの連続したトランザクション間で、データを受け渡しするために使用できる安全なプログラミング手法の主要な例です。ストレージに対して GETMAIN および FREEMAIN 要求を出すのは CICS で、アプリケーション・プログラムではありませんが、CICS は COMMAREA をユーザー・ストレージの特別な形式として扱います。

CICS は、RETURN コマンドに指定された COMMAREA の内容を、次のトランザクションの最初のプログラムで常時利用できるようにします。送信および受信トランザクションが別のターゲット領域で実行される場合にも、CICS は同じように機能します。動的ルーティング・プログラムが次のトランザクションをルーティングする宛先として、どのターゲット領域を選択しても、CICS は疑似会話型では、直前の RETURN コマンドに指定されている COMMAREA を、ターゲット領域において利用できるようにします。これを 図 88 で示します。

COMMAREA の一般的な特性は次のとおりです。

- プロセッサー・オーバーヘッドが低い。
- リカバリー可能ではない。
- RETURN コマンドの COMMAREA の長さは、トランザクションごとに異なり、理論上の上限は 32 763 バイトです。 (ただし安全のため、「アプリケーション・プログラミング・リファレンス」で推奨されているように、24KB (1KB = 1024 バイト) を超えないようにしてください。これは、さまざまな要因により理論上の上限値より限界が低くなることがあるためです。)
- CICS は、端末ユーザーが次のトランザクションに応答するまで CICS 主記憶装置の COMMAREA を保留します。大容量の COMMAREA を使用する場合、CICS が保留する COMMAREA の数は、ある領域におけるある時点でのタスクの最大数ではなく端末の使用率に影響するので、このことは考慮すべき重要事項です。
- 次のトランザクションの最初のプログラムがデータを別のプログラムか、その次のトランザクションに明示的に渡さない限り、COMMAREA は最初のプログラムのみが利用できます。

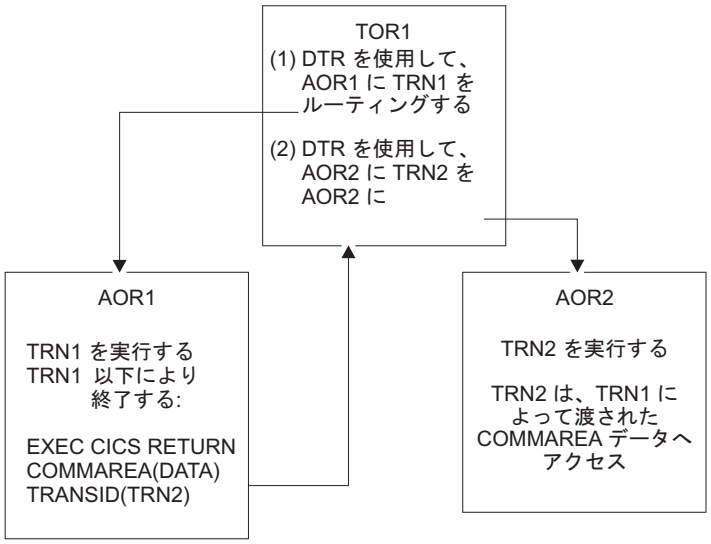


図 88. 動的トランザクション・ルーティング環境における疑似会話型の COMMAREA の使用

疑似会話型トランザクションで使用される COMMAREA は、図 88 に示されているように、CICSplex 上のトランザクション間で受け渡され、COMMAREA にストレージ域のアドレスではなくデータのみが収容されている場合、トランザクション間の類縁性が生じることはありません。

TCTUA

TCTUA は、端末管理テーブル項目 (TCTTE) のオプションによる拡張であり、各項目は拡張が存在するかどうか、またその長さを指定します。端末に関連した TCTUA が必要で、TYPETERM リソース定義の USERAREALEN パラメーターでその長さを定義したとします。すると、同じ TYPETERM 定義を使用して作成されたすべての端末の TCTUA の長さは固定長になります。

疑似会話型トランザクションの連続したトランザクション間でのデータの受け渡しの方法として、端末管理テーブル・ユーザー域 (TCTUA) は動的トランザクション・ルーティング環境において安全に使用できます。COMMAREA と同様に、TCTUA は、疑似会話型のトランザクションが別のターゲット領域にルーティングされても、ユーザー端末で開始されるトランザクションに常にアクセス可能です。これを 346 ページの図 89 で示します。TCTUA の他の一般的な特性は次のとおりです。

- プロセッサー・オーバーヘッドがきわめて小さい (アドレスの獲得のために必要な CICS コマンドは 1 つだけ)。
- リカバリー可能ではない。
- 与えられた TYPETERM 定義に関連した端末グループに対する TCTUA の長さは固定されている。少量のデータのみに適しており、可能な最大サイズは 255 バイト。
- 端末が自動インストールされると、TCTUA は TCTTE が存在する限り存在し、TCTUA の保存期間は AILDELAY システム初期設定パラメーターによって決定されます。したがって、TCTTE および関連した TCTUA は、CICS と端末との間のセッションが終了後、AILDELAY 期間が満了したときに削除されます。

  明示的端末定義により CICS に端末を定義した場合、TCTTE およびそれに関連した TCTUA は端末のインストール時に作成され、CICS の次の初期スタートまたはコールド・スタートまで保有されます。

TCTUA は、ターゲット領域のアプリケーション・プログラムと同様に、ルーティング領域の動的ルーティング環境でも使用可能であることに注意してください。TCTUA は、トランザクションの動的ルーティングに関連した情報を保管するために、使用することができます。例えば、TCTUA を使用して、トランザクションがルーティングされる宛先として選択したターゲット領域名を保管することができます。

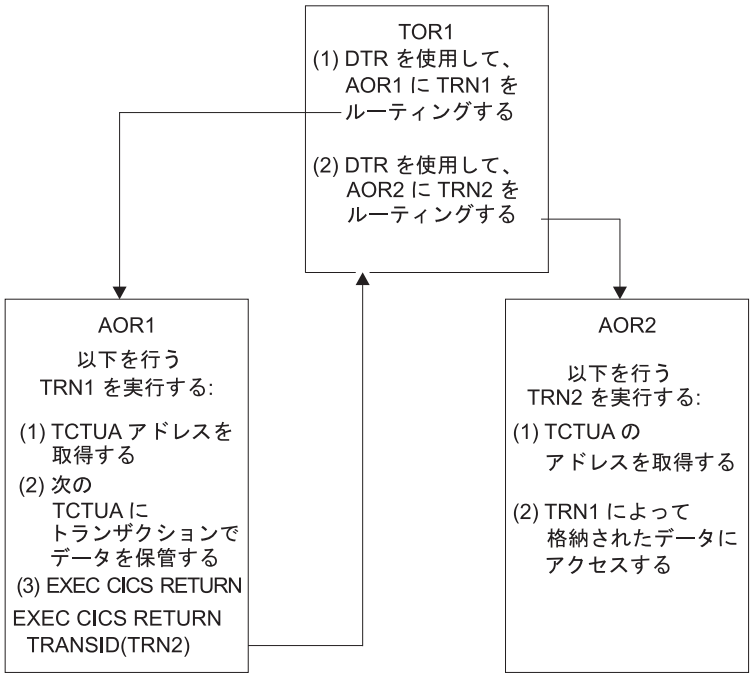


図 89. 動的ルーティング環境における疑似会話型の TCTUA の使用

安全でない方法での TCTUA の使用

EXEC CICS ADDRESS TCTUA(*ptr-ref*) を使用すると、TCTUA に対して直接的にアドレッシング可能になりますが、これは、TCTUA にアクセスする必要がある各タスクが TCTUA アドレスを獲得する方法です。タスクが一時記憶キューなどの他の方法を使用して TCTUA のアドレスを受け渡すか、または TCTUA 自身を使用して他のストレージ域のアドレスを受け渡すと、TCTUA では動的トランザクション・ルーティング環境における安全なプログラミング手法を提供できません。

別のタスクに関連した端末を指定する INQUIRE TERMINAL コマンド (INQUIRE TERMINAL コマンドは指定端末の TCTUA アドレスを返します) を発行することにより、タスクがそれ自体の TCTUA ではなくプリンシパル装置の TCTUA を獲得する可能性もあります。TCTUA 機能のもう 1 つの危険な使用方法の例は、タスク自体のプリンシパル装置以外の端末の TCTUA アドレスを使用することです。状況によりますが、特に動的ルーティング環境では、照会タスクのプリンシパル装置でない端末の TCTUA は、アドレスが獲得された後で削除される可能性があります。例えばターゲット領域では、INQUIRE TERMINAL コマンドは、動的にルーティングされるトランザクションを実行している代理端末に関連した TCTUA アドレスを返すことができます。端末からの次のトランザクションが別のターゲット領域にルーティングされると、TCTUA アドレスはもはや有効でなくなります。

ENQMODEL リソース定義による ENQ および DEQ コマンドの使用

ENQ および DEQ コマンドを使用して、共用リソースへのアクセスを逐次化します。CICS の以前のリリースでは、これらのコマンドは、同一の領域で実行される CICS タスクの有効範囲に限定されていたため、異なる領域に存在するタスクが共有

するリソースへのアクセスの逐次化には使用できませんでした。現在では、ENQ および DEQ が該当する ENQMODEL リソース定義 (ENQMODEL の説明については「*CICS Resource Definition Guide*」のENQMODEL リソース定義を参照) によりサポートされるという**条件で**、シスプレックス全体をその有効範囲にすることができます。

これは主に、トランザクション・ルーティングを決定するシステム・プログラマーにとって重要ですが、アプリケーション・プログラマーも、現在使用可能なこの利点を認識しておくべきです。

シスプレックスのエンキューおよびデキューの概要

CICS のエンキュー / デキュー機能を変更することで、CICS アプリケーション・プログラミング・インターフェースは拡張され、シスプレックス内で動作する指定された CICS 領域の集合での、**名前付きリソース**へのアクセスを逐次化するエンキュー機構が使用できるようになります。これは、単一 MVS イメージ内の CICSplex にも、複数の MVS に常駐する CICSplex にも、同じように適用されます。(シスプレックス全体にわたるエンキューは**リソース**に対してのみサポートされ、**アドレス**上のエンキューにはサポートされていないことに注意してください。)

単一 CICS 領域内のローカル・エンキューは、CICS アドレス・スペース内で管理されます。複数の CICS 領域に影響を及ぼす、シスプレックス全体にわたるエンキューは、グローバル・リソース・サービス (GRS) によって管理されます。CICS のエンキューおよびデキュー機構に対する主な変更点は、以下のとおりです。

- シスプレックスのエンキューおよびデキューは、新しい CICS リソース定義タイプ ENQMODEL の導入によって EXEC CICS ENQ|DEQ コマンドの有効範囲を領域からシスプレックスへ拡張し、シスプレックス全体で有効になるようにリソース名を定義します。
- ENQMODEL リソース定義の属性 ENQSCOPE は、同一のエンキュー有効範囲を共有する領域の集合を定義します。
- インストールされた ENQMODEL リソース定義の名前と一致する名前を持つリソースに対し、EXEC CICS ENQ (または DEQ) コマンドが発行された場合、CICS は以下のようにして、ENQSCOPE 属性の値を検査し、有効範囲がローカルか、シスプレックス全体かを判別します。
  - ENQSCOPE 属性が左方ブランク (デフォルト値) である場合は、CICS は ENQ|DEQ を、発行している CICS 領域でのローカルとして扱います。
  - ENQSCOPE が非ブランクの場合には、CICS は ENQ|DEQ をシスプレックス全体にわたるものとして扱い、GRS にキュー名とリソース名を渡して、エンキューを管理します。リソース名は EXEC CICS ENQ|DEQ コマンド上で指定され、キュー名は、ENQSCOPE に 4 文字の DFHE でプレフィックス変換することによって作成されます。
- シスプレックス全体にわたるエンキューおよびデキュー機能を使用する必要のある CICS 領域は、必ず、必要な ENQMODEL を定義し、インストールしなければなりません。

  これを確実に行うために、CICS 領域が CSD を共用し、初期設定グループ・リストが同一の ENQMODEL グループを組み込む方法をお勧めします。

既存のアプリケーションは、適切な ENQMODEL を定義して、アプリケーション・プログラムに変更を加えなくても、シスプレックス・エンキューが使用できます。

利点

シスプレックス・エンキューには、以下のような利点があります。

- トランザクション間の類縁性の、最も一般的な原因の 1 つを除去する
- コスト・パフォーマンス、処理能力、可用性を向上させ、並列シスプレックスの活用性を高める
- 動的および分散ルーティング・プログラムにおけるトランザクション間の類縁性規則の必要性を削減し、それによって、並列シスプレックス活用のシステム管理コストを引き下げる
- シスプレックス上の複数の CICS タスクにより実行される共有一時記憶キューに対する同時更新の逐次化を可能にする
- 別々の CICS 領域で同時に発生したタスクがリモート一時データ・キューに書き込むレコードのインターリーブを防ぐことができる
- シスプレックス上のタスクのシングル・スレッド化および同期化を可能にする。この機能は、リカバリー可能リソースのロックを目的としているわけではありません。

BTS コンテナー

コンテナーを所有しているのは、BTS アクティビティーです。コンテナーはアクティビティーの外では使用できません。詳しくは、「*CICS Business Transaction Services*」を参照してください。コンテナーを使用して、BTS アクティビティー間で、あるいは同じアクティビティーの異なる起動間で、データを渡すことができます。アクティビティーは GET および PUT コンテナーを使用して、コンテナーの内容を更新します。CICS は、BTS アクティビティーに関連した情報 (コンテナーも含む) をすべて、RLS で使用可能な VSAM ファイルに保管することによって、アクティビティーに対し、適切なコンテナーが使用可能であることを保証します。このような理由から、BTS 環境はシスプレックスの外側では拡張できませんが (「*CICS Business Transaction Services*」を参照)、コンテナーでデータを渡すシスプレックス内部の動的ルーティングは使用できます。

コンテナーの一般的特性の一部を、次に挙げておきます。

- アクティビティーはいくつでもコンテナーを所有できる。ユーザーに制限はありません。
- サイズに制限がない。
- リカバリー可能である。
- 関連したアクティビティーが実行中である場合にのみ、主記憶装置に存在する。それ以外のときは、ディスク上に保存されています。したがって、端末 COMMAREA の場合と違い、ストレージ要件を過度に心配する必要はありません。

類縁性を生じさせるプログラミング手法

CICS アプリケーション・プログラミング手法の一部 (特に共用ストレージにアドレスを渡したり獲得する手法) は、トランザクション間に類縁性を生じさせます。

一般に危険なプログラミング手法を次のセクションで説明します。

- 『共通作業域の使用』
- 350 ページの『GETMAIN SHARED ストレージの使用』
- 351 ページの『LOAD PROGRAM HOLD コマンドの使用』
- 353 ページの『タスク存続期間ストレージの共用』
- 354 ページの『WAIT EVENT コマンドの使用』
- 355 ページの『ENQMODEL リソース定義のない ENQ および DEQ コマンドの使用』

共通作業域の使用

CICS 領域の CWA は、CICS の初期設定の際に作成され (オプション)、CICS が終了するまで存在し、CICS のリスタート (ウォームまたは緊急) ではリカバリーされません。ADDRESS CWA(*ptr-ref*) コマンドは、CWA に対する直接的なアドレッシングを可能にします。

CWA のような長寿命共用ストレージの使用が類縁性を生じさせることを示すいい例は、1 つのタスクが CWA にデータを記憶し、その後のタスクがそのストレージからデータを読み取る場合です。明らかに、データを検索するタスクは、データを保管したタスクと同じターゲット領域で実行されなければなりません。そうでないと、データを検索するタスクは、別のアドレス・スペースのまったく別のストレージ域を参照することになります。このことにより、350 ページの図 90 に示すように、動的または分散ルーティング・プログラムのワークロード平衡化機能が制限されます。

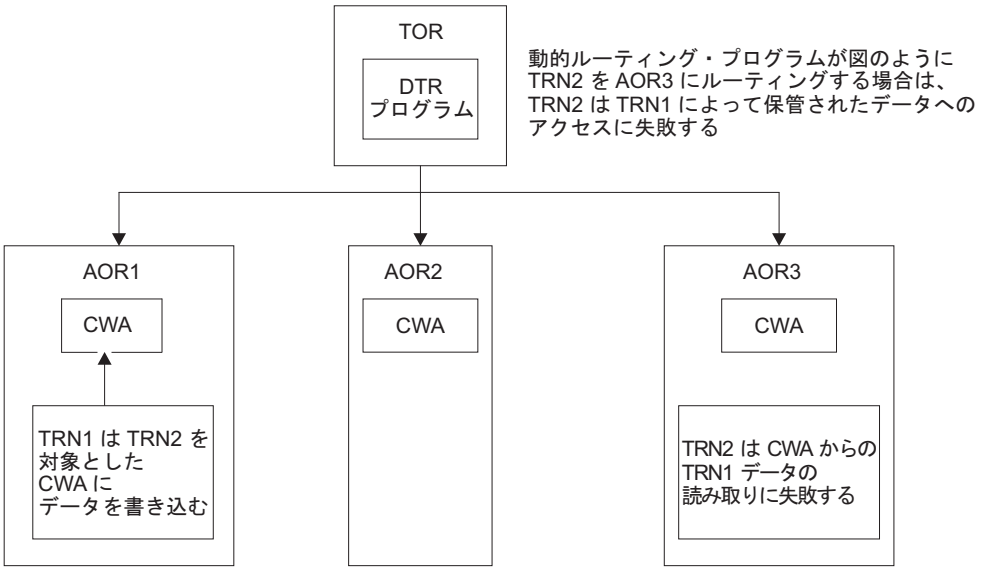


図 90. CWA の使用により生じるトランザクション間の類縁性を示す図: 動的ルーティング・プログラムは、この CWA 類縁性を認識し、TRN2 を TRN1 と同じターゲット領域にルーティングしなければなりません。

しかし、CWA が読み取り専用データを含み、このデータの複製が複数のターゲット領域に存在する場合、CWA を使用し、かつ動的ルーティングの機能を十分に利用することができます。例えば、読み取り専用データが入った CWA を、選択したいくつかのターゲット領域のそれぞれにロードする、CICS 始動 (PLTPI プログラム) の初期設定後の段階でプログラムを実行することができます。これにより、同一の CWA データがロードされたターゲット領域にルーティングされるすべてのトランザクションは、どのターゲット領域にルーティングされても、同一のデータにアクセスできます。CICS サブシステム・ストレージ保護機構により、CICS キー・ストレージから CWA を要求し、CWA を読み取るすべてのプログラムがユーザー・キーで実行されるように定義すると、CWA データの読み取り専用の整合性を保つことができます。

GETMAIN SHARED ストレージの使用

共用ストレージは GETMAIN SHARED コマンドにより割り振られ、同一または別のタスクにより明示的に解放されるまで割り振られたままです。共用ストレージは、共用ストレージの存続期間中に実行されるすべての CICS タスク間でデータを交換する場合に使用できます。この方法で設計されるトランザクションが正常に機能するためには、トランザクションは同一の CICS 領域で実行されなければなりません。動的または分散ルーティング・プログラムは、共用ストレージを使用するトランザクションを、同一のターゲット領域にルーティングしなければなりません。

351 ページの図 91 は、共用ストレージの使用を示しています。

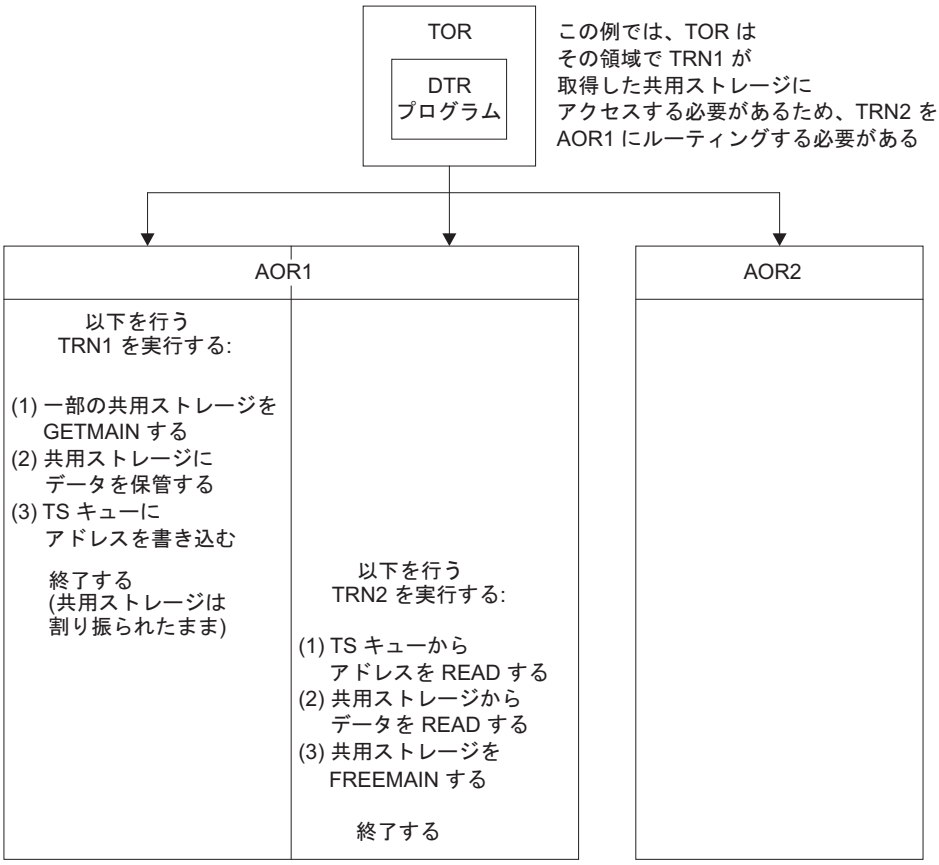


図 91. 共用ストレージの使用により生じるトランザクション間の類縁性を示す図: 動的トランザクション・ルーティング・プログラムは、この類縁性を認識し、TRN2 を TRN1 と同じターゲット領域に確実にルーティングする必要があります。

図 91 に示す 2 つのトランザクションが、疑似会話型トランザクションの一部である場合、COMMAREA によって、共用ストレージの使用を置き換えなければなりません (ストレージの範囲が、COMMAREA のサイズ限度を超えない場合)。

LOAD PROGRAM HOLD コマンドの使用

CICS が LOAD PROGRAM HOLD コマンドに対する応答としてロードするプログラム (またはテーブル) は、同一のタスクまたは別のタスクにより明示的に解放されない限り、直接アドレッシング可能なストレージに存続します。ロード済みプログラム (テーブル) がストレージに保持されている間に実行される CICS タスクはいずれも、次の条件に合えば、データ交換のためにロード済みプログラムのストレージを使用することができます。

- プログラムが読み取り専用ストレージにロードされていない場合、または、
- プログラムが RELOAD(YES) を指定して CICS に定義されていない場合。

一時記憶キューを使用して、ロード済みプログラムのストレージのアドレスを他のタスクでも利用可能にすることができますが、通常使用される方法は、他のタスクが SET(*ptr_ref*) オプションを指定した LOAD PROGRAM コマンドを発行し、CICS が保留プログラムのアドレスを返すことができるようにする方法です。

LOAD PROGRAM HOLD コマンドの使用により生じる類縁性の特性は、実質的には GETMAIN SHARED ストレージの使用により生じる類縁性の特性と同じで (351 ページの図 91、および図 92を参照してください)、同じ規則が当てはまります。アプリケーション設計を保護するためには、動的または分散ルーティング・プログラムにより、ロード済みプログラム (またはテーブル) のアドレスを使用するすべてのトランザクションが、同一のターゲット領域に確実にルーティングされなければなりません。

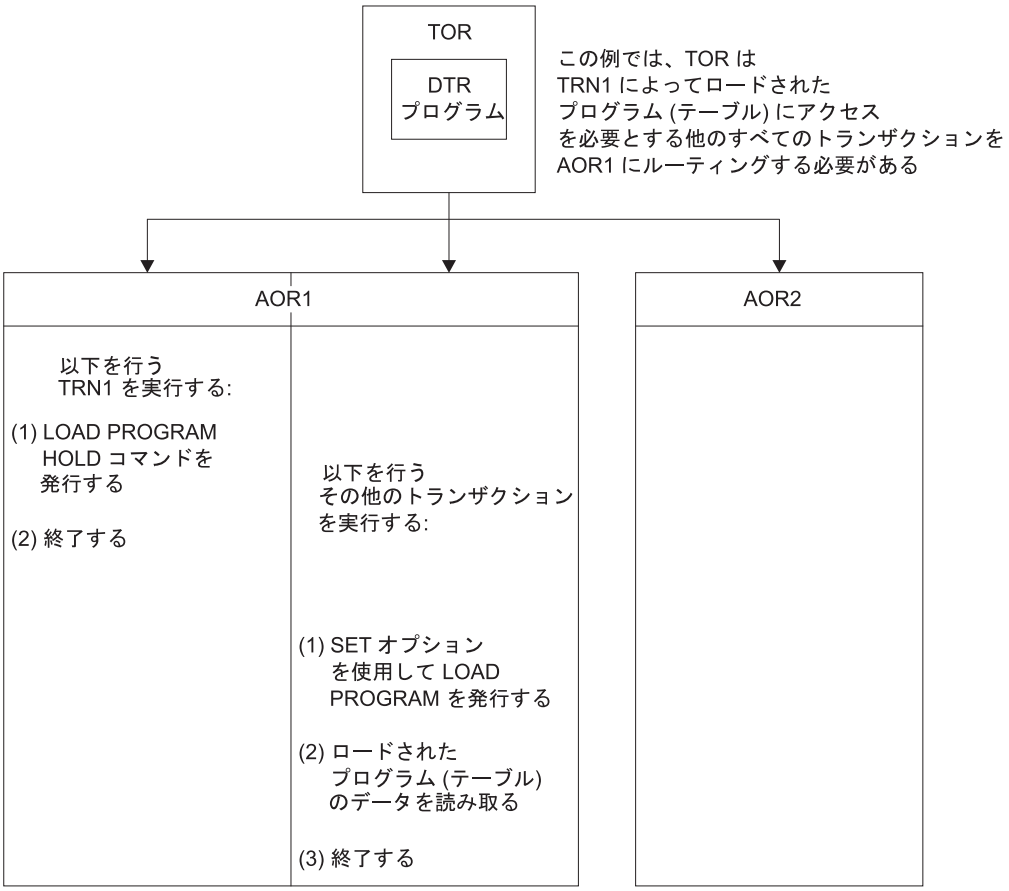


図 92. LOAD PROGRAM HOLD の使用により生じるトランザクション間の類縁性を示す図： 動的ルーティング・プログラムは、この類縁性を認識し、TRN2 を TRN1 と同じターゲット領域に確実にルーティングしなければなりません。

注: この規則は、ロード済みプログラム (HOLD オプションが LOAD コマンドに指定されているかどうかにかかわらず) 用のリソース定義の RESIDENT オプションにより定義されているプログラムにも当てはまります。ただし、類縁性に関する考慮事項を無視して、トランザクション間でデータを共用できるように RESIDENT オプションを使用することは危険です。これは、RESIDENT を使用して定義されているプログラムは SET PROGRAM(*program_name*) NEWCOPY コマンドによって決まり、変更される可能性があるためです。

上記の規則はまた、連絡し合うタスクが同期化される非常駐、非保留ロード済みプログラムにも当てはまります。

タスク存続期間ストレージの共用

1 つのタスクが所有するタスク存続期間ストレージの使用を、別のタスクと共有することができます。ただし、それは所有側タスクが同じ CICS アドレス・スペースに存在する別のタスクにアドレスを渡すことができる場合に限られます。この手法では、連絡し合うタスク間に類縁性を生じさせ、渡されるアドレスを検索および使用するタスクは、タスク存続期間ストレージの所有側タスクと同じターゲット領域で実行されなければなりません。

例えば、一時記憶キューを使用して PL/I 自動変数のアドレス、または COBOL 作業用ストレージ構造のアドレスを渡すことができます (図 93 の例を参照してください)。

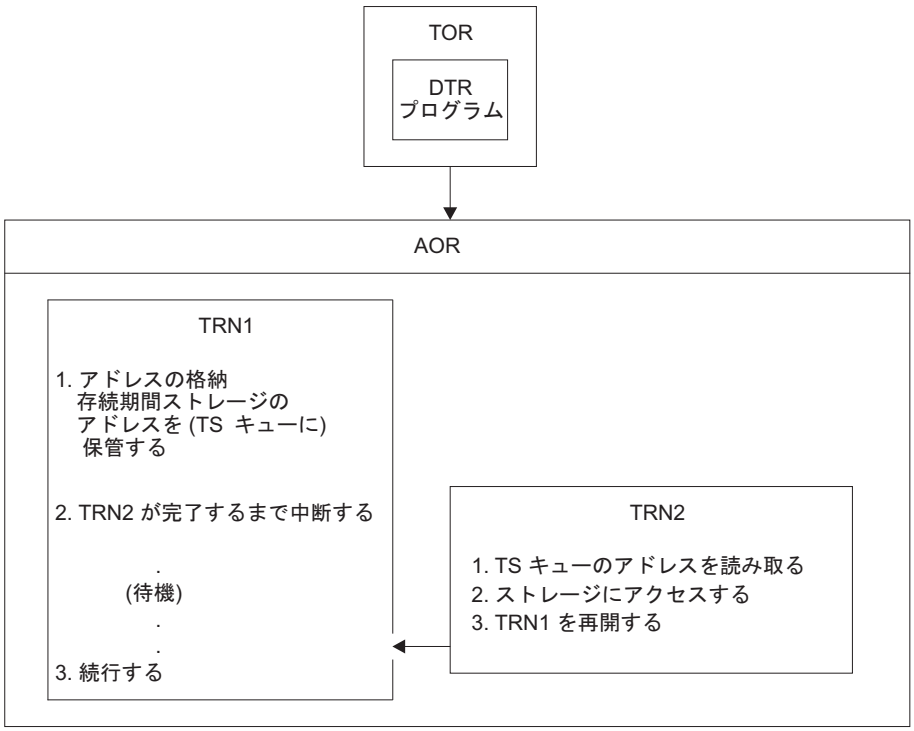


図 93. タスク存続期間ストレージの使用により生じるトランザクション間の類縁性を示す図: TRN2 は、TRN1 と同じターゲット領域で実行されなければなりません。また、TRN2 がタスク存続期間ストレージの使用を完了するまで TRN1 は終了してはなりません。

2 つのタスクが、そのうちの一方のタスクが所有するタスク存続期間ストレージを共用するためには、これらのタスクはなんらかの方法で同期化されなければなりません。ローカル・ストレージのアドレスを渡すタスクを中断および再開するためのコマンドについては、表 20を参照してください。

表 20. タスクの中断および再開 (同期化) 方法

中断命令	再開命令
WAIT EVENT、WAIT EXTERNAL、WAITCICS	POST
RETRIEVE WAIT	START
DELAY	CANCEL
POST	CANCEL

表 20. タスクの中断および再開 (同期化) 方法 (続き)

中断命令	再開命令
START	CANCEL
ENQ	DEQ

これらの手法のいくつかでは、それを使用するトランザクションが同一のターゲット領域で実行されることが必要です。これについては、後述されています。しかしながら、異なるターゲット領域で実行されるタスクを同期化できる場合でも、タスク存続期間ストレージのアドレスをタスク間で受け渡すことは危険です。動的ルーティングを使用しない場合でも、 353 ページの表 20 に示す同期手法に基づいた設計は、ストレージ所有側タスクがパージされる可能性があるため、基本的に危険です。

注:

1. RETRIEVE WAIT/START のような同期手法を使用してタスク存続期間ストレージを共用可能にすることは、お勧めできません。
2. 他のタスクのタスク存続期間ストレージを共用するタスクが、START コマンドにより開始される場合、その START コマンドが、リモート・システムへ機能伝送またはルーティングされるものでない限り、トランザクション間の類縁性は生じません。

WAIT EVENT コマンドの使用

WAIT EVENT コマンドは、1 つのタスクを、別の CICS または MVS タスクにより実行されるイベントの完了に同期化させるために使用します。

イベントの完了は、イベント制御ブロック (ECB) にビット・パターンを設定することにより通知されます。待ちタスクおよび通知タスクの両方が ECB に対して直接アドレッシング可能でなければならず、したがって、両方のタスクが同一のターゲット領域で実行されなければなりません。一時記憶キューを使用する方法は、待ちタスクが ECB のアドレスを別のタスクに渡すことができる 1 つの方法です。

355 ページの図 94 はこの同期手法を説明しています。

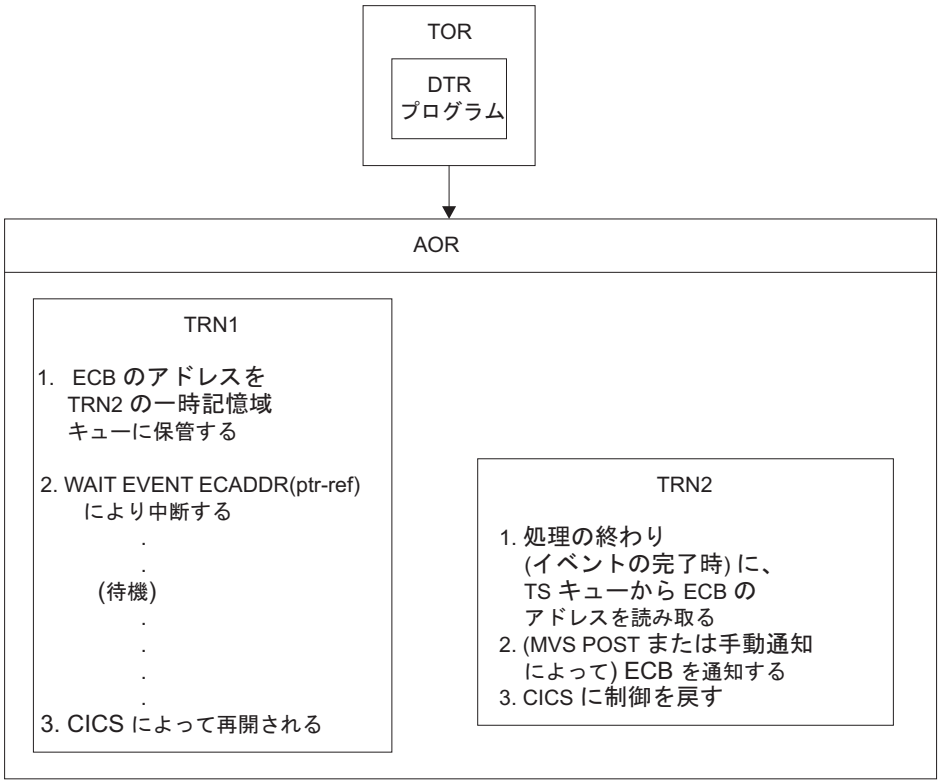


図 94. WAIT EXTERNAL コマンドの使用により生じるトランザクション間の類縁性を示す図： TRN2 は、TRN1 と同じターゲット領域で実行されなければなりません。

図 94 に示されている TRN2 が、TRN1 とは別のターゲット領域で実行された場合、*ptr-ref* の値が無効になって通知操作に予期せぬ結果が生じ、待ちタスクは再開されません。したがって、動的または分散ルーティング・プログラムは、アプリケーション設計を保護するために、通知タスクが待ちタスクと同じターゲット領域で確実に実行されるようにしなければなりません。タスクの同期化に WAIT EXTERNAL コマンドおよび WAITCICS コマンドを使用する場合は、同様の事項を考慮してください。

ENQMODEL リソース定義のない ENQ および DEQ コマンドの使用

ENQ および DEQ コマンドを使用して、共用リソースへのアクセスを逐次化します。これらのコマンドは、同一の領域で実行する CICS タスクに関してのみ有効であり、別の領域に存在するタスクが共有するリソースへのアクセスの逐次化には使用できません。ただし、それらがシスプレックス全体を有効範囲とするように、適切な ENQMODEL リソース定義によってサポートされている場合は**例外**です。ENQMODEL の説明については、 346 ページの『ENQMODEL リソース定義による ENQ および DEQ コマンドの使用』および「*CICS Resource Definition Guide*」の ENQMODEL リソース定義を参照してください。

LENGTH オプションを指定しない ENQ は**アドレス**上でエンキューとして処理されるため、その有効範囲はローカルのみとなるので注意してください。(有効範囲をシスプレックス全体に広げる ENQMODEL 定義を使用せずに) 逐次化する ENQ およ

び DEQ の使用については、図 95 に説明されています。

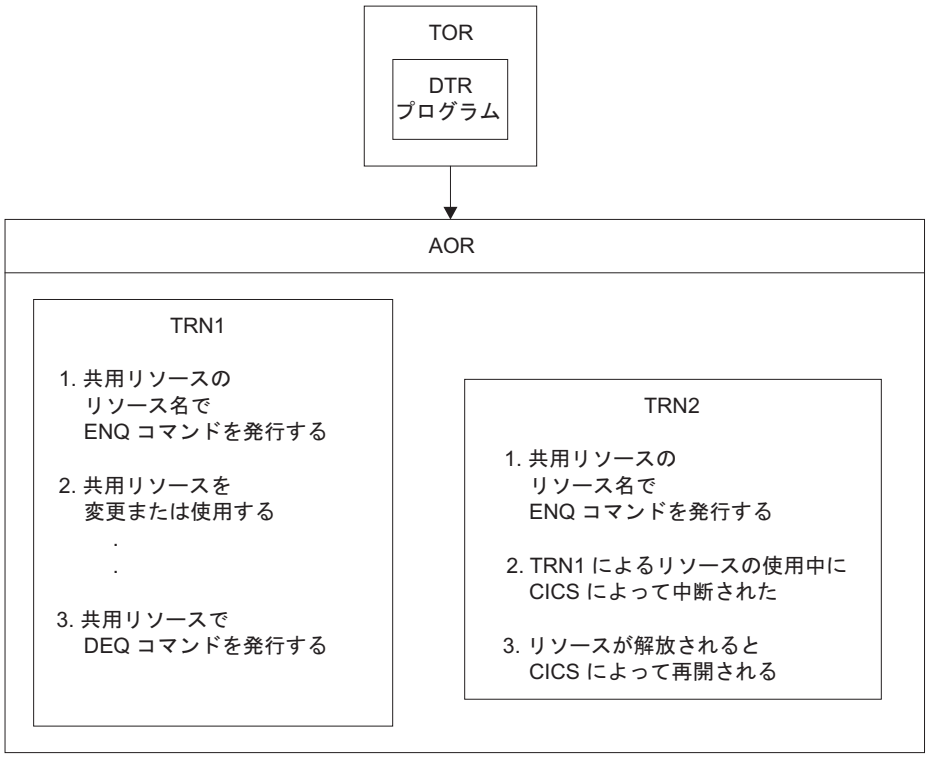


図 95. ENQ/DEQ コマンドの使用により生じるトランザクション間の類縁性を示す図： TRN2 は、TRN1 と同じターゲット領域で実行されなければなりません。

図 95 に示す TRN2 が、TRN1 とは別のターゲット領域で実行された場合、TRN1 が共用リソースにアクセスしても TRN2 は中断されません。したがって、動的または分散ルーティング・プログラムは、アプリケーション設計を保護するために、与えられたリソース名に対してエンキューするすべてのタスクが、同一のターゲット領域で確実に実行されるようにしなければなりません。TRN2 は、もちろん、ターゲット領域において、同一のリソース名に ENQ コマンドを発行する他の CICS タスクと共に逐次化されます。

類縁性を生じさせる可能性のあるプログラミング手法

CICS アプリケーション・プログラミング手法の一部は、組み込まれ方によってトランザクション間に類縁性を生じる場合があります。

危険性のあるプログラミング手法を次のセクションで説明します。

- 357 ページの『一時記憶域の使用』
- 360 ページの『一時データの使用』
- 361 ページの『RETRIEVE WAIT および START コマンドの使用』
- 363 ページの『START および CANCEL REQID コマンドの使用』
- 365 ページの『DELAY および CANCEL REQID コマンドの使用』
- 367 ページの『POST および CANCEL REQID コマンドの使用』

一時記憶域の使用

CICS アプリケーション・プログラムは一般的に、一時記憶域 (TS) キューを使用して一時的なアプリケーション・データを保持したり、スクラッチ・パッドとして使ったりします。

TS キューは、時には、トランザクションの 1 つのインスタンスで実行される、アプリケーション・プログラム間 (例えば、複数プログラム・トランザクションで、LINK または XCTL コマンドを介して制御を渡すプログラム間) のデータの受け渡しに使用されます。TS キューのこのような使用方法では、キューはトランザクション・インスタンスの存続期間だけ存在するので、異なるターゲット領域間で共有する必要はありません。なぜなら、トランザクション・インスタンスは 1 つのターゲット領域だけで実行されるためです。

注: プログラムの 1 つが分散プログラム・リンク・コマンドによりリンクされ、そのリンクされているプログラムがリモート・システム上にある複数プログラム・トランザクションでは、上記の内容は厳密には当てはまりません。このケースでは、DPL コマンドによってリンクしているプログラムは、リモート領域にある別の CICS タスクで実行されます。 DPL プログラムへデータを渡すお勧めの方法は、チャネルまたは COMMAREA を介する方法ですが、DPL アプリケーションにおいて、TS キューをデータの受け渡しに使用する場合、そのキューは 2 つの領域で共用されなければなりません。

時には、TS キューはターゲット領域に固有の情報を保持したり、読み取り専用データを保持したりします。この場合、各ターゲット領域に TS キューの複製を作成することができ、ターゲット領域間でのデータの共用は必要ありません。

しかし多くの場合、TS キューはトランザクション間のデータの受け渡しに使用されます。その場合、キューを使用するトランザクションが、動的に選択されたどのターゲット領域においても実行できるように、キューはグローバルにアクセス可能でなければなりません。TS キューをリモートとして定義できる場合には、TS 要求をキュー所有側領域 (QOR) に機能シップすることにより、一時記憶キューをグローバルにアクセス可能にすることができます。共用キューは、カップリング・ファシリティーの一時記憶域プールを使用して定義されます。共用一時記憶域は、リカバリー不能キューにのみ適用されます。補助記憶装置のキューはリカバリー可能にできますが、主記憶装置のキューはリカバリー可能にはできません。

疑似会話型トランザクションでは、プログラムを変更して会話のフェーズ間でのデータの受け渡しに COMMAREA を使用することができます。しかし、どのターゲット領域に対しても動的ルーティングを使用できるため、一時記憶域データ共用を使用すると、トランザクション間の類縁性は生じません。共用一時記憶キューは、特定の SYSID がリモート・キューの要求と同じ方法でルーティングされることを要求します。共用 TS プールに対して定義される SYSID 値は TST TYPE=SHARED です。

TS プールを指定する方法を使用すると、キューを QOR から TS データ共用プールへマイグレーションすることが容易になります。一時記憶域グローバル・ユーザー出口である XTSEREQ を使用して、TS 要求の SYSID を変更し、SYSID が TS データ共用プールを参照するようにできます。 358 ページの図 96 には、同一の TST (TST=XX) を使用する 4 つの AOR が示されています。TST TYPE=SHARED

マクロの SYSIDNT オプションは、DFHXQTS1 と呼ばれる共用 TS キューのプールに一時記憶域要求をマップします。TS データ共用サーバー DFHXQMN を始動するには、JCL のプール名パラメーターに DFHXQTS1 を定義する必要があります。データ共用サーバーの設定について詳しくは、「*CICS System Definition Guide*」を参照してください。

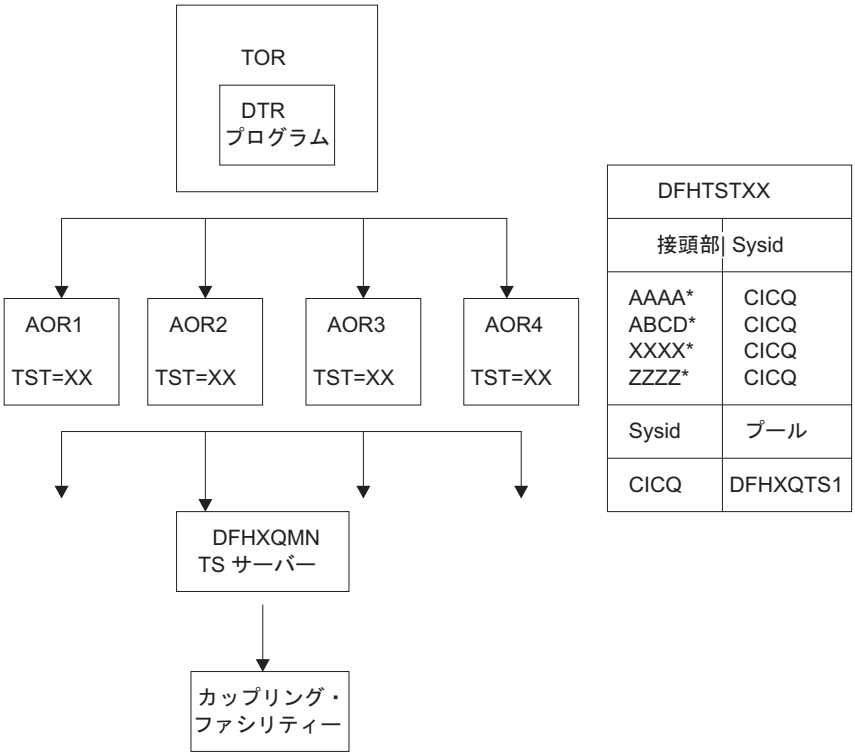


図 96. 一時記憶域データ共用サーバーの使用例

リモート・キューの命名規則

キューをリモートとして定義するためには、キューの項目を一時記憶域テーブル (TST) に入れるか、あるいは適切な TSMODEL を使用しなければなりません。 TS キュー名はしばしば動的に生成されますが、固有の固定名でも構いません。

- TST 命名規則では、CICS アプリケーション・プログラムが、変数の接尾部を追加することができる定数接頭部から成る総称名が、許可されていることによって、動的な名前が認められています。(総称名は、8 文字のキュー名の先行文字から構成され、最長 7 文字までです。8 文字すべてを使用する TST 項目の名前は、固有の TS キューを示します。)
- TSMODEL リソース定義によって定義される TS キューの名前は、Prefix または Remoteprefix オプションで定義する場合は、(指定された文字セットを使用して) 16 文字までの接頭部を、XPrefix または XRemoteprefix オプションで定義する場合は、(任意の 16 進数ストリングを使用して) 32 文字までの接頭部を持つことができます。「*CICS Resource Definition Guide*」のTSMODEL 定義属性には、Prefix、Remoteprefix、XPrefix および XRemoteprefix についての情報が記載されています。

通常規則では、4 文字の接頭部 (例えば、トランザクション ID) の後に接尾部として 4 文字の端末 ID が続きます。これにより、指定した端末に対して固有のキュー名を生成します。このような総称キュー名は、例えば以下のものによって所有されたリモート・キューとして、簡単に定義することができます。

- QOR (これによりトランザクション間の類縁性問題が回避される)
- 一時記憶域データ共用キュー・プールに常駐する共用キュー
- ターゲット領域が所有するリモート・キュー、または一時記憶域データ共用キュー・プールにあるリモート・キュー

リモート・キューおよび共用キューは、アプリケーション・プログラムに対して同じように定義することができますが、特定 SYSID への要求は、TST TYPE=SHARED により一時記憶域データ・サーバーにルーティングされます。しかし、動的に命名されたキューの命名規則がこの規則に従っていない場合は、キューをリモートとして定義することができず、キューにアクセスするすべてのトランザクションが、キューが作成されたターゲット領域にルーティングされなければなりません。さらに、TS 要求のグローバル・ユーザー出口を使用して、TS キュー名をローカル・キュー名からリモート・キュー名に変更することはできません。

リモート・キュー所有領域の使用については、図 97を参照してください。

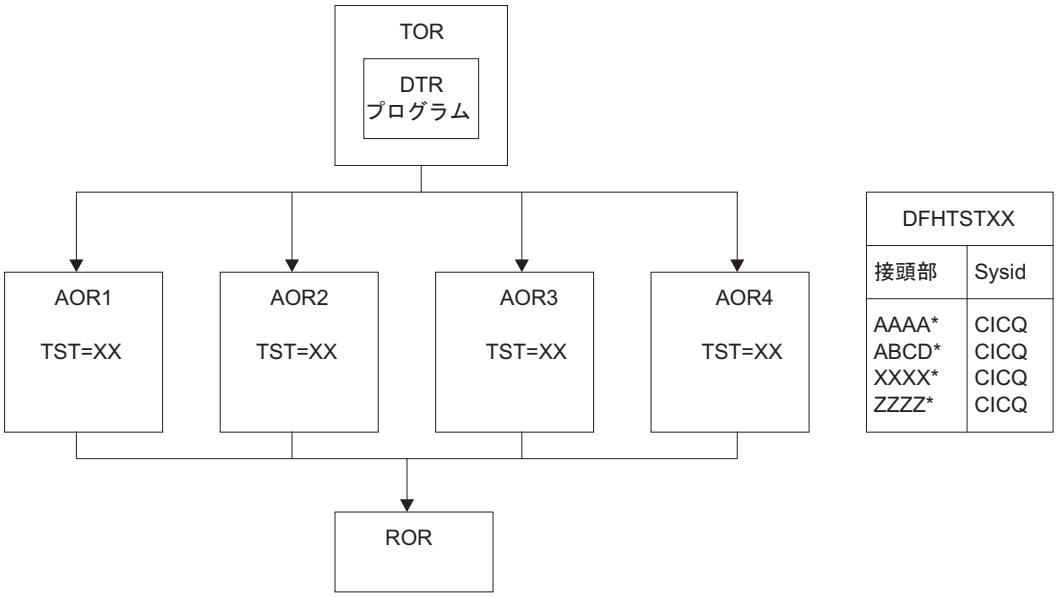


図 97. 一時記憶域に関連したトランザクション間の類縁性を回避するための、リモート・キューの使用： この例は、ファイル所有領域およびキュー所有領域を示しています。別々の領域を使用することができますが、ファイル制御および一時記憶域管理により個々に管理されているデータをリカバリーするときに「未確定」の窓が発生する可能性があるので、リカバリー操作時には特別な配慮が必要です。

グローバルにアクセス可能なキューの例外条件

グローバル QOR を使用して TS キューに関連するトランザクション間の類縁性を生じないようにする場合、例外条件の取り扱いにも注意してください。なぜなら、同一の領域においてトランザクションとキューがローカルである場合には起こらな

かった、いくつかの例外条件が起こる可能性があるためです。このような状態になるのは、ターゲット領域および QOR 個々に、障害が起きることがあるためです。その結果次の状況が生じます。

- ターゲット領域だけに障害が起き、QOR は継続されたため、キューが既に存在する。
- QOR だけに障害が起き、ターゲット領域は継続されたため、キューが検出されない。

一時データの使用

CICS アプリケーション・プログラムが通常使用するデータ・キューのもう一つの形式は、一時データ・キュー (TD) です。TD キューの動的トランザクション・ルーティングに関する考慮事項は、一時記憶域に関する考慮事項と多くの点で共通しています。共有する必要がある TD キューを使用するトランザクションを、ターゲット領域に動的にルーティングできるようにするには、TD キューをグローバルにアクセス可能にしなければなりません。

すべての一時データ・キューは、CICS に定義しなければならず、CICS 領域がリソースを利用できるようするには事前にインストールしておかなければなりません。リモート一時データ・キュー所有領域 (QOR) をサポートするように、これらの定義を変更することができます。

ただし、トリガー機能を使用する TD キューには制限があります。トリガー・レベルに達したときに呼び出されるトランザクションは、キューが常駐する領域 (QOR 内) のローカル・トランザクションとして定義されなければなりません。したがって、トリガー・トランザクションは QOR で実行されなければなりません。しかし、キューに関連した端末を QOR 内のローカル端末として定義する必要はありません。このため、トランザクション間の類縁性が生じることはありません。

361 ページの図 98 は、リモート一時データ・キュー所有領域の使用を示しています。

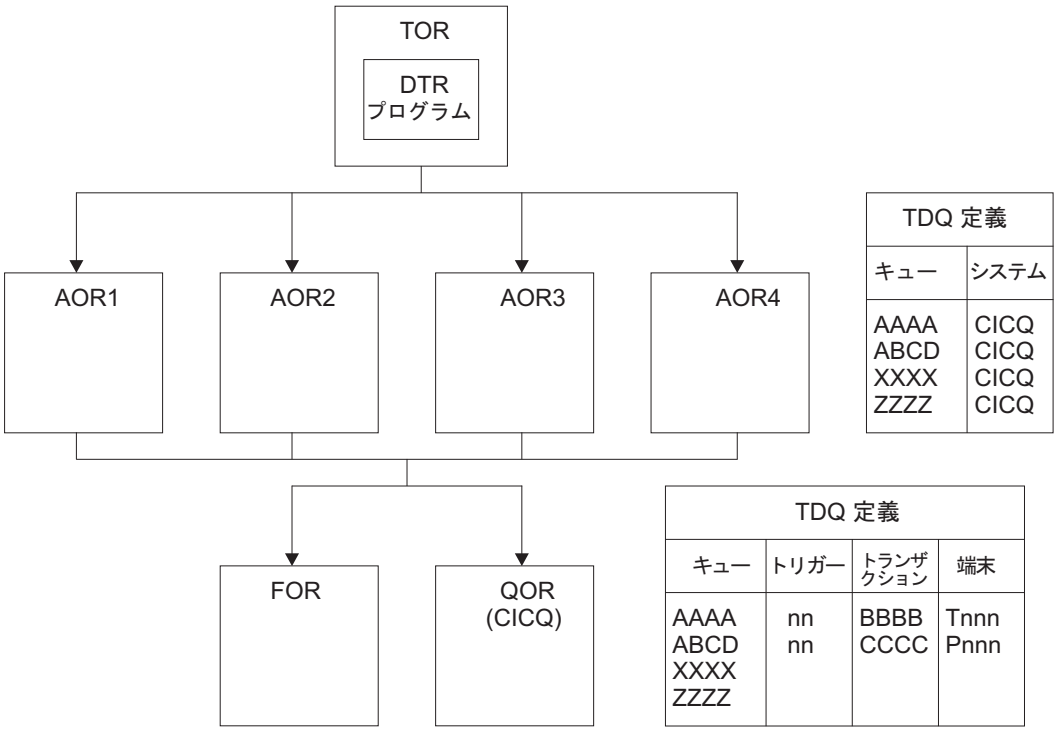


図 98. 一時データに関連したトランザクション間の類縁性を回避するための、リモート・キューの使用： ターゲット領域にインストールされた一時データ・キュー定義は、QOR (CICQ) に所有されたキューとして定義されています。QOR にインストールされた一時データ・キュー定義はすべてローカルで、トリガー・レベルを持つものもいくつかあります。

グローバルにアクセス可能なキューの例外条件

グローバル QOR を使用して TD キューに関連したトランザクション間の類縁性を生じないようにする場合、新しい例外条件があってはなりません (システム定義エラーまたは障害がある場合の SYSIDERR を除く)。

RETRIEVE WAIT および START コマンドの使用

同期手法のいくつかを使用して、2 つの同期化されたタスク間でタスク存続期間ストレージを共有することができます。例えば、362 ページの図 99 に示されているように、RETRIEVE WAIT および START コマンドはこの目的で使用できます。

この図の例では、TRN1 は非同期タスク TRN2 のデータを検索するように設計されており、その結果、TRN2 がデータを利用可能にするまで待機しなければなりません。この機構が正常に機能するためには、TRN1 は端末に関連したトランザクションでなければなりません。

ステップを次に示します。

1. TRN1 は、TS キューにデータ (TRANSID および TERMID を含む) を書き込みます。

2. TRN1 は自分自身を中断させるために RETRIEVE WAIT コマンドを発行し、それにより CICS は、RETRIEVE が解除されるまで、つまり TRN2 が FROM パラメーターによって渡されるデータとともに、START コマンドを発行するまでそのタスクを中断します。
3. しかしながら、中断されたタスクの TRANSID および TERMID (図の例では TRN1) を認識する場合のみ、TRN2 は START コマンドを発行して TRN1 を再開することができます。このようにして TRN2 は、TRN1 が書き込んだ情報を得るために TS キューを読み取ります。一つの方法として、一時記憶キューを使用して、中断されているタスクによりこの情報を受け渡すことができます。
4. TS キューからの情報を使用して、TRN2 は TRN1 に対して START コマンドを発行します。その結果、CICS は未解決の RETRIEVE WAIT を解除することにより TRN1 を再開します。

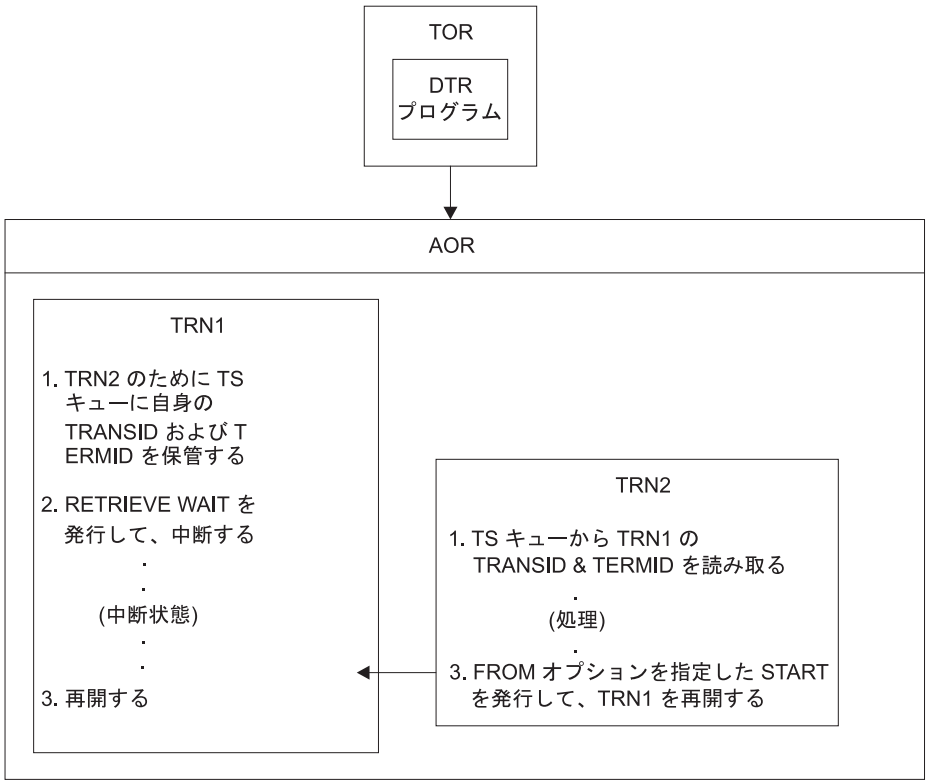


図 99. RETRIEVE WAIT および START コマンドを使用するタスク同期化の図

図 99 に示されている RETRIEVE WAIT および START コマンドを使用するタスク同期化の例では、RETRIEVE WAIT を解除する START コマンドは次のいずれかの条件を満たしている必要があります。

- RETRIEVE WAIT コマンドを発行するトランザクション (例では TRN1) と同じターゲット領域で発行される、または
- RETRIEVE WAIT コマンドが実行されたターゲット領域の SYSID を指定する、または
- RETRIEVE WAIT コマンドを実行したターゲット領域に常駐するリモート・トランザクションとして定義されている、TRANSID (例では TRN1) を指定する

リモート TRANSID 手法に基づくアプリケーション設計は、2 つのターゲット領域に対してのみ作用します。START コマンドの SYSID オプションを使用するアプリケーション設計は、すべてのターゲット領域が他のターゲット領域すべてに対する接続をもっている場合の、複数のターゲット領域に対してのみ機能します (好ましくないアプリケーション設計です)。いずれの場合でも、アプリケーション・プログラムの変更が必要です。動的または分散ルーティング・プログラムでは、ルーティング・プログラムに制限を加えない限り、このプログラミング手法を正しく使用することはできません。一般にこのことは、アプリケーション設計の保護のために、動的または分散ルーティング・プログラムが、TRN2 を TRN1 と同一の領域で実行しなければならないことを意味しています。

START および CANCEL REQID コマンドの使用

この CICS アプリケーション・プログラミング手法を使用して、あるトランザクションが指定した間隔の後に、別のトランザクションを開始する START コマンドを発行します。次に、別のトランザクション (START コマンドが要求するトランザクションではないもの) が、要求されたトランザクション (REQID パラメーターにより識別されるもの) はもはや実行する必要がないと判断し、START 要求を取り消します。この取り消しは、指定されている間隔が終了していない場合にのみ有効であることに気を付けてください。

一時記憶キューの使用は、タスク間で REQID を受け渡す 1 つの方法です。

注: この手法を使用するためには、CANCEL コマンドには REQID オプションを指定しなければなりませんが、START コマンドには必要ありません。これは、START コマンドで NOCHECK オプションが指定されていない場合、CICS が要求に対して REQID を生成し、EIBREQID フィールドの EXEC インターフェース・ブロック (EIB) に、その REQID を保管するためです。

364 ページの図 100 はこのプログラミング手法を説明しています。

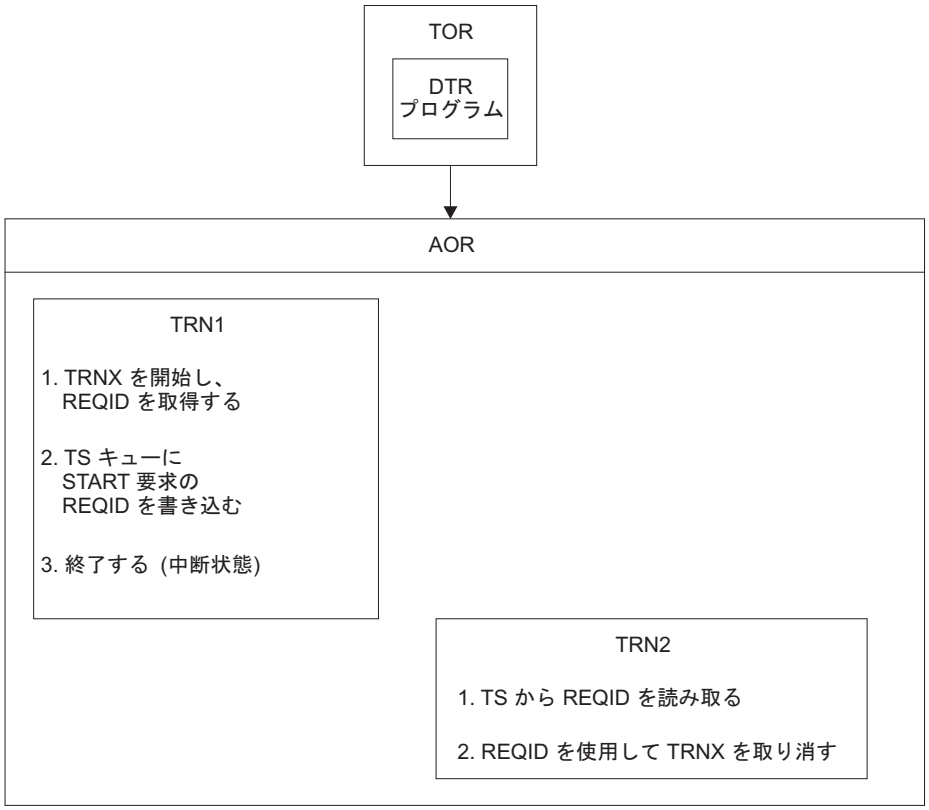


図 100. START および CANCEL REQID コマンドの使用を説明する図

このアプリケーション・プログラミング手法を使用する場合、START 要求を取り消す CANCEL コマンドは、次のいずれかの条件を満たさなければなりません。

- START コマンドが実行されたターゲット領域と同一のターゲットで発行される、または
- START コマンドが実行されたターゲット領域の SYSID を指定する、または
- START コマンドを実行したターゲット領域に常駐するリモート・トランザクションとして定義されている、TRANSID (例では TRNX) を指定する

注: START コマンドは、必ずしも、そのコマンドを発行するアプリケーション・プログラムと同じ領域で、実行されなければならないわけではありません。 START コマンドを機能伝送して (または、非端末関連 START の場合は、ルーティングして) 別の CICS 領域で実行できます。この規則は、START コマンドが最後に実行される領域に適用されます。

リモート TRANSID 手法に基づくアプリケーション設計は、2 つのターゲット領域に対してのみ作用します。CANCEL コマンドの SYSID オプションを使用するアプリケーション設計は、すべてのターゲット領域が他のターゲット領域すべてに対する接続をもっている場合の、複数のターゲット領域に対してのみ作用します (好ましくないアプリケーション設計です)。いずれの場合でも、アプリケーション・プログラムの変更が必要です。動的または分散ルーティング・プログラムでは、ルーティング・プログラムに制限を加えない限り、このプログラミング手法を正しく使用することはできません。

一般にこのことは、アプリケーション設計の保護のために、動的または分散ルーティング・プログラムが TRN2 を TRN1 と同一の領域で実行し、TRNX をその領域でのローカル・トランザクションとして定義しなければならないことを意味しています。

DELAY および CANCEL REQID コマンドの使用

この CICS アプリケーション・プログラミング手法を使用して、あるタスクは DELAY コマンドにより中断されていた別のタスクを再開することができます。DELAY 要求は、DELAY コマンドを発行するタスクとは別のタスクだけが取り消すことができ、CANCEL コマンドは DELAY コマンドに関連した REQID を指定しなければなりません。DELAY および CANCEL コマンドの両方は、この手法を使用するために REQID オプションを指定しなければなりません。

REQID の受け渡しに一時記憶キューを使用するこの手法のステップは、以下のとおりです。

1. タスク (TRN1) が、事前定義された DELAY REQID を TS キューに書き込みます。例:

   ```
   EXEC CICS WRITEQ TS
        QUEUE('DELAYQUE') FROM(reqid_value)
   ```

2. そのタスクは、REQID に *reqid_value* を使用し、DELAY コマンドを発行して別のタスクを待ちます。例:

   ```
   EXEC CICS DELAY
        INTERVAL(1000) REQID(reqid_value)
   ```

3. 別のタスクである TRN2 は、「DELAYQUE」と呼ばれる TS キューから DELAY 要求の REQID を読み取ります。
4. TRN2 は処理を完了し、DELAY 要求を取り消すことにより TRN1 を再開します。

TS キューを使用する処理は、366 ページの図 101 で説明しています。

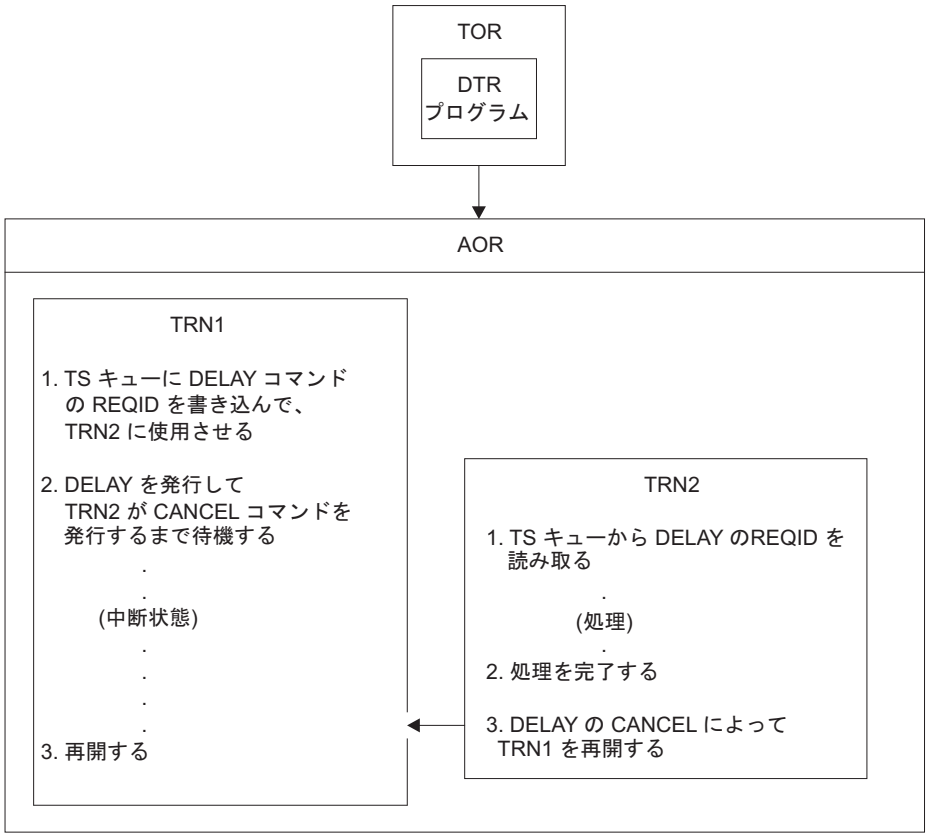


図 101. DELAY および CANCEL REQID コマンドの使用を説明する図

この手法を使用して REQID を受け渡す別の方法は、TRN1 が FROM および TERMID オプションのある START コマンドを使用して TRN2 を開始する方法です。TRN2 が開始されると、TRN2 は INTO オプション付の RETRIEVE コマンドを使用して REQID を獲得することができます。

このアプリケーション・プログラミング手法を使用する場合、DELAY 要求を取り消す CANCEL コマンドは次のいずれかの条件を満たさなければなりません。

- DELAY コマンドが実行されたのと同一のターゲット領域で発行される、または
- DELAY コマンドが実行されたターゲット領域の SYSID を指定する、または
- DELAY コマンドを実行したターゲット領域に常駐するリモート・トランザクションとして定義されている、TRANSID (例では TRN1) を指定する

リモート TRANSID 手法に基づくアプリケーション設計は、2 つのターゲット領域に対してのみ作用します。CANCEL コマンドの SYSID オプションを使用するアプリケーション設計は、すべてのターゲット領域が他のターゲット領域すべてに対する接続をもっている場合の、複数のターゲット領域に対してのみ作用します (好ましくないアプリケーション設計です)。いずれの場合でも、アプリケーション・プログラムの変更が必要です。動的または分散ルーティング環境では、ルーティング・プログラムに制限を加えない限り、このプログラミング手法を正しく使用することはできません。

端末が開始したトランザクションにより CANCEL コマンドが発行された場合、そのトランザクションが正しくないターゲット領域に動的にルーティングされる可能性があります。

POST および CANCEL REQID コマンドの使用

CICS POST コマンドは、指定されている時間の終了通知を要求するために使用されます。指定されている時間が終了した場合、別のトランザクション (TRN2) が POST 要求の CANCEL を発行して通知を強制することができます。

イベント制御ブロック (ECB) にビット・パターンを設定すると、CICS により時間制限が通知されます。通知が受信されたかどうかを判別するためには、要求側トランザクション (TRN1) が定期的に ECB をテストするか、ECB に対して WAIT コマンドを発行しなければなりません。

TS ストレージ・キューの使用は、タスク間で POST 要求の REQID を受け渡すために使用できる 1 つの方法です。

図 102 は、この手法を説明しています。

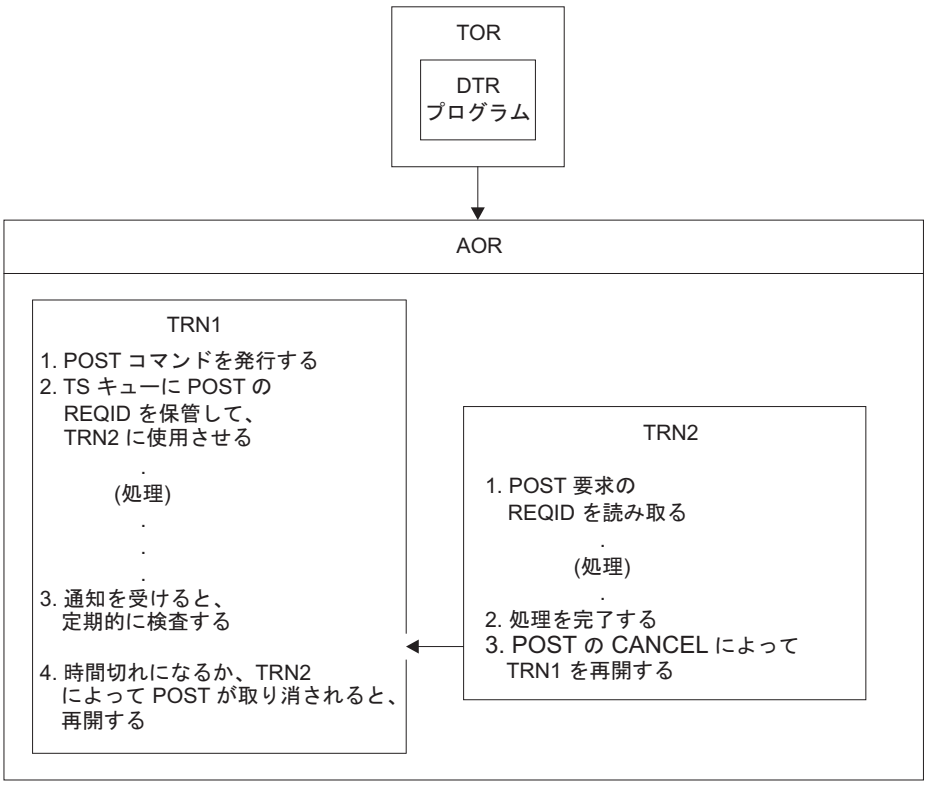


図 102. POST コマンドの使用を説明する図

この手法が使用される場合、動的または分散ルーティング・プログラムは、アプリケーション設計の保護のために、TRN2 を TRN1 と同一の CICS 領域で実行しなければなりません。

POST を発行したタスクを通知する CANCEL コマンドは、次のいずれかの条件を満たさなければなりません。

- POST コマンドが実行されたのと同一のターゲット領域で発行される、または
- POST コマンドが実行されたターゲット領域の SYSID を指定する、または
- POST コマンドを実行したターゲット領域に常駐するリモート・トランザクションとして定義されている、TRANSID (例では TRN1) を指定する

リモート TRANSID 手法に基づくアプリケーション設計は、2 つのターゲット領域に対してのみ作用します。CANCEL コマンドの SYSID オプションを使用するアプリケーション設計は、すべてのターゲット領域が他のターゲット領域すべてに対する接続をもっている場合の、複数のターゲット領域に対してのみ作用します (好ましくないアプリケーション設計です)。いずれの場合でも、アプリケーション・プログラムの変更が必要です。動的または分散ルーティング・プログラムでは、ルーティング・プログラムに制限を加えない限り、このプログラミング手法を正しく使用することはできません。

一般にこのことは、アプリケーション設計の保護のために、動的または分散ルーティング・プログラムが、TRN2 を TRN1 と同一の領域で実行しなければならないことを意味しています。

明らかに、POST を発行したタスクが CANCEL も発行してもまったく問題ありません。別のタスクが POST コマンドを取り消す場合、そのタスクはそのコマンドの特定のインスタンスを識別するために REQID を指定しなければなりません。したがって、REQID のある CANCEL コマンドは、トランザクション間の類縁性の問題を見分けるための目印となります。しかしながら、CICS が REQID を自動的に生成し、それを EIBREQID のアプリケーションに渡すので、POST コマンドに REQID を指定する必要はありません。

トランザクション間の類縁性の検出

このタスクについて

動的ルーティング環境のトランザクション類縁性を管理するには、まず、類縁性のあるトランザクションを見つけ出さなければなりません。どのようにして見つけるのでしょうか。

類縁性を検出する推奨方法は、CICS Interdependency Analyzer を使用する方法です。「*CICS Interdependency Analyzer for z/OS User's Guide and Reference*」には、このユーティリティーおよび使用方法について説明されています。

注: プログラム・リンク要求を動的にルーティングする場合には、どのプログラム (またはそれに関連したトランザクション) が類縁性を有するかを見極めなければなりません。CICS Interdependency Analyzer は、これを行う目的では使用できません。

CICS Interdependency Analyzer を使用しない場合は、以下のいずれかの方法を使用して類縁性を検出できます。

- CICS 提供のロード・ライブラリー・スキャナーである CAULMS を使用する。

  注: CICS TS for z/OS バージョン 3.2 には、以前は CICS トランザクション類縁性ユーティリティーの一部として提供されていた検出機能およびレポート機能のコンポーネントは含まれていません。これらのコンポーネント、およびロー

ド・ライブラリー・スキャナー・コンポーネントは CICS Interdependency Analyzer に取り込まれ、相互依存性および類縁性の両方を分析する機能が提供されています。ロード・ライブラリー・スキャナーの単体は CICS TS for z/OS バージョン 3.2 に残され、類縁性の可能性があるアプリケーション・プログラムに関するレポートを作成できます。

CICS ロード・ライブラリー・スキャナーの使用方法について詳しくは、CICS TS 2.3 の「*Transaction Utilities Guide*」を参照してください。

- トランザクション間の通信に使用する手法に特に注意して、アプリケーション設計を見直す。
- アプリケーション・プログラムのソースを検索し、トランザクション間の類縁性の原因となる可能性のある EXEC CICS コマンドのインスタンスを探す。
- CICS 補助トレースを分析するトレース分析プログラムを実行する。例えば、CICS 補助トレース出力をフォーマットするために ABBREV オプションのある CICS トレース・ユーティリティー・プログラム、DFHTU660 を実行すると、省略トレース・データの結果を分析して、疑わしいコマンドのインスタンスを検出することができます。

アプリケーション生成プログラムによって発生するトランザクション間の類縁性

アプリケーション生成プログラムは、特に難解なトランザクション間の類縁性の問題を生じさせます。

- 類縁性はアプリケーション・プログラマーから隠されている可能性があります。
- アプリケーション生成プログラムは、CICS のトランザクションに対して異なる概念を持っている場合があります。CICS トランザクションは動的にルーティングされるエンティティーなので、問題となるのは CICS トランザクション間の類縁性です。
- アプリケーション生成プログラムのいくつかは、1 つのアプリケーション内のすべてのトランザクションに対して、単一トランザクション・コードを使用するので、類縁性のあるトランザクションのインスタンスをルーターが選択することが困難になります。

トランザクション間の類縁性の期間と有効範囲

動的ルーティング戦略およびトランザクション間の類縁性の管理方法を計画する場合、類縁性の関係および類縁性の存続期間の概念について理解していることが大切です。動的または分散ルーティング・プログラムを設計する場合、トランザクション間の類縁性の関係および存続期間を考慮しなければなりません。なぜなら、それらがトランザクション間の類縁性の有効範囲および期間を定義するためです。言うまでもなく、動的または分散ルーティング・プログラムの理想的な状態は、トランザクション間に類縁性がまったく存在せず、動的ルーティングが利用可能なターゲット領域の選択に制限がないことです。しかしながら、トランザクション間に類縁性が存在する場合でも、これらの類縁性の範囲、つまり類縁性の関係および類縁性の存続期間により決定される範囲には限度があります。

トランザクションの類縁性の関係および存続期間を知っておくことは、動的ルーティング環境においてのトランザクションの類縁性の管理方法を決定する際に重要です。

このセクションでは、以下のトピックを取り上げます。

- 『類縁性トランザクション・グループ』
- 『関係および存続期間』

類縁性トランザクション・グループ

動的ルーティング環境において類縁性を管理するためには、まず最初に、トランザクションを類縁性により分類しなければなりません。そのための一つの方法は、トランザクションをグループに分ける方法です。1 つのグループは、トランザクション間の類縁性を持つトランザクションのセットです。したがって、各類縁性トランザクション・グループ (略して、類縁性グループ) は、互いに類縁性があるトランザクションのグループになります。類縁性グループを定義することは、動的または分散ルーティング・プログラムが、トランザクションをどのターゲット領域にルーティングすればいいかを判別できるようにする 1 つの方法です。

言うまでもなく、ある CICS ワークロードにおいて、トランザクション間の類縁性が多く存在すればするほど、動的ルーティング・プログラムは CICSplex 上のワークロードの平衡化を効率的に行えなくなります。トランザクション間の類縁性の影響を最小限にするために、類縁性グループの類縁性をその関係および存続期間で分類できます。これらの関係および存続期間の属性が類縁性の有効範囲および期間を決定します。

したがって、類縁性トランザクション・グループは、類縁性グループ ID、類縁性グループを構成するトランザクションのセット、ならびにグループに関連した類縁性の関係および類縁性の存続期間によって構成されていることが理想的です。

関係および存続期間

類縁性グループを作る場合、類縁性の関係および類縁性の存続期間の適切な属性をグループに割り当てる必要があります。類縁性の関係は、動的または分散ルーティング・プログラムが、類縁性に関連したトランザクション・インスタンスのためのターゲット領域をどのように選択するかを決定し、類縁性の存続期間は類縁性の終了時期を決定します。

類縁性グループに割り当てることができる類縁性の関係には、次の 4 つの種類があります。

- グローバル
- LU 名
- ユーザー ID
- BAPPL

グローバル関係

類縁性の関係をグローバルと定義するトランザクション・グループは、端末から開始されるグループ内のすべてのトランザクションの、**すべての** インスタンスが、START コマンドによって、あるいは CICS BTS 処理によって、類縁性の存続期間

中、同一のターゲット領域で実行されなければならないトランザクション・グループです。グローバル関係における類縁性の存続期間は次のようになります。

システム
類縁性は、ターゲット領域が存在する限り存続し、ターゲット領域が終了 (正常、即時および異常終了) すると消滅します。

永続型 類縁性はすべての CICS 再始動に拡張されます。これは、すべてのトランザクション間の類縁性の中で最も制約の多いものです。CICSPlex SM を実行している場合は、ワークロードを使用する CICSplex の管理に関係している CMAS がアクティブである限り、この類縁性は続きます。

存続期間が永続型であるグローバル・トランザクション間の類縁性の 1 つの例は、トランザクションがローカルでリカバリー可能な一時記憶キューを使用して (読み取りまたは書き込み、あるいはその両方を行う)、TS キュー名が端末から派生**しない**ものです。(キューがローカルである CICS 領域では、TS キューをリカバリー可能にしか指定できません。)

通常、この類縁性カテゴリーに属するトランザクションは、動的ルーティングに適したものではないので、それらのトランザクションは静的にルーティングするようにしてください。

グローバル関係の例を図 103 に示します。

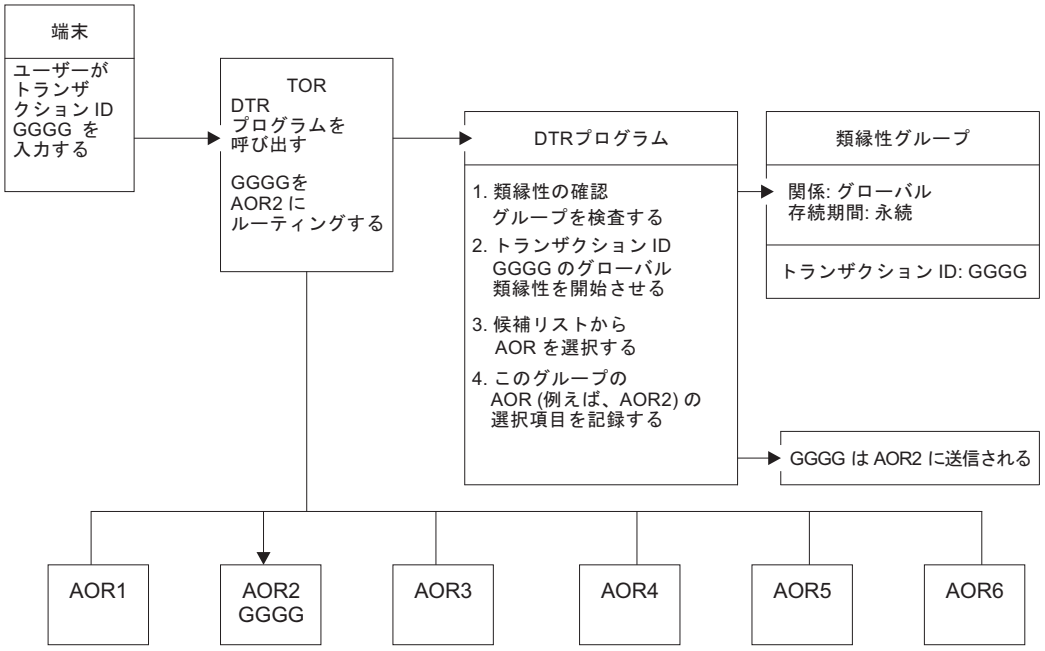


図 103. グローバル関係を有し、存続期間が永続であるトランザクション間の類縁性の管理

この例では、トランザクション GGGG は、永続的でグローバルな類縁性の関係を有するグループに定義されます。任意の端末からのトランザクション ID、GGGG の最初のインスタンスが、存続期間が永続である類縁性を開始させます。GGGG の最初のインスタンスは、任意の適切なターゲット領域にルーティングできます。こ

の例では、AOR2 は AOR1 から AOR6 までの指定可能な範囲から選択されますが、任意の端末からの他のすべてのインスタンスも、同じ領域である AOR2 にルーティングされなければなりません。

LU 名 (端末) 関係

類縁性の関係を LU 名と定義するトランザクション・グループは、***同一の端末*** に関連したグループ内の、すべてのトランザクションの**すべての** インスタンスが、類縁性の存続期間中同一のターゲット領域で実行されなければならない、トランザクション・グループです。LU 名 関係における類縁性の存続期間は次のようになります。

疑似会話型
: 類縁性は、疑似会話中存続し、端末上での疑似会話が終了すると消滅します。個々のトランザクションは、END の疑似会話モードではなく、EXEC CICS RETURN TRANSID で終了しなければなりません。

ログオン型
: 類縁性は、端末が CICS にログオンされている限り存続し、端末がログオフすると消滅します。

システム
: 類縁性は、ターゲット領域が存在する限り存続し、ターゲット領域が終了 (正常、即時および異常終了) すると消滅します。

永続型 類縁性はすべての CICS 再始動に拡張されます。CICSPlex SM を実行している場合は、ワークロードを使用する CICSplex の管理に関係している CMAS がアクティブである限り、この類縁性は続きます。

限界指定型
: 類縁性は、END の疑似会話方式によるトランザクションに遭遇するまで続きます。

LU 名関係を有するトランザクションの典型的な例は次のようになります。

- 疑似会話を行うトランザクション間でのデータの受け渡しにローカル TS キューを使用し、
- TS キュー名の一部分に端末名を使用している (TS キュー名については 358 ページの『リモート・キューの命名規則』を参照してください)。

このタイプのトランザクションは、端末および疑似会話の存続期間に関連する類縁性グループに分類することができます。動的ルーティング・プログラムが、最初のトランザクションを、特定の端末 (LU 名) により開始された疑似会話で検出した場合、そのトランザクションに有効な任意のターゲット領域に、自由にそのトランザクションをルーティングすることができます。しかしながら、**同一の端末**で開始されたその類縁性グループ内の後続のトランザクションはいずれも、疑似会話を開始したトランザクションと同じターゲット領域にルーティングされなければなりません。類縁性が消滅すると (指定端末における疑似会話の終了時)、動的ルーティング・プログラムは再び、最初のトランザクションを任意のターゲット領域候補に自由にルーティングすることができます。

この型の類縁性は管理可能なもので、動的トランザクション・ルーティングに厳密な制約を課しません。したがって、多くの CICSplex に共通に見られるものです。

この類縁性は、動的ルーティング・プログラムにより簡単に管理できるもので、動的ルーティングの使用を禁止する必要はありません。

この類縁性の例を図 104 に示します。

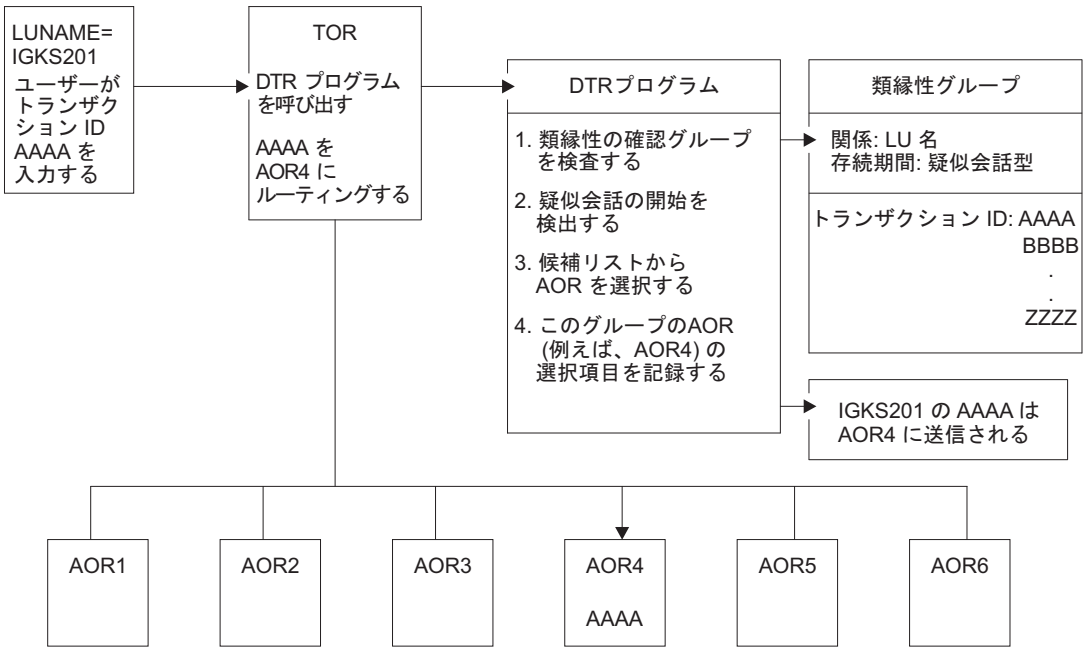


図 104. LU 名関係を有し、存続期間が疑似会話型であるトランザクション間の類縁性の管理

この例では、端末から開始されたトランザクション AAAA の各インスタンスが、存続期間が疑似会話型である類縁性を生じさせます。AAAA は、適切な任意のターゲット領域 (AOR1 から AOR6 まで) にルーティングできますが、同じ端末での同じ疑似会話型のグループ内の他のトランザクション (この例では IGKS201) は、AAAA 用に選択したターゲット領域にルーティングされなければなりません。

ユーザー ID 関係

類縁性の関係をユーザー ID と定義するトランザクション・グループは、端末から、START コマンドあるいは CICS BTS アクティビティーにより開始され、**同じユーザー ID** で実行されるトランザクションの**すべての** インスタンスが、類縁性の存続期間中、同一のターゲット領域で実行されなければならないトランザクション・グループです。ユーザー ID 関係における類縁性の存続期間は次のようになります。

疑似会話型

類縁性は、疑似会話中存続し、そのユーザー ID の疑似会話が終了すると消滅します。個々のトランザクションは、END の疑似会話モードではなく、EXEC CICS RETURN TRANSID で終了しなければなりません。

サインオン

類縁性は、ユーザーがサインオンしている限り存続し、ユーザーがサインオフすると消滅します。各ユーザー ID に対して一人のユーザーだけが許可されている状況でのみ、この存続期間が可能です。サインオン存続期間は、複数のユーザーが同一のユーザー ID を使用して同時に (別々の端末で) サインオンできるように許可されている場合は検出されません。

システム
類縁性は、ターゲット領域が存在する限り存続し、ターゲット領域が終了 (正常、即時および異常終了) すると消滅します。

永続型 類縁性はすべての CICS 再始動に拡張されます。CICSPlex SM を実行している場合は、ワークロードを使用する CICSplex の管理に関係している CMAS がアクティブである限り、この類縁性は続きます。

限界指定型
類縁性は、END の疑似会話方式によるトランザクションに遭遇するまで続きます。

ユーザー ID 関係を有するトランザクションの典型的な例は、TS キューのようなリソースを識別するためにユーザー ID が動的に使用される場合です。このカテゴリーの類縁性の中で制約が最も少ないものは、ユーザーがサインオンしている間だけ存続するものです。

ユーザー ID 関係を有し、存続期間がサインオンである類縁性グループの例を、図 105 に示します。

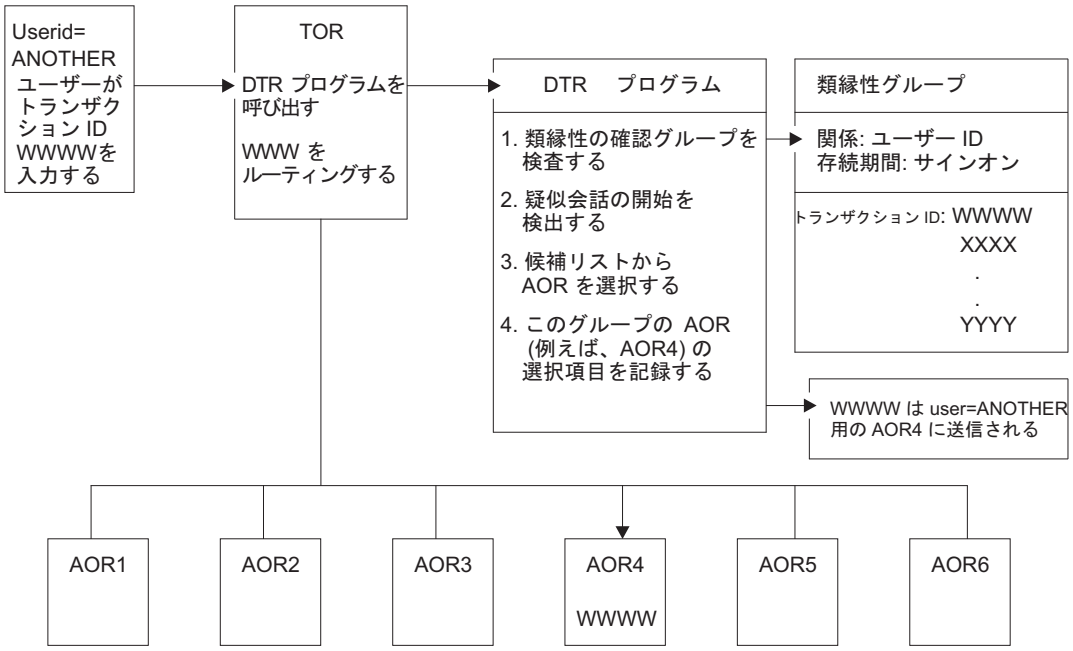


図 105. ユーザー ID 関係を有し、存続期間がサインオンであるトランザクション間の類縁性の管理

この例では、端末からのトランザクションのすべてのインスタンスが、存続期間がサインオンである類縁性を生じさせます。AAAA は、適切な任意のターゲット領域 (AOR1 から AOR6 まで) にルーティングできますが、同じユーザー・グループ内の他のトランザクション (この例では ANOTHER) は、グループ内のトランザクションの最初のインスタンス用に選択されるターゲット領域に、ルーティングされなければなりません。

BAPPL 関係

類縁性の関係を BAPPL と定義するトランザクション・グループは、同じ BTS 処理に関連したグループ内の、すべてのトランザクションの**すべての** インスタンス

が、同一のターゲット領域に誘導されなければならないトランザクション・グループです。 BAPPL 関係における類縁性の存続期間は次のようになります。

処理型 類縁性は、関連した処理が存在する限り続きます。

アクティビティー型
類縁性は、関連したアクティビティーが存在する限り続きます。

システム
類縁性は、ターゲット領域が存在する限り存続し、ターゲット領域が終了 (正常、即時および異常終了) すると消滅します。

永続型 類縁性はすべての CICS 再始動に拡張されます。CICSPlex SM を実行している場合は、ワークロードを使用する CICSplex の管理に関係している CMAS がアクティブである限り、この類縁性は続きます。

BAPPL 関係を有するトランザクションの典型的な例としては、ローカル一時記憶キューを使用して、BTS アクティビティーまたは処理内のトランザクション間でデータの受け渡しを行う場合があります。

BAPPL 関係がある類縁性グループの例は、図 106 に示されています。

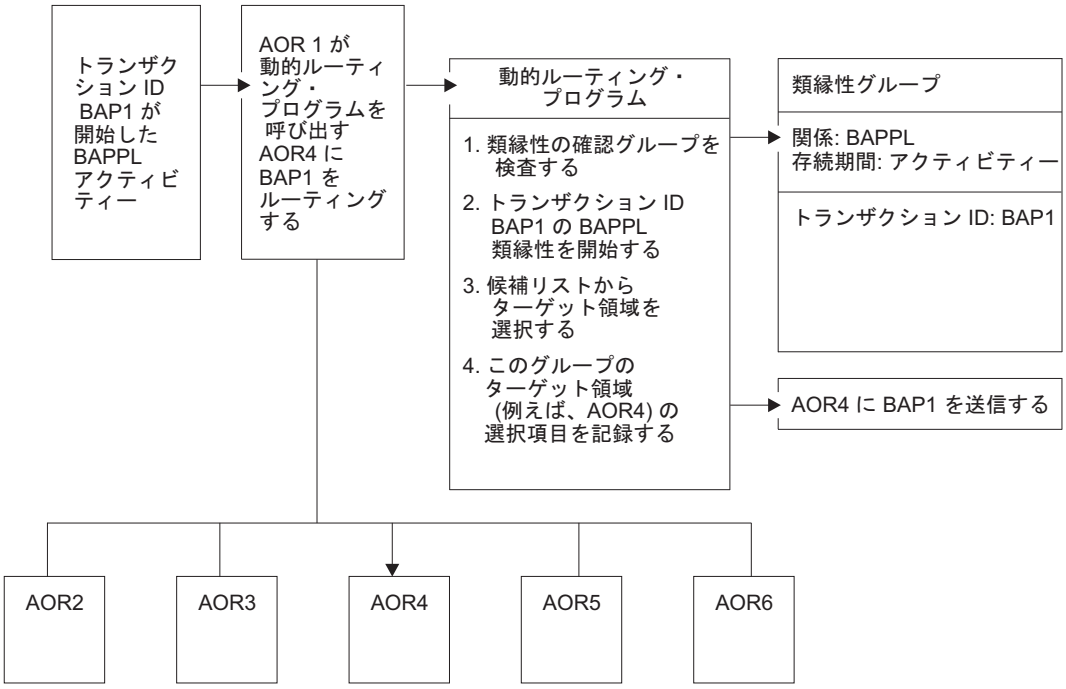


図 106. BAPPL 関係を有し、存続期間がアクティビティーであるトランザクション間の類縁性の管理

この例では、BTS トランザクションの最初のインスタンスである BAP1 が、BAPPL アクティビティー類縁性を開始します。BAP1 の最初のインスタンスは、適切な任意のターゲット領域 (AOR1 から AOR6) にルーティングできますが、アクティビティーの他のインスタンスはすべて、BAP1 用に選択したターゲット領域にルーティングされなければなりません。

BTS 自体は類縁性をもたらすわけではなく、類縁性をもたらすプログラミング手法を抑制するのですが、類縁性をもたらす可能性のある既存コードをサポートしてい

ます。ワークロード管理に対しては、このような類縁性を定義しなければなりません。個々の類縁性の存続期間を指定することが、特に重要です。これを指定しないと、ワークロード管理のルーティング・オプションを不必要に制限する可能性があります。

与えられたアクティビティーが、同期にも非同期にも実行できることに注意してください。ワークロード管理は、非同期に行われる呼び出しを引き受けることができるだけです。さらに、これらの類縁性、特にアクティビティーと処理の類縁性はできる限り発生させないようにしてください。なぜなら、これらの類縁性は、BTS セット上で同期化されるためです。そうなると、システムに重大なパフォーマンス上の影響が生じる可能性があります。

CICSPlex SM では、類縁性が維持できる最長の期間は、ワークロードに関連した CMAS がアクティブにある間、つまり PERMANENT だということにも、注意してください。完全なシステム障害がある場合、またはシャットダウンが予定されている場合は、類縁性は失われますが、CICS のアクティビティーは BTS RLS データ・セットによりリカバリーされます。

第 23 章 リカバリーの設計

CICS では、CICS を実行している間のイベントのリカバリーまたは再構成、あるいはデータ変更を支援する、以下の 2 つの手法が使用できます。

- 『ジャーナル処理』
- 380 ページの『同期点処理』

名前付きカウンター・リカバリーの手法についての説明は、737 ページの『第 56 章 名前付きカウンターのリカバリー』 に記載されています。

ジャーナル処理

CICS は、CICS 処理中の**ジャーナル**の作成および管理を行うための機能を提供します。ジャーナルには、ユーザーがイベントまたはデータ変更を後で再構成するために必要とする、ありとあらゆるデータを入れることができます。例えば、ジャーナルは監査証跡、データベース更新および追加の変更ファイル、またはシステムを通じて受け渡しするトランザクションの記録 (**ログ**と呼ばれることもあります) として使用されます。各ジャーナルはどのタスクからも書き込むことができます。

ジャーナル管理コマンドを使用してアプリケーション・プログラマーができることは、次のとおりです。

- ジャーナル・レコードの作成 (WRITE JOURNALNAME または WRITE JOURNALNUM コマンド)
- ジャーナル出力との (その完了を待ったうえでの) 同期化 (WAIT JOURNALNAME または WAIT JOURNALNUM コマンド)

ジャーナル管理コマンドの実行中に起こる例外条件の取扱方法は、383 ページの『第 24 章 例外条件の取り扱い』に記載されています (以前の JFILEID オプションは、互換性の目的のためにのみサポートされています)。

ジャーナル・レコード

各ジャーナルは、ジャーナル ID と呼ばれる名前または番号によって識別されます。この番号の範囲は 1 から 99 です。DFHLOG という名前は、システム・ログと呼ばれるジャーナル用に予約されています。

ジャーナル・レコードを作成した場合には、データがジャーナル・バッファー域に転送されます。すべてのバッファー・スペースおよびジャーナル操作に必要なその他の作業域は、CICS が獲得および管理を行います。ユーザー・タスクは、ジャーナルに書き込むデータのみを提供します。ログ・マネージャーは、出力サービスを要求しているアプリケーション・プログラマーがジャーナル・レコードの詳細レイアウトおよび詳細な内容に関与する必要がないように設計されています。プログラマーは、使用するジャーナル、指定するユーザー・データの内容、および提供するユーザー ID の内容さえ分っていれば十分です。

© Copyright IBM Corp. 1989, 2009

ジャーナル出力の同期

WAIT オプションを指定した JOURNALNAME または WRITE JOURNALNUM コマンドを発行して同期ジャーナル・レコードを作成する場合、要求タスクは出力が完了するまで待機することができます。このように指定することで、アプリケーション・プログラマーは、ジャーナル・レコードがジャーナルと関連した外部ストレージ装置に確実に書き込まれるようにしてから、処理を続行することができます。このタスクは出力操作と**同期する**と言います。

アプリケーション・プログラマーは、非同期ジャーナル出力を要求することもできます。これにより、ジャーナル・バッファー域にジャーナル・レコードが作成されますが、要求タスクは制御を持ち続けるので、他の処理を続行することができます。タスクは、その後で、WAIT JOURNALNAME または WAIT JOURNALNUM コマンドを発行して出力の完了を検査し待機する (すなわち、同期する) ことができます。

注: 場合によって、SHUTDOWN IMMEDIATE が外部ストレージではなくログ・マネージャー・バッファーに書き込まれると、ユーザーのジャーナル・レコードが失われる可能性があります。これは、CICS シャットダウン援助トランザクション (CESD) が、通常シャットダウン中に SHUTDOWN IMMEDIATE を強制した場合にも起こります。なぜなら、通常シャットダウンがハングするためです。ジャーナル・レコードを失う危険を回避するために、プログラムを終了する前に、定期的に CICS WAIT JOURNALNUM 要求を出すことをお勧めします。

WAIT を使用しないと、バッファーがデータでいっぱいになるまで、または関連のない別のアクティビティーがバッファーの固定化を要求し、それにより、入出力操作の回数が減るまで、CICS はログ・ストリームにデータを書き込みません。また、WAIT を使用すると、CICS がログ構造のバッファー・サイズを正確に計算することが難しくなります。CF ログ・ストリームの場合、これによって、カップリング・ファシリティーにおけるストレージを効率的に利用できなくなる可能性があります。

指定されたジャーナルの CICS バッファー・スペースにジャーナル・レコードを作成するという基本的な処理は、以下のいずれかのイベントが起こるまで続きます。

- システム・ログに対して
  - システムが、整合性および今後の緊急再始動の許可を要求するとき
  - ログ・ストリーム・バッファーがいっぱいになった場合
- ユーザー・ジャーナルに対して
  - ログ・ストリーム・バッファーがいっぱいになった場合 (または、ジャーナルが SMF に常駐する場合は、ジャーナル・バッファーがいっぱいになった場合)
  - ジャーナル・レコードを出力するために WAIT オプションを指定した要求が出た場合 (任意のタスクから)
  - EXEC CICS SET JOURNALNAME コマンドが発行された場合
  - EXEC CICS DISCARD JOURNALNAME コマンドが発行された場合
  - 同一のログ・ストリームにマップする他のジャーナルに対して上記のいずれかがおこった場合
  - 通常シャットダウンの場合

- 順方向リカバリー・ログに対して
  - ログ・ストリーム・バッファーがいっぱいになった場合
  - 同期点になったとき (第 1 段階)
  - ファイルを閉じたとき
- 自動ジャーナルに対して
  - ログ・ストリーム・バッファーがいっぱいになった場合
  - ジャーナル・レコードを出力するために WAIT オプションを指定した要求が出た場合 (任意のタスクから)
  - ファイルを閉じたとき
- ログのログ (DFHLGLOG) に対して
  - ファイルの OPEN および CLOSE 要求時

上記のいずれかがおこった場合、バッファー内に存在するジャーナル・レコードは、非同期要求によって据え置かれた出力を含め、1 つのブロックとしてすべてログ・ストリームに書き込まれます。

ジャーナル出力を据え置くことによって得られる利点は、次のとおりです。

- 待ち時間が短くなることで、トランザクションの応答時間が改善される。
- ホスト・システムでの物理入出力要求の負荷が軽減される。
- ログ・ストリームに、少数の大きいブロックが含まれるようになることで、1 次ストレージがより効率的に利用される。

しかし、これらの利点を達成可能なのは、かなり複雑なプログラミングを行った場合だけです。ジャーナル出力との同期を制御するように計画およびプログラミングを行う必要があります。アプリケーション・プログラムでは、ジャーナル・レコードのデータ内容およびその使用方法によっては追加の決定を行う必要があります。いずれにしても、ジャーナル出力の据え置きの全面的な恩恵にあずかれるのはジャーナルの負荷が高い場合だけです。

要求時点で利用可能なジャーナル・バッファー・スペースがジャーナル・レコードを入れるのには不十分な場合には、条件 NOJBUFSP が起こります。この条件に関する HANDLE CONDITION コマンドがいずれもアクティブになっていない場合は、要求タスクは制御を失い、現行バッファーの内容が書き出され、制御が要求タスクに戻る前に、解放済みバッファー・スペースにジャーナル・レコードが作成されます。

要求タスクが制御を失いたくない場合 (例えば、他のタスクが制御を獲得する前にハウスキーピングを実行する必要がある場合など) には、HANDLE CONDITION コマンドを発行する必要があります。NOJBUFSP 条件が起こった場合には、その要求に関するジャーナル・レコードは作成されず、制御は、HANDLE CONDITION コマンドで指定された位置で要求プログラムに直接返されます。要求プログラムは、ジャーナル出力要求を再発行する前に、任意のハウスキーピングを実行することができます。

ジャーナル・コマンドは、ジャーナルの即時出力または出力の据え置きの原因となります。システム・ログ・レコードは、要求に JOURNALNAME(DFHLOG) を指定することによって他のレコードと区別されます。ユーザー・ジャーナル・レコード

は、別の JOURNALNAME または JOURNALNUM を使用して作成されます。すべてのレコードに、ジャーナル・タイプ ID (JTYPEID) がなければなりません。ユーザー・ジャーナリングをシステム・ログに対して行う場合、ジャーナル・タイプ ID (高位ビットの設定による) は、ウォーム・リスタート時または緊急リスタート時において、グローバル・ユーザー出口 XRCINPT に対するこれらの表示も制御します。ログの逆方向スキャン中に表示されるレコードは、次のようになります。

- 未了タスクまたは未確定タスクのみ (高位ビットの指定がオフの場合)
- スキャン終了までに確認されたすべてのレコード (高位ビットの指定がオンの場合)

ジャーナル・レコードの形式および構造については、「*CICS Customization Guide*」のCICS フォーマット・ジャーナル・レコードの構造と内容を参照してください。リカバリー処理の背景情報および説明については、「*CICS Recovery and Restart Guide*」のCICS コールド・スタートにある緊急再始動のセクションを参照してください。

同期点処理

CICS タスクの異常終了または CICS システムに障害が起きたときのリカバリーを容易にするために、システム・プログラマーは、CICS テーブルの生成時に、特定のリソース (例えば、ファイル) をリカバリー可能として定義することができます。タスクが異常終了した場合には、これらのリソースは、タスクの開始時点の状態に復元され、その後で再実行することができます。タスクと関連したリソースを復元する処理は、**バックアウト**と呼ばれます。

個別のタスクが失敗した場合には、バックアウトは動的トランザクション・バックアウト・プログラムによって実行されます。CICS システムが失敗した場合には、バックアウトは緊急再始動処理の一部として実行されます。一般に、これらの機能はアプリケーション・プログラムのコーディングには影響しませんが、これらの機能については、「*CICS Recovery and Restart Guide*」のSTART=INITIAL パラメーターを指定した CICS の始動を参照してください。

しかし、長時間稼働プログラムの場合には、多くの変更を行い、ある期間それを積み重ね、タスクまたはシステムの障害が起こった場合にそれらを取り消せなくなってしまう可能性があるのは望ましくありません。この可能性は、SYNCPOINT コマンドを使用してプログラムを作業単位 (UOW) と呼ばれる論理的に独立した部分に分割することによって回避することができます。UOW の終わりは、同期点 (**syncpoint**) と呼ばれます。同期点について詳しくは、「*CICS Recovery and Restart Guide*」のリカバリー機能および再始動機能を参照してください。

同期点を過ぎてからタスクが完了するまでの間に障害が発生した場合は、同期点以降に行われた変更だけがバックアウトされます。

SYNCPOINT コマンドの代わりに、SAA リソース・リカバリー・インターフェースを使用することができます。これにより、既存の CICS リソース・リカバリー・サービスに対する代替 API が用意されます。SAA リソース・リカバリー・インターフェースは、共通 API の整合性が有用であると考えられる複数の SAA プラットフ

ォームを含むネットワークで使用することができます。CICS システムでは、SAA リソース・リカバリー・インターフェースによって EXEC CICS API と同じ機能が提供されます。

制約事項: フル機能 SAA リソース・リカバリーには、CICS インプリメンテーションではサポートされていないいくつかの戻りコードが用意されています (「*SAA Common Programming Interface for Resource Recovery Reference*」にある CICS の付録を参照してください)。

SAA リソース・リカバリー・インターフェースは、以下の 2 つの呼び出しタイプを持つ呼び出しインターフェースとしてインプリメントされます。

SRRCMIT
コミット - SYNCPOINT コマンドに相当。

SRRBACK
バックアウト - SYNCPOINT ROLLBACK コマンドに相当。

SAA リソース・リカバリー・インターフェース・インターフェースについて詳しくは、「*SAA Common Programming Interface for Resource Recovery Reference*」を参照してください。

UOW は、保護リソースに関してだけでなく、実行フローに関しても論理的に完全に独立していなければなりません。一般に、UOW は SEND および RECEIVE コマンドによってバインドされた完全な会話型操作を構成します。ブラウズは、UOW の別の例であり、ENDBR コマンドは同期点より前になければなりません。

同期点と見なされる場合の DL/I 終了呼び出しだけでなく、SYNCPOINT コマンドを実行しても、CICS は DL/I 終了呼び出しを発行します。後続の UOW で DL/I PSB が必要な場合には、プログラム制御ブロック (PCB) 呼び出しまたは SCHEDULE コマンドを使用して再スケジュールする必要があります。

分散プログラム・リンク (DPL) によって、サーバー・プログラムで同期点を取ることを指定し、制御をクライアントに戻す前に、サーバーのリソースをコミットすることができます。これは、LINK コマンドで SYNCONRETURN オプションを使用することによって可能になります。SYNCONRETURN オプションに関するプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」の 5 (415 ページ) および の『サーバー・プログラムの SYNCONRETURN オプション』を参照してください。

SYNCPOINT コマンドを処理する時に、開始されたが完了しない BMS 論理メッセージは、暗黙 SEND PAGE コマンドによって強制的に完了されます。しかし、最初のページが不完全な論理メッセージは失われるので、このコマンドは信頼できません。SYNCPOINT コマンドの前またはトランザクションの終了の前に、明示 SEND PAGE コマンドもコーディングする必要があります。

トランザクション再始動の候補となるトランザクションにおいて同期点を発行する場合は、システム・プログラマーに相談してください。

第 24 章 例外条件の取り扱い

ユーザー・アプリケーションの 1 つで EXEC CICS コマンドを処理するたびに、CICS は起こった内容を通知する条件、あるいは戻りコードを自動的に作成します。CICS EXEC インターフェース・プログラムによってユーザー・アプリケーションに返される、通常は NORMAL であるこの条件はいくつかの方法で受け取ることができます。これは、コマンドに RESP オプションを使用してその内容を保持することができるので、RESP 値と呼ばれることもあります。あるいは、この値を EXEC インターフェース・ブロック (EIB) から読み取って入手することもできます。

通常外の何かが起こった場合には、NORMAL 以外の条件を意味する*例外条件* を受け取ります。この条件をテストすることによって、起こったことの内容と、おそらく、その理由を見極めることができます。

多くの例外条件は、例外条件と関連した追加 (RESP2) の値を持っており、これが詳細情報を提供します。この RESP2 値は、コマンドで、RESP オプションに加えて RESP2 オプションを指定するか、EIB から読み取ることで入手できます。

条件が NORMAL ではなくとも、すべてがエラー状態を示しているわけではありません。例えば、ファイルのブラウズ中に READNEXT コマンドに対して ENDFILE 条件を受け取った場合、その内容は予定どおりのものにすぎません。起こる可能性があるすべての条件およびそれが起こりうるコマンドについては、「*CICS アプリケーション・プログラミング・リファレンス*」のマニュアルを参照してください。

このセクションでは、以下について説明します。

- 『デフォルト CICS 例外処理』
- 384 ページの『インライン・コードによる例外条件の処理』
- 387 ページの『デフォルト CICS 例外処理の変更』

デフォルト CICS 例外処理

アプリケーションが C、C++、または Java 以外の言語で書かれており、他の方法を指定しない場合、CICS は例外条件が発生したときには常に組み込み例外処理を使用します。アプリケーションが C または C++ で書かれている場合は、例外条件が発生したときに CICS そのものは何も処理を行わず、アプリケーションがそれを処理するようにそのままにしておきます。例外条件の処理については、384 ページの『インライン・コードによる例外条件の処理』を参照してください。

CICS による最も一般的な処置は、何らかのタイプの異常終了を発生させることです。各条件およびコマンドの振る舞いについて詳しくは、「*CICS アプリケーション・プログラミング・リファレンス*」および「*CICS System Programming Reference*」を参照してください。

CICS デフォルト例外処理で十分な場合もあり、その場合にはなにも行う必要はありません。ほとんどの場合は、なにか他の処置を必要とするでしょう。

© Copyright IBM Corp. 1989, 2009

例外条件のデフォルト CICS 処理をオフにするには、以下のようにいくつかの方法があります。

- 特定の EXEC CICS コマンド呼び出しで NOHANDLE オプションを指定して、例外条件のデフォルト CICS 処理をオフにする。
- あるいは、コマンドに RESP オプションを指定して、例外条件のデフォルト CICS 処理をオフにする。このオプションは、NOHANDLE と同様の方法でデフォルト CICS 処理をオフに切り替えます。また、これにより、RESP の引数で名前を指定した変数が、コマンドによって返される条件の値によって更新されます。この詳細は、『インライン・コードによる例外条件の処理』に説明があります。
- C または C++ でアプリケーション・プログラムを作成する。

デフォルト CICS 例外処理をオフにした場合には、コマンド呼び出しで起こる可能性があるすべてのことを、ユーザー・プログラムで確実に処理する必要があります。

推奨される方法ではありませんが、他のアクションを指定する従来の方法は、C または C++ 以外の言語でプログラミングしている場合に限り使用可能です。この方法は、HANDLE ABEND、HANDLE CONDITION および IGNORE CONDITION の各コマンドを組み合わせて、デフォルトの CICS 例外処理を変更するというものです。この説明は、387 ページの『デフォルト CICS 例外処理の変更』にあります。

インライン・コードによる例外条件の処理

例外条件を処理するこの方式は、新規アプリケーションの場合にお勧めします。プログラムが C または C++ 言語で書かれている場合は、この方式のみが利用可能です。プログラムが C または C++ で書かれていない場合は、NOHANDLE オプションを使用するか、または EXEC CICS コマンドで RESP オプションを指定することになり、CICS はそのデフォルト例外処理が実行できなくなります。さらに、RESP オプションは、処置を改善するために例外条件の値をユーザー・プログラムで直接利用可能にします。

このタスクについて

プログラムが C または C++ で書かれている場合は、例外条件の処理にはインライン・コードのみが使用できます。

NOHANDLE オプションまたは RESP オプションを使用する場合は、コマンドの実行中に起こる可能性があるすべての条件を、ユーザー・プログラムで確実に処理する必要があります。RESP 値は、ユーザー・プログラムが処理内容を決めるため、および EXEC インターフェース・ブロック (EIB) に入る情報のうちで必要になる可能性がある詳細情報を決めるために利用可能です。特に、RESP2 値は EIB のフィールドの 1 つに入ります。 EIB の詳細については、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。代わりに、プログラムがコマンドの RESP2 を指定する場合は、直接 CICS によって RESP2 値が返されます。

DFHRESP 組み込み変換プログラム機能は、RESP 値のテストを非常に容易にします。これにより、ユーザーは RESP 値を記号的に調べることができます。この方法

は、コードを読む者にとってわかりにくい 2 進数値を調べるより容易です。

RESP オプションおよび RESP2 オプションの使用方法

RESP の引数は、ユーザー定義のフルワード 2 進データ域 (長整数) です。この引数には、コマンドから戻る時に、起こった可能性がある条件と対応する値が入ります。通常、この値は DFHRESP(NORMAL) です。

COBOL と PL/I における RESP および DFHRESP の使用

これは、RESP オプションを使用する COBOL で書かれた EXEC CICS 呼び出しの例です。 PL/I の例も同様ですが、PL/I の場合には、終わりは END-EXEC ではなく「;」になります。

```
EXEC CICS WRITEQ TS FROM(abc)
          QUEUE(qname)
          NOSUSPEND
          RESP(xxx)
          END-EXEC.
```

DFHRESP を使用して RESP 値を検査する例は次のとおりです。

```
IF xxx=DFHRESP(NOSPACE) THEN ...
```

C と C++ における RESP および DFHRESP の使用

RESP 変数の宣言などの RESP オプションを使用する、C における EXEC CICS 呼び出しの例を以下に示します。

```
long response;
.
.
.
EXEC CICS WRITEQ TS FROM(abc)
          QUEUE(qname)
          NOSUSPEND
          RESP(response);
```

DFHRESP を使用して RESP 値を検査する例は次のとおりです。

```
if (response == DFHRESP(NOSPACE))
{
.
.
.
}
```

アセンブラーにおける DFHRESP の使用

アセンブラー言語で RESP 値をテストする例は次のとおりです。

```
  CLC   xxx,DFHRESP(NOSPACE)
  BE    ...
```

C における例外処理の例

次の例は、BMS マップを受け取り、例外条件を処理するために使用することができる代表的な機能です。

```
int ReadAccountMap(char *mapname, void *map)
{
    long     response;
    int      ExitKey;
    EXEC CICS RECEIVE MAP(mapname)
              MAPSET("ACCOUNT")
              INTO(map)
              RESP(response);
    switch (response)
    {
    case DFHRESP(NORMAL):
        ExitKey = dfheiptr->eibaid;
        ModifyMap(map);
        break;
    case DFHRESP(MAPFAIL):
        ExitKey = dfheiptr->eibaid;
        break;
    default:
        ExitKey = DFHCLEAR;
        break;
    }
    return ExitKey;
}
```

図 107. C における例外処理の例

ReadAccountMap 関数は、次の 2 つの引数をもっています。

1. *mapname* は、受け取るマップの名前が入る変数です。
2. *map* は、マップを書き込む先のメモリー内の区域のアドレスです。

RESP 値は、*response* で返されます。*response* の宣言は適切な型の自動変数をセットアップします。

EXEC CICS ステートメントは、変数 *response* によって保持される条件の値とともに、*mapname* で名前が指定されているマップ・セット ACCOUNT のマップを、変数 *map* が指しているメモリーの区域の中に読み取ることを要求します。

条件処理は if ステートメントを使用して行うことができます。ただし、読みやすくするために、この例のように、if ... else ステートメントの組み合わせの代わりに、switch ステートメントを使用する方がいい場合が多くあります。プログラム実行時間への効果はごくわずかです。

次の 2 つの条件は特定の場合です。

1. 条件 NORMAL は、通常であると思われる状態です。ここの例で、条件 NORMAL が検出された場合には、この機能はユーザーが CICS に戻るために押したキーを判別し、この値を ExitKey に渡します。次に、プログラムは、ModifyMap によってメモリーに保持されるマップを更新します。以後、これついては何もする必要はありません。
2. 条件 MAPFAIL はユーザーがなにも画面を更新しなかったことを意味し、まったく正常で、ここでは特別に取り扱われます。この場合、プログラムは再び ExitKey を更新しますが、ModifyMap は呼び出しません。

この例では、他のすべての条件はエラーとして保持されています。例では、ExitKey を DFHCLEAR (ユーザーが画面を消去した場合に設定されるものと同じ値) に設定

してから、これを呼び出し側プログラムに返します。ReadAccountMap からの戻りコードを検査することによって、呼び出し側のプログラムはマップが更新されず、改善処置が必要なことを認識します。

COBOL での例外処理の例

次の例は、BMS マップを受け取り、例外条件を処理するために使用することができる代表的な機能です。

```
03   RESPONSE                           PIC S9(8)  BINARY.
03   EXITKEY                            PIC X.
EXEC CICS RECEIVE MAP(MAPNAME)
          MAPSET('ACCOUNT')
          INTO(MAP)
          RESP(RESPONSE)
          END-EXEC.
IF (RESPONSE NOT = DFHRESP(NORMAL)) AND
   (RESPONSE NOT = DFHRESP(MAPFAIL))
    MOVE DFHCLEAR TO EXITKEY
ELSE
    MOVE EIBAID TO EXITKEY
    IF RESPONSE = DFHRESP(NORMAL)
        GO TO MODIFYMAP
    END-IF
END-IF.

MODIFYMAP.
```

図 108. COBOL での例外処理の例

MAPNAME は、受け取るマップの名前が入る変数です。

RESP 値は、RESPONSE で返されます。RESPONSE は、データ・セクションでフルワード 2 進変数として宣言されます。

EXEC CICS ステートメントは、変数 *RESPONSE* によって保持される条件の値とともに、*MAPNAME* で名前が指定されているマップ・セット ACCOUNT のマップを読み取ることを要求します。

条件処理は IF ... ステートメントを使用して実行されます。条件が NORMAL でも MAPFAIL でもない場合には、プログラムはユーザーが画面を消去したものとして動作します。

条件が NORMAL または MAPFAIL のいずれかの場合には、プログラムは、ユーザーが画面を終了するために押したキーの値を EXITKEY に保管します。さらに、条件が NORMAL の場合には、プログラムは MODIFYMAP へ分岐して、追加の機能を実行します。

デフォルト CICS 例外処理の変更

CICS が例外と異常終了を処理する方法を変更するには、アプリケーション・プログラムで 3 つのコマンドを使用できます。コマンドは HANDLE CONDITION、IGNORE CONDITION、および HANDLE ABEND です。

このタスクについて

注: これらのコマンドは、C、C++、または Java プログラムでは使用できません。この情報は、こうした言語で作成されたプログラムには適用されません。

HANDLE CONDITION
条件が起こった場合に制御を渡すラベルを指定します。

IGNORE CONDITION
条件が起こった場合に処置を行わないことを指定します。

HANDLE ABEND
異常終了処理のための出口ルーチンの活動化、取り消し、または再活動化を行います。

異常終了は、CICS が例外条件を処理する場合の最も一般的な方法です。

IGNORE CONDITION、HANDLE ABEND、および HANDLE CONDITION の現在の効果は、PUSH HANDLE を使用すると中断し、POP HANDLE を使用すると復元します。

特定のラベルに制御権を渡すには、以下の 2 とおりの方法があります。

1. HANDLE CONDITION condition(label) コマンドを使用する。ここで、condition は例外条件の名前です。
2. HANDLE CONDITION ERROR(label) コマンドを使用する。

HANDLE CONDITION コマンドは、対象の条件に名前を付けて CICS コードをセットアップしてから、その条件が起こった場合に、このコードを使用して、制御をユーザー・アプリケーションの適切なセクションに渡します。アクティブの HANDLE CONDITION コマンドを使用するので、制御は常にユーザーがその特定の条件に指定したラベルに適用されます。

場合によっては、多くの異なるコマンドでさまざまな理由により、同じ条件が起こることがあります。例えば、ファイル制御操作、インターバル制御操作、およびその他の操作の途中で IOERR 条件が起こることがあります。したがって、最初の作業の 1 つは、特定の条件を生じさせたコマンドを見極めることです。そのコマンドを見つけ出してはじめて、その条件が起こった原因の調査を始めることができます。多くのプログラマーにとって、これは新規 CICS アプリケーションで RESP オプションの使用を開始するのに十分な理由になります。複数の条件にエラー処理を設定するために HANDLE CONDITION コマンドは 1 つしか必要ありませんが、CICS コマンドがユーザー・コード内で失敗した場合に、複数の HANDLE CONDITION コマンドのうちのどのコマンドが現在アクティブになっているのかを正確に見つけ出すのが難しい場合もあります。

名前を指定していない条件が起こった場合には、CICS はデフォルトのアクションがタスクを異常終了させるようになっていない限り、デフォルトのアクションを実行します。この場合には、ERROR 条件が引き起こされます。条件の名前を指定していても、そのラベルを指定していない場合には、その条件に対して HANDLE CONDITION コマンドはまったく使用できなくなり、この条件が起こった場合、CICS は、元どおりデフォルトのアクションを実行することになります。

すべての条件を処理しそこなうというのが、HANDLE CONDITION コマンドを使用する際のエラーに関する共通の原因です。 使い慣れないコマンドを使用する場合には、「*アプリケーション・プログラミング・リファレンス*」のコマンド説明を参照して、起こる可能性がある例外条件を知っておく必要があります。その後で、起こりうる例外条件のすべてに HANDLE コマンドを発行した場合でも、すべてのエラー処理コードを完全に処理することはできません。RETURN コマンドを発行することによって、エラー処理ルーチンが、不完全なデータ変更または誤ったデータ変更をコミットする結果になる場合があります。

最良の方法は、HANDLE CONDITION コマンドを使用することですが、特定の問題に関する明確な方法が見つからない場合には、システム・デフォルトのアクションに引き継ぐことをお勧めします。

単に待機の原因となる条件 (待機の原因となる条件の例については、391 ページの『CICS が実行する内容を見失わない方法』を参照してください) および SEND MAP コマンドのオーバーフロー処理の特殊ケースのエラー条件の相違点を念頭におき、ユーザー・アプリケーション内で HANDLE CONDITION condition(label) コマンドまたは HANDLE CONDITION ERROR(label) コマンドを実行した後で、HANDLE CONDITION コマンドをアクティブにします。

HANDLE CONDITION コマンドが条件に対してアクティブになっていなくても、ERROR に対してアクティブになっている場合は、条件が待機ではなくエラーであれば、制御は ERROR 処理を行うラベルに渡ります。

HANDLE CONDITION コマンドを使用しているか、あるいはそのコマンドを使用するアプリケーションを保守している場合には、ループの原因になるので、ルーチンの元の分岐を提供したのと同じ条件を起こすことがあるどのコマンドもエラー・ルーチンには組み込まないでください。

ERROR 条件そのものでループを起こすことがないように、特に注意してください。 ERROR 条件が起こった場合のループは、システム・デフォルトのアクションを一時的に復帰させることによって回避することができます。これは、ラベルを指定しないで HANDLE CONDITION ERROR コマンドをコーディングすることにより行います。エラー処理ルーチンの終わりに、適切なラベルを指定した HANDLE CONDITION ERROR コマンドを組み込むことによってエラー処置を復元させることができます。直前の HANDLE CONDITION 状態がわかっている場合には、これを明示的に行うことができます。ユーザー・コードのいくつかの異なる地点から呼び出される可能性がある一般的なサブルーチンでは、PUSH HANDLE コマンドおよび POP HANDLE コマンドが役立つ可能性があります。 394 ページの『PUSH HANDLE および POP HANDLE コマンドの使用』を参照してください。

HANDLE CONDITION コマンドの使用

このタスクについて

HANDLE CONDITION コマンドを使用して、ある条件が起こった場合に制御を渡すラベルを指定します。このとき、条件の名前を組み込む必要があり、関連した条件を引き起こす可能性があるコマンドの前に、必ず、HANDLE CONDITION コマンドを実行しなければなりません。

同一のコマンドに 16 を超える条件を組み込むことはできません。それを超える条件は、追加の HANDLE CONDITION コマンドに指定しなければなりません。また、同一リストに ERROR 条件を使用して、他のすべての条件によって、制御を同一ラベルに渡すことを指定することができます。

所定の条件に関する HANDLE CONDITION コマンドは、コマンドを指定したプログラムにのみ適用されます。HANDLE CONDITION コマンドは、

- プログラムが実行されている間、または以下の時点まで、アクティブ状態のままである。
  - 同じ条件についての IGNORE CONDITION コマンドが見付かったとき。この場合には、HANDLE CONDITION コマンドは指定変更されます。
  - 同じ条件について別の HANDLE CONDITION コマンドが見つかったとき。この場合には、新しいコマンドによって前のコマンドの指定が変更されます。
- コマンドに NOHANDLE オプションまたは RESP オプションの指定があると、一時的に非活動化される。

LINK コマンドまたは XCTL コマンドによって別のプログラムに制御が渡ると、呼び出し側のプログラムでアクティブであった HANDLE CONDITION コマンドは非活動化されます。あるプログラムに対して、それより論理レベルの低いプログラムから制御が返される場合には、制御が移動する前に、レベルの高い方のプログラムでアクティブであった HANDLE CONDITION コマンドが再びアクティブ化され、下位プログラムの HANDLE CONDITION コマンドは非活動化されます (論理レベルについては、326 ページの『アプリケーション・プログラムの論理レベル』を参照してください)。

以下の例は、DUPREC、LENGERR など、データ・セットにレコードを追加するため WRITE コマンドを使用するときに起こる状態の処理方法を示しています。DUPREC を特殊ケースとして取り扱い、LENGERR については標準システム処置 (すなわち、タスクを異常終了する) を行い、さらにその他の条件をエラー・ルーチン ERRHANDL によって取り扱うものとします。以下のようにコーディングします。

```
EXEC CICS HANDLE CONDITION
          ERROR(ERRHANDL)
          DUPREC(DUPRTN) LENGERR
END-EXEC.
```

PL/I アプリケーション・プログラムでは、条件が発生して、非アクティブのプロシージャーまたは非アクティブの開始ブロック内のラベルに分岐すると、予測不可能な結果になります。

アセンブラー言語アプリケーション・プログラムでは、ラベルへの分岐が条件により引き起こされた場合は、アプリケーション・プログラムのレジスターは、その条件の原因となったコマンドが実行された地点でのプログラム内の値に復元されます。

RESP および NOHANDLE オプション

コマンドで RESP オプションまたは NOHANDLE オプションを使用して、任意の HANDLE CONDITION コマンドの効果を一時的に非活動化することができます。こ

れらのオプションの使用方法については、 384 ページの『インライン・コードによる例外条件の処理』に説明があります。このオプションを指定した場合には、そのコマンドのすべてのシステム・デフォルトのアクションを使用する機能は失われます。すなわち、ユーザー独自の「キャッチ・オール」エラー処理を行う必要があります。

CICS が実行する内容を見失わない方法

CICS は、ユーザー・アプリケーションの HANDLE CONDITION コマンドおよび IGNORE CONDITION コマンドによって参照される条件のテーブルを持っています。これらのコマンドの 1 つを実行すると、このテーブル内の既存の項目が更新されるか、あるいは条件がこれらのコマンドで最初に引用される場合には、CICS が新規項目を作成します。各項目は、次の 3 種類の例外処理状態のうち、ユーザー・アプリケーション・プログラムが取り得る状態を 1 つ示すことによって、CICS に何をすべきかを伝えます。

1. CICS から、プログラム内で失敗したコマンドに続く次の命令に返される制御から**プログラムを続行させる**。その後で、テストで起こった内容、例えば、コマンドの実行後に CICS が戻す RESP 値などを見極めることができます。このテストの結果によって、次に実行する内容を決定することができます。詳細については、 384 ページの『インライン・コードによる例外条件の処理』を参照してください。

   この推奨方法は、「*Sample Applications Guide*」に記載されている「File A」サンプル・プログラム、および「*Designing and Programming CICS Applications*」の COBOL のサンプル・アプリケーションで使用されているアプローチです。また、これはすべての新規 CICS アプリケーションに関して推奨するアプローチでもあります。これは、プログラムを構造化コードにして、過去の CICS に必要だった暗黙の GOTO の必要性が取り除かれます。
2. 名前を指定した条件が起こった場合に**指定ラベルに制御を渡す**。これは、HANDLE CONDITION コマンドまたは HANDLE CONDITION ERROR コマンドを使用して、条件とルーチンのラベルの両方の名前を、それを取り扱うコードの中に指定することによって行います。詳細については、 389 ページの『HANDLE CONDITION コマンドの使用』および 392 ページの『HANDLE CONDITION ERROR コマンドの使用』を参照してください。
3. **CICS システム・デフォルトのアクションを行う**。ほとんどの条件の場合、これはタスクを異常終了することで、条件のテストまたは処理という手段では何も行われないことを意味します。

条件 ENQBUSY、NOJBUFSP、NOSTG、QBUSY、SESSBUSY、および SYSBUSY の場合、通常のデフォルトのアクションでは、必要なリソース (例えば、ストレージ) が利用可能になるまでタスクを強制的に待機させてから、コマンドの処理を再開します。この動作は、NOSUSPEND オプションを使用して、条件を無視するように変更することができます。NOSPACE 条件の場合、通常のデフォルトのアクションでは、WRITEQ TS コマンドを処理中の場合は待機しますが、WRITEQ TD、WRITE、または REWRITE コマンドを処理中の場合はタスクを異常終了します。NOSUSPEND オプションを指定した WRITEQ TS コマンドをコーディングすると、発生するすべての NOSPACE 条件が無視されます。詳しくは、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

CICS は、各リンク・レベルについて、これらの条件のテーブルを保持します。したがって、基本的に、各プログラム・レベルは、独自の条件処理を管理する独自の HANDLE 状態テーブルをもっています。

この動作を変更するには、HANDLE CONDITION ERROR と IGNORE CONDITION を使用します。

HANDLE CONDITION ERROR コマンドの使用

このタスクについて

図 109 は、プログラム ACCT01 で使用される 2 つしかない HANDLE CONDITION コマンドのうちの最初の方を示しています。

```
 PROCEDURE DIVISION.
*
*    INITIALIZE.
*    TRAP ANY UNEXPECTED ERRORS.
     EXEC CICS HANDLE CONDITION
     ERROR(OTHER-ERRORS)
     END-EXEC.
*
```

図 109. HANDLE CONDITION ERROR コマンドによる予期しない条件のトラップ

NOHANDLE または RESP を指定していないコマンドに対して何らかの条件が起こった場合に、ACCT01 はラベル OTHER-ERRORS の段落に制御を渡します。

HANDLE CONDITION ERROR コマンドは、この COBOL プログラムの手続き部で実行される最初のコマンドです。この理由は、処理する条件を引き起こす可能性があるすべての CICS コマンドを処理する前に HANDLE CONDITION コマンドを処理する必要があるためです。しかし、ユーザー・プログラムが HANDLE CONDITION コマンドを処理する場合には、ユーザー・プログラムにはその効果はわかりません。その効果がわかるのは、後に、名前を指定した条件の 1 つを引き起こす CICS コマンドをユーザー・プログラムが発行したときです。

このプログラムおよび他の ACCT プログラムでは、一般に、RESP オプションを使用します。RESP オプションを指定しているすべてのコマンドには、特定条件に関する明示的なテストの**後** に、「キャッチ・オール」テスト (IF RESPONSE NOT = DFHRESP(NORMAL) GO TO OTHER-ERRORS) がコーディングされています。そのため、特に「期待」している以外の例外が起きると、各プログラムの OTHER-ERRORS の段落に制御が渡されます。この HANDLE CONDITION ERROR コマンドにより、コマンド上に RESP をもっていない比較的少数のコマンドによって、NORMAL 以外のどの条件になった場合にもまったく同じ場所に制御が渡されます。

IGNORE CONDITION コマンドの使用

このタスクについて

HANDLE CONDITION コマンドを使用して、特定条件の特定ラベルに制御を渡すようにすることができるのとまったく同様に、特定の条件が起こった場合に、プログラムを続行させることができます。これは、コマンドで潜在的に起こる可能性があ

る条件の 1 つ以上を無視するように IGNORE CONDITION コマンドをセットアップすることによって行います。IGNORE CONDITION コマンドは、条件が起こった場合に、なにも処置を行わないことを意味します。制御はこのコマンドの次の命令に戻り、戻りコードは EIB に設定されます。次の例は MAPFAIL 条件を無視します。

```
EXEC CICS IGNORE CONDITION MAPFAIL
END-EXEC.
```

単一の EXEC CICS コマンドは、処理されている間に、複数の条件を起こすことができます。例えば、無効なだけでなく、ファイル制御テーブルで定義されていないファイルに適用されるファイル制御コマンドを使用することができます。CICS はこれらを検査し、(IGNORE CONDITION コマンドによって) 無視されない最初の条件をアプリケーション・プログラムに返します。CICS はアプリケーション・プログラムに一度に 1 つの例外条件しか返しません。

指定条件についての IGNORE CONDITION コマンドは、コマンドが指定されているプログラムのみに適用され、そのプログラムが実行されている間、あるいは後で同じ条件名を指定した別の HANDLE CONDITION コマンドが見付かるまでは、アクティブ状態のままです。同じ条件名を指定した別の HANDLE CONDITION コマンドが見付かると、IGNORE CONDITION コマンドは指定変更されます。

プログラムが用意したスペースより長くなることがあるレコードを読み取っても、これをエラーとは見なさず、それに対して処置を行わない場合には、IGNORE CONDITION コマンドを選択することができます。したがって、READ コマンドを発行する前に IGNORE CONDITION LENGERR をコーディングすることができます。

また、IGNORE CONDITION ERROR コマンドを使用すると、ラベルを含む HANDLE CONDITION コマンドで現在アクティブになっているものがないという理由で、エラーと見なされる条件すべてをキャッチすることもできます。エラーが起こった場合には、制御は次のステートメントに渡され、EIB の戻りコードは、そのプログラムの責任において検査されます。エラーと**見なさない**条件の例については 391 ページの『CICS が実行する内容を見失わない方法』を参照してください。

また、条件を無視する状態から条件を処理する状態、あるいはシステム・デフォルトのアクションを使用する状態に切り替えることもできます。例えば、次のようにコーディングすることができます。

```
*    MIXED ERROR PROCESSING
     EXEC CICS IGNORE CONDITION LENGERR
     END-EXEC.

     EXEC CICS HANDLE CONDITION DUPREC(DUPRTN)
          LENGERR
          ERROR(ERRHANDL)
          END-EXEC.
```

このコードは、最初、条件 LENGERR を無視するので、プログラムで LENGERR 条件が起こった場合には、何も起こりません。アプリケーションはその処理を続行するだけです。もちろん、LENGERR が発生したという事実が、明らかにアプリケーションが続行できないことを意味する場合には、問題が起こります。

コードの後方で、ラベルのない HANDLE CONDITION コマンドに LENGERR 条件を指定することによって、LENGERR 条件をシステム・デフォルトのアクションとして明示的に設定することができます。このコマンドを実行した場合、プログラムは LENGERR 条件を無視することはなくなり、次にこの条件が起こった場合には、システム・デフォルトのアクションが行われることになります。方式の混合についての要点は、それが可能で、各条件は別個に取り扱われるということです。

同一のコマンドに 16 を超える条件をコーディングすることはできません。追加の条件はその後の IGNORE CONDITION コマンドで指定しなければなりません。

HANDLE ABEND コマンドの使用

Java、C および C++ プログラマーへの注意事項

HANDLE ABEND は Java プログラムには適用されません。HANDLE ABEND は、PROGRAM オプションを指定して使用する場合には C および C++ でサポートされますが、C および C++ プログラムでの例外条件は異常終了の原因にはならないので、このセクションの内容は関連しません。

HANDLE ABEND コマンドは、アプリケーション・プログラムのプログラム・レベル異常終了出口の活動化または再活動化を行います。このコマンドを使用して、既に活動化されている出口を取り消すこともできます。

CICS では、DFHEIENT および DFHEIRET を使用しないアセンブラー・プログラムで HANDLE ABEND LABEL を使用することはできません。Language Environment のスタブ CEESTART を使用するアセンブラー・プログラムは、HANDLE ABEND PROGRAM、または CEEHDLR のような Language Environment を使用する必要があります。

HANDLE ABEND により、異常終了の処理にユーザー独自のコードが実行されます。これにより、ユーザー・アプリケーションは異常状態を順序正しく処理し、実行を続けることができます。ユーザーはユーザー出口プログラムを用意し、必要な場合に CICS によりそれを呼び出します。

異常終了処理時の制御のフローが、401 ページの図 110 に示されています。

PUSH HANDLE および POP HANDLE コマンドの使用

PUSH HANDLE
HANDLE CONDITION、IGNORE CONDITION、HANDLE ABEND および HANDLE AID コマンドの現在の効果を中断します。

POP HANDLE
HANDLE CONDITION、IGNORE CONDITION、HANDLE ABEND および HANDLE AID コマンドの効果を、直前の PUSH HANDLE コマンドが呼び出される前の状態に復帰させます。

また、CICS は、対応する POP HANDLE コマンドが見つかっていない各 PUSH HANDLE コマンドに関する条件のテーブルも保持します。

それぞれの条件が起こった場合に、CICS は次の一連のテストを実行します。

1. コマンドに RESP オプションまたは NOHANDLE オプションが指定されている場合には、制御はアプリケーション・プログラムの次の命令に返されます。そうでない場合には、CICS は条件テーブルをスキャンして、実行内容を見つけます。
2. 条件についての項目が存在する場合には、その項目によって処置が決まります。
3. 項目が存在せず、その状態に関するデフォルトのアクションが実行の中断である場合は、次のように処理されます。
   a. コマンドに NOSUSPEND オプションまたは NOQUEUE オプションが指定されている場合には、制御は次の命令に返される。
   b. コマンドにこれらのオプションのいずれも指定されていない場合には、タスクが中断される。
4. この条件についての項目が存在せず、デフォルトのアクションが異常終了する場合、二度目の探索が ERROR 条件について行われ、次のように処理されます。
   a. ERROR が見つかった場合は、この項目によって処置が判別される。
   b. ERROR が見つからない場合は、タスクが異常終了する。異常終了を処理することを選択することができます。

注: SEND MAP コマンドでの OVERFLOW 条件は、上記の規則に対する例外です。

コマンド ALLOCATE、ENQ、GETMAIN、WRITE JOURNALNAME、WRITE JOURNALNUM、READQ TD、および WRITEQ TS はすべて、デフォルトのアクションによって、指定のリソースが利用可能になるまでアプリケーション・プログラムが中断される条件を起こす可能性があります。したがって、これらのコマンドには、このような待機を禁止し、アプリケーション・プログラム内の次の命令に即時に戻るための NOSUSPEND オプションがあります。

関連しない複数のコマンドの実行中に、いくつかの条件が起こることがあります。発生するすべての条件に同じ処置が必要な場合には、プログラムの始めに、単一の HANDLE CONDITION コマンドをコーディングします。

注: RESP の使用が NOHANDLE を暗黙指定するときには、RESP を RECEIVE コマンドに使用すると、HANDLE CONDITION コマンドの他に HANDLE AID コマンドも指定変更されるので注意してください。これは、PF キー応答が無視されることを意味し、この理由により、前もって ACCT コードでテストしなければなりません。 824 ページの『HANDLE AID コマンドの使用』を参照してください。

第 25 章 異常終了のリカバリー

CICS では、タスクの異常終了時に制御を受け取ることができる独自の出口ルーチンを作成できるように、プログラム・レベルの異常終了出口機能が提供されます。このような異常終了出口によって実行される機能の例として、開始されたのに正常に完了していないプログラムの「終結処理」があげられます。

異常終了になる原因には、以下のものがあります。

- 例えば、以下のようなユーザー要求。

  ```
  EXEC CICS ABEND ABCODE(...)
  ```

- 無効なユーザー要求の結果としての CICS 要求。例えば、無効な FREEMAIN 要求でトランザクション異常終了コードの ASCF が返される場合。
- プログラム・チェック。この場合は、システム・リカバリー・プログラム (DFHSRP) が駆動され、タスクは ASRA コードを出して異常終了します。
- オペレーティング・システム異常終了。この場合は、DFHSRP が駆動され、タスクは ASRB コードを出して異常終了します。
- タスクのループ。この場合は、DFHSRP が駆動され、タスクは AICA コードを出して異常終了します。

注: CICS コードで ASRB または ASRA が検出されると、CICS は、HANDLE ABEND 出口を呼び出す前にダンプを生成します。

問題の修正について詳しくは「*CICS Problem Determination Guide*」を参照し、CICS によって開始される、異常終了に対するトランザクション異常終了コード、その意味、およびユーザーの対応について詳しくは、「*CICS Messages and Codes*」を参照してください。

HANDLE ABEND コマンドは、アプリケーション・プログラムのプログラム・レベル異常終了出口の活動化または再活動化を行います。このコマンドを使用して、既に活動化されている出口を取り消すこともできます。

出口ルーチンを活動化する場合は、PROGRAM オプションを使用して制御を受け取るプログラムの名前を指定するか、または (C、C++、および PL/I プログラムを除いて) LABEL オプションを使用し、異常終了条件が発生した場合の制御の分岐先にするルーチン・ラベルを指定する必要があります。PL/I での ON ERROR ブロックの使用は、HANDLE ABEND LABEL コマンドの使用と同等です。

HANDLE ABEND コマンドは、同一論理レベルにあるすべてのアプリケーション・プログラム内で先行するこのようなコマンドのすべてを指定変更します。トランザクションの各アプリケーション・プログラムは独自の異常終了出口を持つことができますが、各論理レベルでは異常終了出口を 1 つしかアクティブにすることができません (論理レベルの説明は、325 ページの『第 21 章 プログラム制御』にあります)。

タスクが異常終了すると、CICS は異常終了が発生したアプリケーション・プログラムの論理レベルから始め、次々と高いレベルへ進みながら、アクティブの異常終了

© Copyright IBM Corp. 1989, 2009

出口を検索します。最初に見つかったアクティブな異常終了出口 (ある場合) に、制御が渡されます。この手順は、 401 ページの図 110 に示されています。また、ここには、ユーザー作成の異常終了出口が後続の異常終了処理を判別する方法も示されています。

異常終了出口が見つからない場合には、CICS は制御を異常終了条件プログラムに渡して、タスクを異常終了させます。このプログラムは、ユーザー置換可能プログラム・エラー・プログラム、DFHPEP を呼び出します。DFHPEP のカスタマイズ方法に関するプログラミング情報については、「*CICS Customization Guide*」のプログラム・エラー・プログラムの作成を参照してください。

異常終了出口での再帰的な異常終了を避けるために、CICS は出口ルーチンまたは出口プログラムに入る時にその出口ルーチンを非活動化します。操作を再試行したい場合は、異常終了時に制御中だったプログラム内の地点へ分岐し、HANDLE ABEND RESET コマンドを発行して異常終了出口を再活動化することができます。また、このコマンドを使用して、HANDLE ABEND CANCEL コマンドによって一度取り消された (発行プログラムの論理レベルにある) 異常終了出口を再活動化することもできます。 394 ページの『PUSH HANDLE および POP HANDLE コマンドの使用』 に説明されているように、PUSH HANDLE コマンドおよび POP HANDLE コマンドを使用する方法により、HANDLE ABEND コマンドを中断することができます。

異常終了を処理する場合には、動的トランザクション・バックアウト・プログラムは呼び出されないことに注意してください。動的トランザクション・バックアウト・プログラムが必要な場合には、暗黙または明示の同期点を取るか、SYNCPOINT ROLLBACK コマンドを発行するか、または ABEND コマンドを発行する必要があります。

異常終了が、IRC 接続されたシステムで実行されているトランザクションの障害で起きた (例えば、AZI2 など) 場合、同期点処理がバックアウト処理中に同じ IRC 接続を使用しようとすると、ASP1 コードが出てその同期点処理が異常終了することがあります。

HANDLE ABEND コマンドは、異常終了コードの ASPx または APSJ を代行受信することはできません。

このセクションでは、以下について説明します。

- 『プログラム・レベルの異常終了出口の作成』
- 400 ページの『操作の再試行』
- 401 ページの『トレース』
- 403 ページの『アプリケーション・パフォーマンスのモニター』
- 403 ページの『ダンプ』

プログラム・レベルの異常終了出口の作成

このタスクについて

RDO またはプログラム自動インストール出口のいずれかを使用することにより、異常終了出口を定義することができます。自動インストールの方法を使用すると、こ

のプログラム定義は、HANDLE ABEND の発行時に使用できません。つまり、プログラムは、最初に呼び出されたときには違う動作をすることがあるということです。 HANDLE ABEND の発行時にプログラムが未定義であり、プログラム自動インストールがアクティブでる場合は、プログラム名に関するセキュリティー・チェックのみが実行されます。他のチェックは、異常終了プログラムが呼び出されたときに行われます。自動インストールに失敗すると、タスクは APCT で異常終了し、制御は 1 つ上のレベルに渡されます。

異常終了出口プログラムは、サポートされる言語であればどの言語でコーディングしてもかまいませんが、異常終了出口ルーチンはプログラムと同じ言語でコーディングしなければなりません。

異常終了出口ルーチンの場合、アドレッシング・モードおよび実行キーは、HANDLE ABEND コマンドが発行されたときのアドレッシング・モードおよび実行キーに設定されます。

異常終了出口プログラムへの入り口では、その言語でコーディングされたすべてのアプリケーション・プログラムの場合に通常想定されている以外のアドレス可能度を想定することはできません。 C、C++、および PL/I の各言語では HANDLE ABEND ラベルをサポートしていないため、これらの言語用のレジスター値はありません。

異常終了出口ルーチンに入るときのレジスター値は、以下のとおりです。

COBOL

レジスターが復元されて HANDLE ABEND コマンドに制御が戻り、次に COBOL GOTO が実行されます。

アセンブラー

レジスター 15

異常終了ラベル。

レジスター 0-14

最後の CICS サービス要求時点の内容。

異常終了出口ルーチンまたはプログラムの処理を終了させるには、以下の 3 とおりの方法があります。異常終了ルーチンおよびプログラムが CICS 内部論理によって呼び出されたときは、異常終了の形で終了することをお勧めします。処理を続行することにより、さらに問題が拡大することがあるためです。

1. RETURN コマンドを使用して、1 つ上の論理レベルのプログラムに制御を渡し、タスクの実行を続行することを指示します。1 つ上の論理レベルのプログラムが存在しなければ、タスクは正常終了し、任意のリカバリー可能リソースがコミットされます。
2. ABEND コマンドを使用して、論理レベルがより高いプログラムに指定されている異常終了出口に、あるいはそれが存在しない場合には、異常終了処理に関する異常条件プログラムに制御を渡して、タスクを異常終了することを指示します。
3. 操作を再試行するための分岐。操作を再試行するこの方法を使用しており、二度目の障害が発生した後で元の異常終了出口ルーチンまたはプログラムに再入したい場合には、この異常終了出口ルーチンまたはプログラムは、分岐する前に

HANDLE ABEND RESET コマンドを発行する必要があります。これは、CICS が、異常終了出口に再入できないように出口ルーチンまたはプログラムを使用不能にしているためです。

未解決の RECEIVE コマンドがタイムアウトになったために起こった異常終了の場合、CICS が RECEIVE を取り消せるように CICS 異常終了を継続させることが重要です。

操作の再試行

CICS サービスの呼び出し時に異常終了が起きた場合に、**同じサービス**をさらに要求すると、予期しない結果となる場合があるので注意してください。これは、出口ルーチンで、ポインターや作業域の再初期設定、およびストレージ域の解放が完了していないことがあるためです。

さらに入出力操作を試みて、ATNI 異常終了または ATND 異常終了をリカバリーしようとしてはいけません。これらの異常終了のいずれかは TERMERR 条件になり、すべての場合にセッションを終了する必要があります。AZCT 異常終了または AZIG 異常終了からのリカバリー中は、CICS が RTIMOUT の終結処理を完全に行っていないだけでなく、不明確な待機が発生することがあるので、端末制御コマンドを発行しないようにしてください。

システム間通信を使用している場合、リモート・システムで異常終了が起こると、指定されたプログラムまたはラベルへ分岐する場合があり、リモート・システムの同じリソースをそれ以降に使用しようとする要求は失敗する可能性があります。リモート・システムへの接続の失敗が原因で異常終了が発生した場合は、それ以降にリモート・システムの任意の リソースを使用する要求を出すと、失敗する可能性があります。

BMS コマンドの結果として異常終了が起こった場合には、制御が BMS プログラムに返される前に制御ブロックは拘束されていないので、コマンドを再試行した場合には、結果が予測できなくなります。

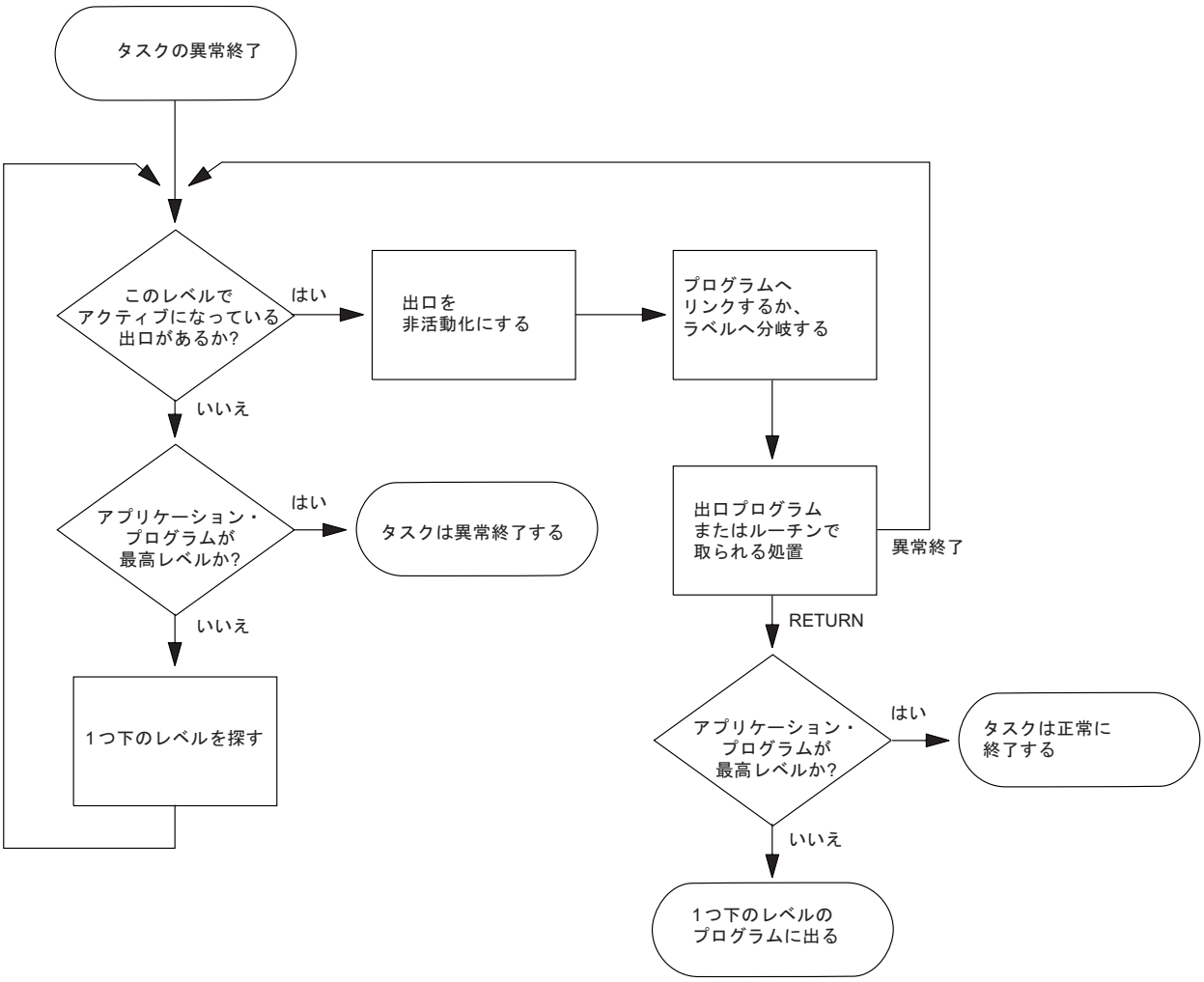


図 110. ABEND 出口の処理

トレース

CICS トレースは、アプリケーション・プログラマー、システム・プログラマー、および IBM 技術員のためのデバッグ援助機能です。CICS トレースは、トレース・コマンドに応じてトレース項目を生成します。トレース項目は、現在アクティブになっている任意のトレースの宛先に送ることができます。宛先は以下のとおりです。

- 内部トレース・テーブル
- 補助トレース・データ・セット
- 汎用トレース機能 (GTF) データ・セット

トレースの宛先について詳しくは、「*CICS Problem Determination Guide*」のトレース宛先および関連オプションの選択を参照してください。

以下のことを行うことができます。

- ユーザー・トレースの入り口点を指定する (ENTER TRACENUM)。

- CICS 内部トレースのオン / オフの切り替えは、SET TRACEDEST、SET TRACEFLAG、および SET TRACETYPE コマンドを使用して行います。

トレースの入り口点

CICS の操作中にトレース項目が生成される地点には、システム・トレース入り口点、ユーザー・トレース入り口点、例外トレース入り口点、およびユーザー例外トレース入り口点の 4 タイプがあります。トレースの詳細については、「*CICS Problem Determination Guide*」の問題判別におけるトレースとダンプの使用を参照してください。

システム・トレースの入り口点

トレース制御要求がなされる CICS 内の地点です。EXEC インターフェース・プログラムのシステム・トレース入り口点が、アプリケーション・プログラマーにとって最も重要です。これらは、CICS コマンドが処理されるたびに、トレース・テーブルに項目を生成します。

2 つのトレース項目が作成されます。1 つはコマンドが発行された時点、もう 1 つは CICS が要求された機能を実行し終え、制御をアプリケーション・プログラムに戻そうとする時点です。この 2 つの時点の間で、これら 2 つのトレース項目によって、アプリケーションによる制御のフローをトレースし、アプリケーションの実行中に例外条件が起こった場合に、どの例外条件が起こったのかを検査することができます。ABEND、RETURN、TRACEFLAG、および XCTL コマンドで生成されるのは、単一の項目だけです。

ユーザー・トレースの入り口点

ユーザー・トレース入り口点は、プログラムを完全にデバッグすることができるように、トレース・テーブルに組み込むことができるアプリケーション・プログラム内の追加地点です。例えば、ループに入った回数を示すカウンター値が入るプログラム・ループ用の項目を指定することができます。

トレース項目は、ENTER TRACENUM コマンドが実行されるたびに生成されます。固有 ID を付与することができるトレース項目要求のたびに、データがトレース・テーブルに配置されます。

例外トレースの入り口点

CICS が例外条件を検出した追加地点です。CICS コードの特定地点から作成されたもので、データは、原因に関する情報を提供する可能性がある区域から入手されます。例外トレース入り口点には、関連する「レベル」属性がありません。トレースは、例外条件が発生した場合に、その入り口点からのみ呼び出されます。

ユーザー例外トレースの入り口点

これらは、(内部トレースがオフに設定されている場合でも) 常に内部トレース・テーブルに書き込まれ、宛先がアクティブになっている場合だけ、他の宛先にも書き込まれるというトレース項目です。トレース項目は、CICS ユーティリティー・プログラムによって生成される定様式トレース出力で文字ストリング *EXCU によって識別することができます。ユーザー例外トレースの入り口点に関する一般情報については「*CICS Problem Determination Guide*」のユーザー例外トレースの入り口を参

照してください。プログラミング情報については「*CICS Customization Guide*」のトレース・エントリーの作成を参照してください。

アプリケーション・パフォーマンスのモニター

CICS モニターは、ユーザー・アプリケーション・トランザクションのパフォーマンスについての情報を提供します。MONITOR コマンドは、ユーザー・イベントのモニター・ポイントで使用します。

システム定義の任意の CICS のモニター点 (EMP) から収集されるモニター・データに加えて、ユーザー・アプリケーション・プログラムは、CICS モニター・レコードのユーザー・フィールドにデータを提供することができます。これは、MONITOR POINT コマンドを使用してユーザー定義の EMP を呼び出すことによって行うことができます。各 EMP において、各パフォーマンス・モニター・レコードごとに最大で 16384 バイトのユーザー独自データを追加または変更できます。これらの 16384 バイトでは、以下の任意の組み合わせを使用することができます。

- 範囲 0 ～ 256 のカウンター
- 範囲 0 ～ 256 のクロック
- 単一の 256 バイトの文字ストリング

例えば、これらのユーザー EMP を使用して、特定のイベントの発生回数のカウントや、2 つのイベントの間の時間間隔の計測が可能です。

ダンプ

CICS ダンプでは、DUMP TRANSACTION によって、主記憶装置の区域の内容を順次データ・セット (ディスクまたはテープのいずれも可能) 上にダンプするように指定できます。

PERFORM DUMP コマンドでシステム・ダンプを要求できます。このコマンドのプログラミング情報については、「*CICS System Programming Reference*」の PERFORM DUMPを参照してください。

ダンプ・データ・セットの内容は形式設定が可能です。また、トランザクション・ダンプの場合は CICS ダンプ・ユーティリティー・プログラム (DFHDU660) を使用し、システム・ダンプの場合は対話式問題制御システム (IPCS) を使用して、ダンプ・データ・セットの内容をオフラインで印刷できます。これらのプログラムの使用に関する説明については、「*CICS Operations and Utilities Guide*」のダンプ・ユーティリティー・プログラムを参照してください。

ダンプ管理コマンドは一度に 1 つしか処理されません。追加のダンプ管理コマンドを発行しても、別のタスクがトランザクション・ダンプを取得している場合、そのダンプが完了するまで、追加のコマンドと関連したタスク内のアクティビティーは延期されます。残りのダンプ・コマンドは発行した順に処理されます。 DUMP TRANSACTION コマンドを使用すると、EIB および TCA の一部のフィールド (例えば、EIBFN および EIBRCODE) が上書きされます。

DUMP TRANSACTION コマンドのオプションによって、主記憶装置の以下の区域をさまざまな組み合わせでダンプすることができます。

- タスク関連ストレージ域: 要求タスクに関係のある選択された主記憶装置。通常、これらの区域のダンプを使用して、ユーザー・アプリケーション・プログラムのテストおよびデバッグを行います (関連したタスクが異常終了した場合には、自動的に CICS がこのサービスを提供します)。
- CICS 管理テーブル
  - ファイル管理テーブル (FCT)
  - プログラム管理テーブル (PCT)
  - 処理プログラム・テーブル (PPT)
  - システム初期設定テーブル (SIT)
  - 端末管理テーブル (TCT)

  これらのテーブルのダンプは、テストの基本を設定しなければならないテストにおいて、通常、最初に取得するダンプです。それ以降は、通常、タスク関連ストレージ・タイプのダンプを取得することになります。
- タスクの実行中は、タスク関連ストレージ域および CICS 管理テーブルの両方のダンプを作成したほうがいい場合があります。通常は、1 つの CICS 管理テーブル・ダンプと多数のタスク関連ストレージ・ダンプを指定するほうが、それに見合う数の完全ダンプを指定するより効率的です。しかし、CICS 管理テーブルは基本的には静的区域なので、この機能を排他使用してはいけません。
- さらに、3 つのオプション SEGMENTLIST、LENGTHLIST、および NUMSEGMENTS を使用した DUMP TRANSACTION コマンドにより、一連のタスク関連ストレージ域を同時にダンプすることができます。

RELOAD(YES) 属性が定義されたプログラムの場合、プログラム・ストレージはダンプされません。

さらに、印刷されたダンプの最後に、アドレスによる索引が付いた CICS 中核モジュールおよびアクティブ状態の PPT プログラムのリストが出力されます。

第 26 章 QUERY SECURITY コマンド

QUERY SECURITY コマンドを使用して、端末ユーザーが外部セキュリティー・マネージャーで定義されているリソースに対するアクセス権を持っているかどうか判別できます。

QUERY SECURITY コマンドは、RACF® または同等の外部セキュリティー・マネージャー (ESM) で使用できます。

ここで言う端末ユーザーとは、QUERY SECURITY コマンドを含んでいるトランザクションを呼び出すユーザーのことです。

Java および C++

ここで説明するアプリケーション・プログラミング・インターフェースは、Java プログラムでは使用されない CICS API です。JCICS クラスを使用してセキュリティー管理サービスにアクセスする Java プログラムについて詳しくは、「*Java Applications in CICS*」のJCICS クラス・ライブラリーおよび JCICS Javadoc 文書を参照してください。CICS C++ クラスを使用した C++ プログラムについて詳しくは、「*CICS C++ OO Class Libraries*」を参照してください。

QUERY SECURITY コマンドに応答して、CICS は、端末ユーザーのセキュリティー権限に関する情報を返します。CICS は、外部セキュリティー・マネージャーからこの情報を得ます。ユーザーに許可されたアクセスによって、別の方法で処理を続けるようにアプリケーション・プログラムをコーディングすることができます。

CICS リソース・タイプ名によって、照会しているリソースのタイプを指定します。例えば、ファイルのアクセスに関するユーザー権限について照会する場合には、RESTYPE('FILE') を指定できます。タイプ内の特定のファイルを識別するためには、RESID パラメーターを指定します。

QUERY SECURITY コマンドの使用

QUERY SECURITY コマンドの代表的な用法は、メニュー内の特定のトランザクション・コードを表示する***前に***、ユーザーがそのトランザクションの使用を許可されているかどうかを調べることです。

レコード・レベルまたはフィールド・レベルでのセキュリティー保護

QUERY SECURITY のもう一つの用法は、ユーザーがレコード・レベルまたはフィールド・レベルのデータに対しアクセス制御できるようにすることです。例えば、CICS におけるファイル・リソースに対する通常のリソース・セキュリティー検査は、ファイル・レベルでのみ機能します。個別のレコード、あるいはレコード内のフィールドに対するアクセスを制御するためには、QUERY SECURITY を使用することができます。この用途の場合、セキュリティー管理者は、保護したいレコードまたはフィールドに対して、適切なアクセス権限とともに、リソース・プロファイ

© Copyright IBM Corp. 1989, 2009

ル名を定義しなければなりません。これらのプロファイルが定義されているのは、管理者が定義するユーザー・リソース・クラスにおいてであって、CICS リソース・クラスにおいてでは***ありません***。

これらのクラスおよびリソースを照会するために、QUERY SECURITY コマンドは、RESCLASS および RESID オプションを使用します (RESCLASS および RESTYPE は互いに排他的なオプションです)。QUERY SECURITY によって返された CVDA 値を使用して、レコードまたはフィールドにアクセスするかどうかを判別します。

CICS 定義のリソース ID

SPCOMMAND リソース・タイプの場合以外すべて、リソース ID はユーザー定義されます。しかし、SPCOMMAND タイプの場合、ID は CICS で決められています。「*CICS RACF Security Guide*」のRESID 値では、SPCOMMAND リソース・タイプで使用可能な RESID 値について説明しています。

SEC システム初期設定パラメーター

SEC システム初期設定パラメーターの設定が、QUERY SECURITY コマンドによって返される CVDA 値に影響します。**SEC** システム初期設定パラメーターについて詳しくは、*CICS RACF Security Guide*を参照してください。

プログラミングのヒント

- 端末ユーザーがアクセスできるリソースのリストを作成するために、トランザクションで QUERY SECURITY コマンドを使用して、リソース数を照会することができます。この手法を利用して、トランザクションのアクセスが許可されていないリソースに関する照会ごとに、リソース違反メッセージを最大 4 つまで生成することができます。これらのメッセージは、システム・コンソール、CSCS TD キュー、および SMF ログ・データ・セットに現れます。これらのメッセージを抑制したい場合には、QUERY SECURITY コマンドに NOLOG と指定します。
- トランザクションが、1 回の実行中に同じリソースを何度もアクセスする場合には、トランザクション・リソース定義において、そのトランザクションに RESSEC(NO) と定義することで、パフォーマンスを改良することができます。さらに、QUERY SECURITY コマンドを単体で発行し、返された CVDA 値に従ってリソースに対するアクセスを許可するように、トランザクションをコーディングすることができます。詳しくは、「*CICS RACF Security Guide*」のQUERY SECURITY コマンドを使用したセキュリティー検査を参照してください。

第 27 章 CICS の相互通信

このセクションでは、他の CICS システムと通信するアプリケーションを作成する場合の考慮事項の要点のみを説明します。詳しくは、「*CICS 相互通信ガイド*」を参照してください。

アプリケーション・プログラムは、次の 1 つ以上の機能を使用する CICS 相互通信環境で実行することができます。

トランザクション・ルーティング
ある CICS システムの端末が、別の CICS システムのトランザクションを実行できるようにします。 408 ページの『トランザクション・ルーティング』を参照してください。

機能シップ
ユーザーのアプリケーション・プログラムが、別の CICS システムのリソースにアクセスできるようにします。 409 ページの『機能シップ』を参照してください。

分散プログラム・リンク (DPL)
ある CICS 領域で実行されているアプリケーション・プログラムが、リモートの CICS 領域で実行されている別のアプリケーション・プログラムにリンクできるようにします。 409 ページの『分散プログラム・リンク (DPL)』を参照してください。

非同期処理
CICS トランザクションが、リモート・システムの別のトランザクションを開始し、オプションでそのトランザクションにデータを渡せるようにします。 423 ページの『非同期処理』を参照してください。

分散トランザクション処理 (DTP)
CICS トランザクションから、別のシステムで実行されているトランザクションと通信できるようにします。DTP に使用可能なインターフェースは、コマンド・レベル EXEC CICS と、共通プログラミング・インターフェース・コミュニケーション (CPI コミュニケーション) として知られる DTP 用 SAA インターフェースの 2 つがあります。 423 ページの『分散トランザクション処理 (DTP)』を参照してください。

共通プログラミング・インターフェース・コミュニケーション (CPI-C)
APPC 接続に DTP を提供し、複数システムのプラットフォームで使用できる API を定義します。 423 ページの『共通プログラミング・インターフェース・コミュニケーション (CPI コミュニケーション)』を参照してください。

外部 CICS インターフェース (EXCI)
MVS で実行している非 CICS プログラムが CICS システムにセッションを割り振り、それをオープンできるようにします。さらに、これらのセッションで DPL 要求を出せるようにします。 CICS Transaction Server for z/OS バージョン 4 リリース 1 の場合、CICS は外部 CICS インターフェースを

© Copyright IBM Corp. 1989, 2009

使用するアプリケーションで MVS リソース・リカバリー・サービス (RRS) をサポートします。 424 ページの『外部 CICS インターフェース (EXCI)』を参照してください。

本書では、CICS フロントエンド・プログラミング・インターフェース (FEPI) の相互通信については説明されていません。FEPI について詳しくは、「*CICS Front End Programming Interface User's Guide*」を参照してください。

設計上の考慮事項

アプリケーション・プログラムでこれらの機能を複数使用する場合は、個々の機能に関する設計上の考慮事項に留意しなければなりません。また、ユーザー・プログラムで分散トランザクション処理に複数のシステム間セッションを使用する場合は、各セッションをそのセッションのタイプの規則にしたがって制御しなければなりません。

プログラミング言語

一般に、COBOL、C、C++、PL/I、またはアセンブラー言語を使用し、CICS 相互通信機能を使用するアプリケーション・プログラムを作成できます。しかしながら、これには例外があります。EXEC CICS API を使用して APPC 非マップ式会話を保持する DTP アプリケーション・プログラムでは、C、C++、またはアセンブラー言語のみが使用可能です。

トランザクション・ルーティング

別の CICS システムが所有している端末から呼び出すことのできるトランザクション、またはトランザクション開始時に別の CICS システムが所有している端末を獲得できるトランザクションは、トランザクション・ルーティング環境において実行できなければなりません。

一般に、このようなトランザクションはローカル環境で使用するトランザクションとまったく同様に設計およびコーディングを行うことができます。しかし、基本マッピング・サポート (BMS)、疑似会話型トランザクション、およびユーザー・トランザクションを実行する端末に関連した制約事項がいくつかあります。トランザクションが使用するすべてのプログラム、テーブル、およびマップは、そのトランザクションを所有しているシステム上に 常駐している**必要** があります (プログラム、テーブル、およびマップは、必要なシステムの数だけ複写することができます)。

いくつかの CICS トランザクションは、例えば、CWA への共通アクセスまたは GETMAIN コマンドを使用して獲得される共用ストレージにより互いに関連しています。その場合は、システム・プログラマーはそれらのトランザクションを同一の CICS システムにルーティングしなければなりません。動的トランザクション・ルーティングの実行に悪影響を及ぼす、トランザクション間の類縁性を生じさせる可能性のある手法は (可能な限り) 避けてください。

これらのコマンドを発行するプログラムにおいて発生する可能性のある問題の識別を容易にするため、CICS Interdependency Analyzer を使用できます。このユーティリティーについて詳しくは「*CICS Interdependency Analyzer for z/OS User's Guide*

and Reference」を参照し、トランザクションの類縁性について詳しくは 339 ページの『第 22 章 類縁性』を参照してください。

トランザクションを処理する要求をある CICS システムから別のシステムに伝送する場合に、トランザクション ID をローカル名からリモート名に変換することができます。しかし、RETURN コマンドに指定されるトランザクション ID は、トランザクションをトランザクション所有システムから端末所有システムに伝送する場合には変換されません。

機能シップ

リモート・システムにあるリソースにアクセスするプログラムは、リソースがローカル・システムにある場合とほぼ同じ方法でコーディングします。次のものを使用することができます。

DL/I 呼び出し (EXEC DLI コマンド)
リモート CICS システムに関連するデータにアクセスする。

ファイル制御コマンド
リモート・システムのファイルにアクセスする。TOKEN キーワードを含む要求は、機能シップしないでください。

一時記憶域コマンド
リモート・システムの一時記憶キューにあるデータにアクセスする。

一時データ・コマンド
リモート・システムの一時データ・キューにアクセスする。

リモート・リソースを使用すると、3 つの追加の例外条件が起こることがあります。これらの条件が起こるのは、リモート・システムが利用不能の場合 (SYSIDERR)、要求が無効の場合 (ISCINVREQ)、あるいはミラー・トランザクションが異常終了した (ISC 接続の場合は ATNI で、MRO の場合は AZI6) 場合です。

分散プログラム・リンク (DPL)

分散プログラム・リンク機能は、CICS プログラム (クライアント・プログラム) がリモート CICS 領域にある別の CICS プログラム (サーバー・プログラム) を呼び出せるようにします。ユーザーのアプリケーションの設計で分散プログラム・リンクを使用する理由はいくつかあります。これらの一部は、次のとおりです。

- エンド・ユーザー・インターフェース (例えば、BMS 画面処理など) を、アプリケーションのビジネス・ロジック (データのアクセスや処理など) から分離して、アプリケーションの各部分をホストからワークステーションにより簡単に移送できるようにする
- プログラムを利用するリソースのなるべく近くで実行することで、パフォーマンスを向上させ、機能シップ要求を繰り返す必要を減らす
- 分散トランザクション処理 (DTP) アプリケーションを作成する代わりの (多くの場合)、単純な方法を提供する

アプリケーションがリンクするプログラムをリモート・プログラムとして指定するには、以下のようないくつかの方法があります。

1. LINK コマンドでリモート・システム名を指定する

2. インストールしたプログラム・リソース定義でリモート・システム名を指定する
3. 動的ルーティング・プログラムを使用して、リモート・システム名を指定する (インストールされたプログラム定義で DYNAMIC(YES) が指定されている、またはインストールされたプログラム定義がない場合)
4. XPCREQ グローバル・ユーザー出口にリモート・システム名を指定する

分散プログラム・リンクにおける基本的な流れについては、「*CICS 相互通信ガイド*」のCICS 分散プログラム・リンクに説明されています。図 111 で説明されている次の用語は、分散プログラム・リンクで使用されます。

クライアント領域
別の CICS 領域のプログラムへのリンクを発行するアプリケーション・プログラムを実行する CICS 領域。

サーバー領域
クライアント領域がリンク要求をシップする先の CICS 領域。

クライアント・プログラム
リモート・リンク要求を出すアプリケーション・プログラム。

サーバー・プログラム
リンク要求で指定されたアプリケーション・プログラムであり、サーバー領域で実行される。

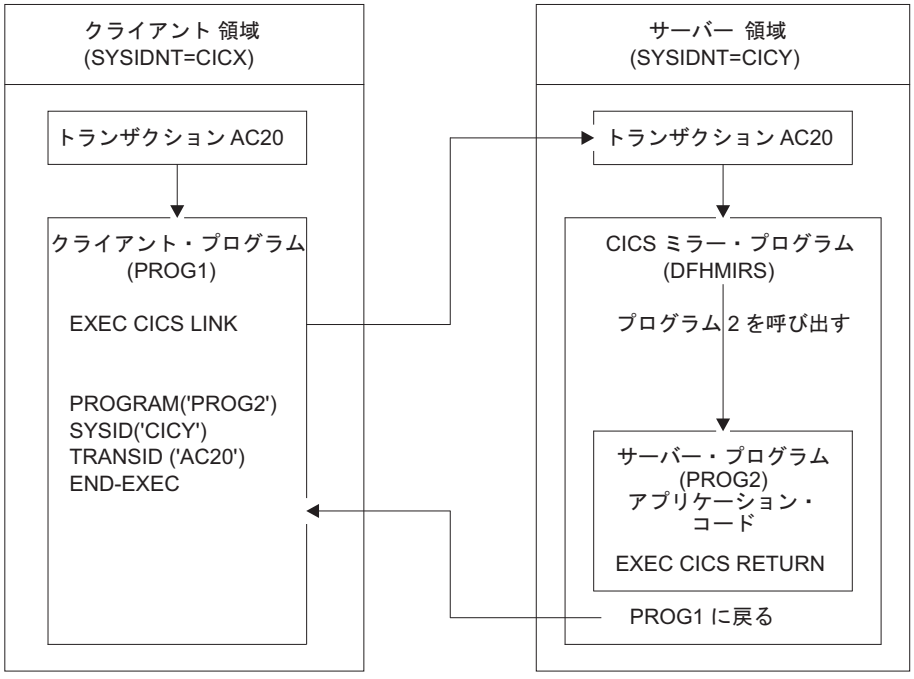


図 111. 分散プログラム・リンクの図

分散プログラム・リンク機能の使用

分散プログラム・リンク機能はオプションをいくつか提供します。次を指定することができます。

- リモート・システムの名前 (サーバー領域)。
- サーバー・プログラムの名前 (サーバー領域において別の名前で認識されている場合)。
- リンクされるプログラムをローカルで実行したいが、テストの目的で、アプリケーション・プログラミング・インターフェース (API) の分散プログラム・リンク・サブセットに制限すること (サーバー・プログラムをリモート側で実行する場合には、CICS API のすべてを使用することはできません。制約事項は、 421 ページの表 23 にリストされています)。
- サーバー・プログラムがクライアントとは独立して同期点を設定すること。
- サーバー領域のもとでプログラムを実行するトランザクションの名前。
- 受け渡しする COMMAREA のデータ長。

サーバー・プログラムそのものが分散プログラム・リンクを発行して、リンク先のプログラムとの関係でクライアント・プログラムとして動作することができます。

分散プログラム・リンク機能をサポートするために、LINK コマンドには表 21 に示したオプションが使用され、PROGRAM リソース定義には 412 ページの表 22 に示したオプションが使用されています。

表 21. DPL サポート用の LINK コマンドのオプション

キーワード	説明
DATALENGTH	アプリケーション・プログラムがサーバー・プログラムに伝送する、ストレージの連続区域 (COMMAREA の先頭から) の長さを指定する。
SYSID	クライアント領域がプログラム・リンク要求をシップする先の、サーバー領域への接続の名前を指定する。 **注:** LINK コマンドで指定されたリモート SYSID は、プログラム・リソース定義で指定された REMOTESYSTEM 名、または動的ルーティング・プログラムによって返されたシステム識別名を指定変更する。
SYNCONRETURN	サーバー・プログラムの正常終了時に、サーバー領域が同期点を取ることを指定する。 **注:** このオプションは LINK コマンド特有のもので、プログラム・リソース定義に指定することはできません。
TRANSID	サーバー領域がサーバー・プログラムの実行で接続するトランザクションの名前を指定する。 **注:** LINK コマンドで指定される TRANSID は、プログラム・リソース定義で指定されたどの TRANSID も指定変更します。

注: LINK コマンドの完全な構文などのプログラミング情報は、「*CICS アプリケーション・プログラミング・リファレンス*」に記載されています。ただし、分散プログラム・リンクの場合、INPUTMSG または INPUTMSGLEN オプションは指定できないので注意してください。

表 22. *DPL サポート用の PROGRAM リソース定義のオプション*

キーワード	説明
REMOTESYSTEM	クライアント領域がプログラム・リンク要求をシップする先の、サーバー領域 (SYSID) への接続の名前を指定する。
REMOTENAME	プログラムがサーバー領域において認識される名前 (ローカル名と異なる場合) を指定する。
DYNAMIC	プログラム・リンク要求を動的にルーティングできるかどうかを指定する。DPL 要求の動的ルーティングについては、*CICS 相互通信ガイド*を参照してください。
EXECUTIONSET	プログラムを、CICS API の分散プログラム・リンクのサブセットに限定するかどうかを指定する。 **注:** このオプションはプログラム定義特有のもので、LINK コマンドに指定することはできません。
TRANSID	サーバー領域がサーバー・プログラムの実行で接続するトランザクションの名前を指定する。

分散プログラム・リンクの例

分散プログラム・リンク・コマンドの COBOL の例は、図 112 に示されています。例の右側にある番号は、図の後にある番号の付いたセクションを指し、そこでは各オプションについての情報が提供されます。

重要

LINK コマンドの SYSID オプションがリモート・リージョンの名前を指定する場合、プログラム定義で指定された、あるいは動的ルーティング・プログラムによって返された属性 REMOTESYSTEM、REMOTENAME、または TRANSID は、すべて無効になります。

```
EXEC CICS LINK PROGRAM('DPLPROG')           1
               COMMAREA(DPLPRO-DATA-AREA)   2
               LENGTH(24000)                2
               DATALENGTH(100)              2
               SYSID('CICR')                3
               TRANSID('AC20')              4
               SYNCONRETURN                 5
```

図 112. 分散プログラム・リンクの COBOL の例

1. **サーバー・プログラムのプログラム名**

   プログラム名は、クライアント領域とサーバー領域で異なる場合があります。LINK コマンドで指定する名前は、SYSID オプションを指定するかどうかによって異なります。

   LINK コマンドの SYSID オプションでリモート領域の名前を指定する場合、CICS は、クライアント領域のプログラム・リソース定義の REMOTENAME 属

性や、動的ルーティング・プログラムによって返されたプログラム名を参照せずにサーバー領域にリンク要求をシップします。*この場合、LINK コマンドで指定する PROGRAM 名は、そのプログラムのサーバー領域で認識される名前にする必要があります。*

LINK コマンドで SYSID オプションを指定しない場合、またはローカルのクライアント領域の名前を指定する場合、*LINK コマンドで指定する PROGRAM 名は、そのプログラムのクライアント領域で認識される名前にする必要があります。*CICS は、クライアント領域内のプログラム・リソース定義を検索します。インストールされたプログラム定義の REMOTESYSTEM オプションがリモート・リージョンの名前を指定すると想定すると、リモート・リージョンのサーバー・プログラムの名前は、以下で指定されます。

a. プログラム定義の REMOTENAME 属性
b. REMOTENAME が指定されていない場合は、LINK コマンドの PROGRAM オプション。

プログラム定義で DYNAMIC(YES) が指定されている、またはインストールされたプログラム定義がない場合、動的ルーティング・プログラムが起動されて、サーバー・プログラムの名前の受け入れまたは変更ができるようになります。

2. **連絡データ域 (COMMAREA)**

パフォーマンスを改良するために、LINK コマンドに DATALENGTH オプションを指定することができます。これにより、クライアント領域でサーバー・プログラムに受け渡す COMMAREA データの容量を指定できます。この例は典型的なもので、このオプションを使用するのは、サーバー・プログラムがクライアント・プログラムに戻すデータを保持するために大きな COMMAREA が必要ですが、クライアント・プログラムがサーバー・プログラムに送る必要があるのは少量のデータだけという場合です。

複数のサーバー・プログラムが、同じ COMMAREA をクライアント・プログラムに渡される前に更新する場合は、DATALENGTH オプションを使用して、COMMAREA の長さを指定してください。サーバー・プログラムのいずれかが XCTL コマンドを使用して、次のサーバー・プログラムに COMMAREA を渡す場合は、COMMAREA には同じ長さとアドレスを指定するようにしてください。これによって、オリジナルの COMMAREA がクライアント・プログラムに返されるようになります。異なる長さまたはアドレスを指定すると、呼び出されるプログラムはオリジナルの COMMAREA ではなく COMMAREA のコピーを受け取るため、オリジナルの COMMAREA はクライアント・プログラムに返されません。COMMAREA を使用した別のプログラムへのデータの受け渡しについて詳しくは、327 ページの『COMMAREA』を参照してください。

3. **リモート・システム ID (SYSID)**

アプリケーション領域がプログラム・リンク要求をシップする先の、サーバー領域の 4 文字の名前は、以下のいずれかを使用して指定できます。

- LINK オプションの SYSID オプション
- プログラム・リソース定義の REMOTESYSTEM オプション
- 動的ルーティング・プログラム

優先順位の規則は、以下のとおりです。

a. EXEC CICS LINK コマンドの SYSID オプションでリモート CICS 領域が指定される場合は、CICS によってリモート・リージョンに要求が伝送される。

   プログラム定義で DYNAMIC(YES) が指定されている場合、またはプログラム定義がない場合は、通知のみの目的で動的ルーティング・プログラムが起動され、要求を転送することはできません。

b. SYSID オプションが指定されていない、またはローカル CICS 領域に同じ名前が指定されている場合は、以下のようになります。

   1) プログラム定義で DYNAMIC(YES) が指定されている場合、またはインストールされたプログラム定義がない場合は、動的ルーティング・プログラムが起動され、要求をルーティングできます。

      プログラム定義の REMOTESYSTEM オプションが指定されている場合は、そのオプションで動的ルーティング・プログラムに渡されるデフォルト・サーバー領域が指名されます。

      注: REMOTESYSTEM オプションでリモート・リージョンが指名されると、動的ルーティング・プログラムは要求をローカルでルーティングすることができません。

   2) プログラム定義で DYNAMIC(NO) が指定されると、CICS は、REMOTESYSTEM オプションで指名されたリモート・システムに要求を伝送します。REMOTESYSTEM が指定されていない場合、CICS はプログラムをローカルで実行します。

指定する名前は、クライアント領域にインストールされ、サーバー領域との接続を定義する接続定義の名前です。(CICS は、テーブル検索で接続名を使用し、サーバー領域のネット名 (VTAM APPLID) を取得します。) 指定するサーバー領域の名前はクライアント領域の名前にすることもできますが、この場合、プログラムはローカルで実行されます。

サーバー領域が要求されたプログラム (この例では DPLPROG) をロードまたは実行できない場合、CICS はリンク要求に対する応答でクライアント・プログラムに PGMIDERR 状態を返します。EIBRESP2 値は、サーバー領域でエラーが検出された分散プログラム・リンク要求に対して、リンクを通じては返されないことに注意してください。クライアント領域で検出されたエラーの場合には、EIBRESP2 値が返されます。

また、XPCREQ グローバル・ユーザー出口プログラムにおいてサーバー領域の名前を指定または変更することもできます。XPCREQ グローバル・ユーザー出口ポイントに関するプログラミング情報については、「*CICS Customization Guide*」の特定の呼び出しタイプの使用可能化を参照してください。

4. **接続されるリモート・トランザクション (TRANSID)**

   TRANSID オプションは、LINK コマンドとプログラム・リソース定義の両方で使用可能です。このオプションにより、サーバー・プログラムが実行されるミラー・タスクを添付する場合に使用するトランザクション ID をサーバー領域に通知できます。TRANSID オプションを指定する場合には、サーバー領域でトランザクションを定義し、それを提供されたミラー・プログラム DFHMIRS に関連

付けする必要があります。このオプションによって、パフォーマンスおよび最適調整のために、トランザクション定義でユーザー独自の属性を指定することができます。例えば、タスク優先順位およびトランザクション・クラス属性を変更することができます。

インストールされたプログラム定義で DYNAMIC(YES) が指定されている、またはインストールされたプログラム定義がない場合、動的ルーティング・プログラムが起動されて (ただし、LINK コマンドの SYSID オプションがリモート・リージョンを指名しなかったとき)、TRANSID 属性の値を変更することができます。

優先順位は、次のとおりです。

a. LINK コマンドの SYSID オプションがリモート・リージョンを指定した場合は、LINK に提供された TRANSID。
b. 動的ルーティング・プログラムで提供された TRANSID。
c. LINK コマンドで提供された TRANSID。
d. プログラム定義の TRANSID 属性。
e. ミラー TRANSID、CSMI。

クライアント・プログラムのトランザクション ID をサーバー・プログラムのトランザクション ID として指定することをお勧めします。これにより、収集するデータの統計とモニターを同一トランザクションのもとで正しく相関させることができます。

分散リンク・プログラム要求で使用するトランザクション ID は、次のようにサーバー・プログラムに渡されます。

- 分散リンク・プログラム要求にユーザー独自のトランザクション ID を指定すると、EIB の EIBTRNID フィールドでサーバー・プログラムに渡されます。
- EIBTRNID には、DPL API またはサーバーのリソース定義で指定されている TRANSID 値が設定されます。そうでない場合は、EIBTRNID はクライアントのトランザクション・コードのデフォルトになります。これは、クライアントの EIBTRNID に入っている値と同じです。

5. **サーバー・プログラムの SYNCONRETURN オプション**

SYNCONRETURN オプションを指定する場合には、クライアントに制御を返す直前に、サーバーのリソースを別々の作業論理単位ごとにコミットすることを意味します。すなわち、サーバーがクライアントに制御を返す直前に、サーバーに対して暗黙の同期点が発行されます。 416 ページの図 113 には、SYNCONRETURN オプションを使用した分散プログラム・リンクの使用例が示されています。SYNCONRETURN オプションは、クライアント・プログラムがいかなるリカバリー可能リソースも更新しない場合、例えば、画面処理を実行する場合などに使用するためのものです。しかし、クライアントにリカバリー可能リソースがない場合には、この時点ではコミットされません。これらがコミットされるのは、クライアントそのものが同期点に達するか、またはクライアント・タスクの終わりの暗黙の同期点においてです。クライアント・プログラムおよびサーバー・プログラムがこの目的のために正しく設計されているかどうか、およびデータ保全性のリスクを犯していないかどうかを確認してください。例えば、クライアント・プログラムがサーバーにデータをシップし、その結果、サーバー

領域により所有されているデータベースをサーバーが更新する場合は、独立した同期点のみの指定が安全であり、クライアント・プログラムで発生する内容に依存していなければ、独立した同期点のみを指定します。EXECUTIONSET (DPLSUBSET) が指定されていない限り、サーバー・プログラムがクライアント領域でローカルに実行されている場合には、このオプションは影響を及ぼしません。この場合、ローカル・リンク用の同期点の規則が適用されます。

SYNCONRETURN オプションを使用しないと、クライアントは、タスクの終わりで明示コマンドまたは暗黙の同期点のいずれか一方によって、クライアント・リソースとサーバー・リソースの両方の論理作業単位をコミットします。つまり、この場合には、サーバー・リソースはクライアント・リソースのコミットと同時にコミットされます。 417 ページの図 114 には、SYNCONRETURN オプションを使用しない分散プログラム・リンクの使用例が示されています。

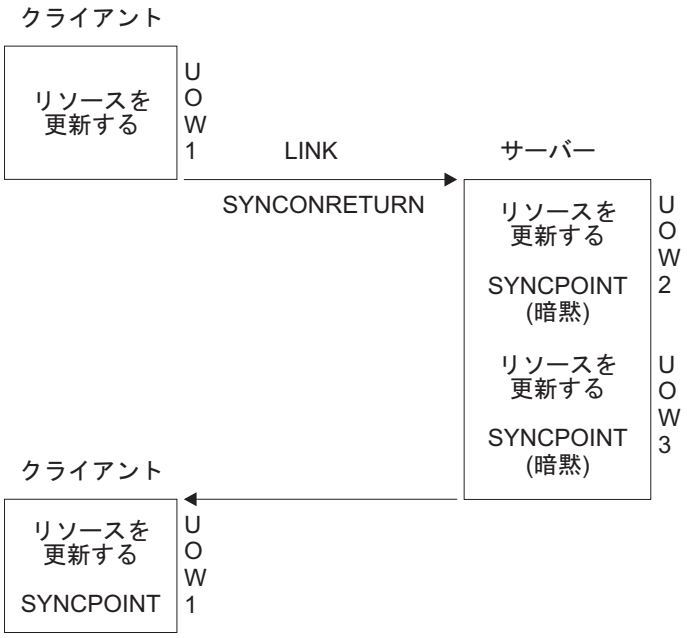


図 113. SYNCONRETURN オプションを使った分散プログラム・リンクの使用例

注: これには、3 つの論理作業単位 (1 つはクライアント用で、2 つはサーバー用) が含まれています。クライアント・リソースはサーバーとは独立してコミットされます。

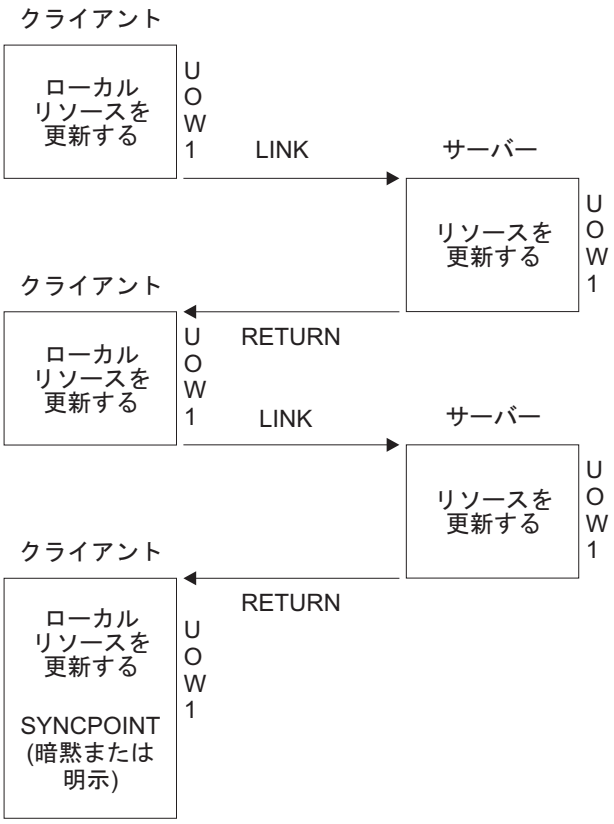


図 114. SYNCONRETURN オプションを使わない分散プログラム・リンクの使用例

注: 暗黙の同期点または明示の同期点によって、クライアントとサーバーの両方のリソースがコミットされます。クライアントに、クライアントとサーバーの両方のリソースをコミットする時点を判別する責任があるので、論理作業単位は 1 つしかありません。

クライアントが HANDLE ABEND コマンドを使用する場合を考慮する必要があります。クライアントがサーバーの異常終了を扱う場合には、サーバーが異常終了すると制御はクライアントに移ります。LINK コマンドに SYNCONRETURN オプションが指定されている場合も同様です。その場合は、クライアントが最小の終結処置を実行した後で異常終了を発行するようにすることをお勧めします。これにより、クライアントおよびサーバー両方の論理作業単位がバックアウトされます。

分散プログラム・リンクのプログラミングに関する考慮事項

分散プログラム・リンクを使用するアプリケーション・プログラムを作成する場合は、考慮すべき要素がいくつかあります。

同一のクライアント・タスクからの複数の分散プログラム・リンクの発行

SYNCONRETURN オプションが指定されていない限り、クライアント・タスクは、複数のトランザクション・コードを単一のクライアント作業単位で使用して、単一 CICS サーバー領域への分散プログラム・リンクを要求できません。クライアント・

タスクは、同一またはデフォルト・トランザクション・コードを使用して複数の分散プログラム・リンクを 1 つの CICS サーバー・システムに発行することができます。

クライアント・プログラムとサーバー・プログラム間のリソースの共用

サーバー・プログラムは、例えば TWA などの、クライアントのタスク存続時間のストレージへのアクセス権を持っていません。また、ファイル要求が機能シップされたものである場合を除き、クライアント・プログラムが使用している、例えばファイルなどのリソースに対するアクセス権も必ずしも持っていません。

同一の CICS システムに対する DPL と機能シップの混合

同じクライアント・タスクから同一の CICS システム対して、機能シップと DPL とを一緒に使用する場合は、特に注意が必要です。次の点について、考慮してください。

- クライアント・タスクはシップを要求する機能がないため、同一セッション (同一の論理単位、またはシステム初期設定パラメーターとして MROFSE=YES が指定されている) の中で、SYNCONRETURN オプションを指定した分散プログラム・リンクを使用します。分散プログラム・リンクは INVREQ 応答で失敗します。この場合には、EIBRESP2 は 14 に設定されます。
- クライアント・タスクは、同一クライアント論理作業単位の中では、要求を機能シップし、さらに TRANSID オプションを指定した分散プログラム・リンクを使用することができません。分散プログラム・リンクは INVREQ 応答で失敗します。この場合には、EIBRESP2 は 15 に設定されます。
- 機能シップされた要求が、SYNCONRETURN オプションを指定した DPL 要求に続いている場合には、サーバー作業論理単位とは別の作業論理単位で実行されます。
- 実行中の機能シップされた要求が、同一のサーバー領域に対する、TRANSID オプションを指定した DPL 要求の後に続いている場合は、デフォルト・ミラー・トランザクション・コードのもとではなく、TRANSID オプションで指定されているトランザクション・コードのもとで実行されます。機能シップされた要求は、クライアントがコミットするときに、クライアント論理作業単位全体の中の一部としてコミットされます。
- 実行中の機能シップされた要求が、SYNCONRETURN または TRANSID オプションを指定しない DPL 要求の前または後に続く場合には、クライアントがコミットするときに、クライアント論理作業単位全体の中の一部としてコミットされます。

機能シップについて詳しくは、「*CICS 相互通信ガイド*」のCICS 機能シップを参照してください。

同一の CICS システムに対する DPL と DTP の混合

同一アプリケーションで DPL と DTP の両方を使用する場合、特に、サーバー・プログラムで DTP を使用する場合には、注意する必要があります。例えば、SYNCONRETURN オプションを使用していない場合には、同期点を取る DPL サーバー・プログラムを必要とする DTP パートナーで同期点を取ることは避ける必要

があります。

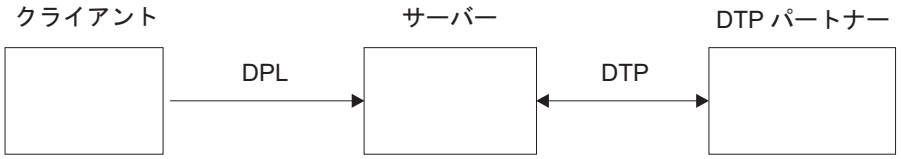


図 115. DPL と DTP を混合した例

分散プログラム・リンクのサブセットへのプログラムの制限

プログラムが分散プログラム・リンクの結果として実行される場合には、分散プログラム・リンクのサブセットと呼ばれる全 CICS API のサブセットに制限されます。サーバー・プログラムで禁止されるコマンドの要約は 421 ページの表 23 にあります。

EXECUTIONSET(DPLSUBSET) オプションを指定して、ローカル LINK コマンドによって呼び出すプログラムをこのサブセットに制限することを、プログラム・リソース定義にだけ指定することができます。禁止されているコマンドのどれかを使用すると、アプリケーション・プログラムを分散環境で使用する前にそれらを検出することができます。EXECUTIONSET(DPLSUBSET) オプションは非常に初期のテストのためにだけ使用し、稼働中には決して使用しないでください。

サーバー・プログラムがローカルで実行されている場合、次の考慮事項が適用されます。

- サーバー・プログラムに EXECUTIONSET(DPLSUBSET) が指定されている場合、SYNCONRETURN オプションにより、クライアント・プログラムに制御を返す前にローカル・サーバー・プログラムにおいて暗黙の同期点が取られます。この場合には、サーバー・プログラムがローカルで実行中であるために、クライアントとサーバーの両方のリソースがコミットされます。しかし、SYNCONRETURN は、クライアントにリカバリー可能リソースがない場合に使用するためのものである点に注意する必要があります。
- サーバー・プログラムに EXECUTIONSET(FULLAPI) を指定した場合、SYNCONRETURN オプションは無視されます。
- TRANSID オプションおよび DATALENGTH オプションは、ローカル・リンクの処理時には無視されますが、引数の形式は検査されます。例えば、TRANSID 引数をすべてブランクにすることはできません。

プログラムが呼び出された方法の判別

ASSIGN コマンドの STARTCODE オプションに返される 2 バイト値は、分散プログラム・リンク機能のサポート用に拡張され、サーバー・プログラムが分散プログラム・リンクのサブセットに制限されていることを検出できるようになっています。EXEC CICS コマンドに関するプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

ASSIGN コマンドを使用したユーザー関連情報へのアクセス

サーバー・プログラムで ASSIGN コマンドの USERID および OPID キーワードにより返される値は、クライアント CICS 領域とサーバー CICS 領域の間で使用される接続に対する ATTACHSEC オプションの定義方法によって異なります。例えば、サーバー・プログラムがクライアント・プログラムと同一の USERID および OPID 値にアクセスできるようにシステムを定義するか、あるいは ATTACHSEC オプションによって判別される異なった値にアクセスできるようにシステムを定義することができます。

ATTACHSEC(LOCAL) が指定されている場合は、OPID および USERID パラメーターと対応するユーザー ID は、示されている順序に従い、次の 1 つになります。

1. SESSIONS リソース定義が存在する場合は、その定義の (事前設定セキュリティーのための) USERID パラメーターに指定されたユーザー ID。
2. 接続リソース定義が存在し、事前設定セキュリティー・ユーザー ID がセッションに定義されていない場合は、その定義の SECURITYNAME パラメーターに指定されているユーザー ID。
3. セッションおよび接続定義の両方においてユーザー ID が指定されていない場合は、サーバー領域の DFLTUSER システム初期化パラメーターに指定されているユーザー ID。

ATTACHSEC に LOCAL 以外の値が指定されている場合、サインオン・ユーザー ID は、クライアント領域からの機能管理ヘッダー (FMH5) で受け取ったユーザー ID です。

リンク・セキュリティーおよび MRO による DPL セキュリティーについては、「*CICS RACF Security Guide*」を参照してください。

セキュリティーに関連して考慮しなければならない別の事項は、ASSIGN コマンドの CMDSEC および RESSEC オプションの使用についてです。これらのオプションは、サーバー領域のミラー・トランザクションに対するトランザクション定義の属性です。それらは、同一の TRANSID を使用する場合でも、クライアント領域における定義とは異なる定義を使用できます。

LINK コマンドの例外条件

DPL をサポートするためのエラー条件があります。

クライアント・プログラムに返される例外条件: クライアント・プログラムに返される条件コードは、サーバー・プログラムにおける「リモート・システムが不明」または「コミットの失敗」などのイベントを示しています。LINK コマンドで、EIBRESP2 値によって識別される、条件 INVREQ および LENGERR が起こる理由はいくつかあります。また、条件 ROLLEDBACK、SYSIDERR、および TERMERR も起こることがあります。これらのコマンドに関するプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

リモート・リージョンのミラー・トランザクションに障害が発生したときに、*以下の 2 つの状態が真の場合のみ*、DPL 要求を発行したアプリケーション・プログラ

ムでそのミラーの異常終了を処理して、そのトランザクションに固有のローカル・リソースをコミットすることができます。

1. アプリケーション・プログラムが、そのミラーの障害によって発生した異常終了を明示的に処理し、以下のいずれかを処理する場合。
   - トランザクションの正常終了による暗黙的な同期点要求の受信
   - または、明示的な同期点要求の発行。
2. リモート・ミラー・トランザクションが、アプリケーション・プログラムの作業単位の範囲内でリカバリー可能な作業を何も実行しない場合。すなわち、このミラーが、SYNCONRETURN を指定した分散プログラム・リンク (DPL) 要求のためだけに起動された場合。

他のすべてのケースでは、すなわち、アプリケーション・プログラムが異常終了を処理しない場合、またはミラーがすべてのリカバリー可能な作業 (例えば、リカバリー不能なファイルを含むファイル更新) を行う場合には、CICS は、トランザクションをバックアウトさせます。

ローカル・プログラム定義で、プログラムがリモート・プログラムであることを指定した場合には、HANDLE ABEND PROGRAM、LOAD、RELEASE、および XCTL コマンドで条件 PGMIDERR が起こります。この例外条件では EIBRESP2 の値が 9 になります。

サーバー・プログラムに返される例外条件: 条件 INVREQ は、禁止された API コマンドの使用も含みます。INVREQ がサーバー・プログラムに返されるのは、サーバー・プログラムが表 23 で要約されている禁止コマンド (EIBRESP2 の値が 200 になります) の 1 つを実行した場合です。サーバー・プログラムが条件 INVREQ を処理しない場合には、デフォルトの処置は、サーバー・プログラムが実行中のミラー・トランザクションを、異常終了コード ADPL で異常終了することです。

DPL 関連の例外条件に関するプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」マニュアルを参照してください。

表 23. DPL によって呼び出されるプログラムで禁止されている API コマンド

コマンド	オプション
ASSIGN	ALTSCRNHT ALTSCRNWD APLKYBD APLTEXT BTRANS COLOR DEFSCRNHT DEFSCRNWD DELIMITER DESTCOUNT DESTID DESTIDLENG DS3270 DSSCS EWASUPP EXTDS FACILITY FCI GCHARS GCODES GMMI HILIGHT INPARTN KATAKANA LDCMNEM LDCNUM MAPCOLUMN MAPHEIGHT MAPLINE MAPWIDTH MSRCONTROL NATLANGINUSE NEXTTRANSID NUMTAB OPCLASS OPSECURITY OUTLINE PAGENUM PARTNPAGE PARTNS PARTNSET PS QNAME SCRNHT SCRNWD SIGDATA SOSI STATIONID TCTUALENG TELLERID TERMCODE TERMPRIORITY TEXTKYBD TEXTPRINT UNATTEND USERNAME USERPRIORITY VALIDATION
CONNECT PROCESS	すべて
CONVERSE	すべて
EXTRACT ATTRIBUTES	すべて

表 23. DPL によって呼び出されるプログラムで禁止されている API コマンド (続き)

コマンド	オプション
EXTRACT PROCESS	すべて
FREE	すべて
HANDLE AID	すべて
ISSUE	ABEND CONFIRMATION ERROR PREPARE SIGNAL PRINT ABORT ADD END ERASE NOTE QUERY RECEIVE REPLACE SEND WAIT
LINK	INPUTMSG INPUTMSGLEN
PURGE MESSAGE	すべて
RECEIVE	すべて
RETURN	INPUTMSG INPUTMSGLEN
ROUTE	すべて
SEND	CONTROL MAP PARTNSET TEXT TEXT(MAPPED) TEXT(NOEDIT) PAGE
SIGNOFF	すべて
SIGNON	すべて
START	TERMID。この値はシステム間セッションの ID。(すなわち、発行側タスクの基本ファシリティーが端末ではなくセッションである。)
START CHANNEL	TERMID。この値はシステム間セッションの ID。(すなわち、発行側タスクの基本ファシリティーが端末ではなくセッションである。)
SYNCPOINT	LINK で SYNCONRETURN を指定した場合は、サーバー領域で発行可能。
WAIT TERMINAL	すべて
XCTL	INPUTMSG INPUTMSGLEN

以下のコマンドも制限付きのコマンドですが、LINK で SYNCONRETURN を指定した場合はサーバー領域で使用できます。

- CPIRR COMMIT
- CPIRR BACK
- EXEC DLI TERM
- CALL DLI TERM

上記のコマンドは、一定のオプションしか禁止されていないコマンドです。リストされているすべての APPC コマンドが禁止されるのは、主要機能を参照する場合です。これらのコマンドの 1 つである CONNECT PROCESS コマンドは、非 DPL 環境で主要機能を参照してもエラーの原因になります。CONNECT PROCESS コマンドがリストされているのは、このコマンドがサーバー・プログラムで主要機能を参照した場合、発生した例外条件が DPL エラーを示すためです。

非同期処理

リモートで開始されたトランザクションからの応答は、トランザクションを開始したタスクに必ずしも返されるわけではありません。これが、処理が**非同期**と呼ばれるわけです。非同期処理が有用なのは、リモート要求を処理している間はローカル・リソースを拘束する必要がないか、あるいは拘束したくない場合です。例えば、リモート・データベースへのオンライン照会によって、端末オペレーターは最初の照会への応答を待たずに照会の入力を続行することができます。

ローカル・トランザクションとまったく同様に、START コマンドを使用して、リモート・システムでトランザクションを開始することができます。RETRIEVE コマンドを使用すると、リモートから発行された START、CANCEL、SEND、または RECEIVE コマンドの結果としてタスクに保管されたデータを、ローカル・トランザクションであったかのように検索することができます。

分散トランザクション処理 (DTP)

DTP の主な利点は、2 つのトランザクションがセッションの排他制御権を持ち、「会話」を行うことができる点です。DTP が特に使用されるのは、リモート・リソースをリモート処理する必要がある場合、またはシステム間でデータを転送する必要がある場合です。また、非常に柔軟性があり、効率的なアプリケーションを設計することもできます。DTP は、EXEC CICS または CPI コミュニケーションのいずれかで使用することができます。 EXEC CICS API を使用して LU タイプ 6.2 非マップ式会話を保持する DTP アプリケーション・プログラム、および CICS 相互通信機能を使用するアプリケーションには、C、C++、およびアセンブラー言語を使用できます。

DTP は、APPC をサポートするパートナーであれば、いろいろなパートナー (CICS および非 CICS プラットフォームを含む) と組み合わせて使用することができます。DTP について詳しくは、「*CICS Distributed Transaction Programming Guide*」および「*CICS Family: Interproduct Communication*」の各資料を参照してください。

共通プログラミング・インターフェース・コミュニケーション (CPI コミュニケーション)

CPI コミュニケーションは、既存の CICS APPC サポートに代替 API を提供します。CPI コミュニケーションは APPC 接続での DTP を提供し、COBOL、C、C++、PL/I、およびアセンブラー言語で使用できます。

CPI コミュニケーションは、複数のシステム・プラットフォームで構成される APPC ネットワークで使用可能な API を定義します。APPC ネットワークでは、この整合性のある共通の API にメリットがあります。

CPI コミュニケーション・インターフェースは、APPC API を提供する任意のシステム上のアプリケーションと会話することができます。このアプリケーションには、CICS プラットフォームのアプリケーションも含まれます。ある会話の一方で EXEC CICS APPC API コマンドを使用し、もう一方で CPI コミュニケーション・コマンドを使用することもできます。

CPI コミュニケーションでは、パートナー・プログラムとの会話を始めるための特定の情報 (サイド情報) が必要です。システム・プログラマーが管理を担当しているパートナー・リソースを使用することにより、サイド情報の CICS における実装を実現できます。

アプリケーションからの CPI コミュニケーション・インターフェースの呼び出しは、その呼び出しを CICS CPI コミュニケーション・スタブ (DFHCPLC) とリンク・エディットすることによって解決することができます。この方法に関する情報は、140 ページの『CICS 提供インターフェース・モジュールの組み込み』に記載されています。

CPI コミュニケーション API は、汎用の呼び出しインターフェースとして定義されています。このインターフェースは、「*Common Programming Interface Communications Reference*」に説明されています。

外部 CICS インターフェース (EXCI)

外部 CICS インターフェースは、MVS で実行されている非 CICS プログラム (クライアント・プログラム) が CICS 領域で実行されているプログラム (サーバー・プログラム) を呼び出して、連絡域を通じてデータの受け渡しができるようにするアプリケーション・プログラミング・インターフェースです。CICS プログラムは、別の CICS プログラムがリンクしたかのように呼び出されます。

このプログラミング・インターフェースを使用すると、CICS システムにセッション (パイプ) を割り振ってオープンし、それらを介して分散プログラム・リンク (DPL) 要求を渡すことができます。CICS 領域間通信 (IRC) は、これらの要求をサポートし、1 つの MRO セッションの各パイプ・マップをサポートします。

EXCI に関するプログラミング情報については、「*CICS External Interfaces Guide*」の外部 CICS インターフェースを参照してください。

外部 CICS インターフェースを使用するクライアント・プログラムは、同一の MVS アドレス・スペースに同時に存在する異なるユーザー (同一または別々の TCB) の複数セッションを、ユーザーが互いに認識していない場合でも、または互いに影響を与えることなく操作することができます。

外部 CICS インターフェースは、2 つのプログラミング・インターフェースの形式を提供します。

- EXCI CALL インターフェースは、次のことを可能にする 6 つのコマンドから構成されます。
  - MVS で実行されている非 CICS プログラムから CICS システムへセッションを割り振り、オープンする
  - これらのセッションで非 CICS プログラムから DPL 要求を発行する
  - DPL 要求の完了時にセッションをクローズし、割り振り解除する
- EXEC CICS インターフェースは次のものを提供します。
  - 1 回の呼び出しで EXCI CALL インターフェースの 6 つのコマンドをすべて実行する、単一複合コマンド (LINK PROGRAM)

そのコマンドの形式は、CICS コマンド・レベル・アプリケーション・プログラミング・インターフェースの分散プログラム・リンク・コマンドと同じです。

CICS は、外部 CICS インターフェースを使用するアプリケーションで MVS リソース・リカバリー・サービス (RRS) をサポートします。これは、以下のことを意味します。

- 内部で CICS サーバー・プログラムがリカバリー可能リソースを変更する作業単位は、ここで、EXCI クライアント・プログラムに関連した MVS のリカバリー単位の一部になる。
- サーバー・プログラムがクライアントに制御を返すとき、または 複数の EXCI DPL 呼び出しにわたって継続するとき、EXCI クライアントがリカバリー単位をコミットするかバックアウトするかを決定するまで、CICS サーバー作業単位がコミットされることがある。

第 5 部 データ・マッピング

アプリケーション・データを XML に変換したりその逆の変換を行ったりするためのマッピングを作成します。

© Copyright IBM Corp. 1989, 2009

第 28 章 アプリケーション・データおよび XML のマッピングと変換

アプリケーション・プログラムを書き込んで、アプリケーションのバイナリー・データを XML に変換したり逆の変換を行ったりすることができます。CICS は、多数の高水準言語をサポートし、ランタイム処理中のデータの変換方法をマップするための XML 支援機能を提供します。CICS は Web サービス・サポートの一部として、同じテクノロジーを使用してアプリケーション・データを SOAP メッセージ内の XML にマップします。

始める前に

CICS によるデータまたは XML の変換時に、XML 支援機能や任意の妥当性検査を実行するには、Java をインストールしておく必要があります。

このタスクについて

このアプローチを使用してアプリケーション・データから XML への (またはその逆の) 変換を行う利点は、 XML パーサーが備える機能を超える処理が CICS によって行われることです。 CICS は、XML を解釈して、アプリケーション・データのレコード・ベース変換を行うことができます。したがって、このアプローチを使用することで、XML を処理するアプリケーションをより簡単かつ迅速に作成できます。

CICS は、IBM z/OS XML システム・サービス (XMLSS) パーサーを使用して、アプリケーションのバイナリー・データから XML への (またはその逆の) 変換を行います。このパーサーは、2 GB 境界より上のストレージを使用することで 2 GB 境界より下のストレージをユーザー・プログラム用により多く残すため、パフォーマンスが向上します。 XMLSS パーサーではまた、XML 構文解析を zSeries® Application Assist Processor (zAAP) にオフロードして、CPU 時間の空きを作ることでトランザクションのコストを削減することも可能です。 zAAP の詳細については、IBM Redbook の「*zSeries Application Assist Processor (zAAP) Implementation*」(http://www.redbooks.ibm.com/abstracts/sg246386.html) を参照してください。

CICS の XML 支援機能は、アプリケーションのバイナリー・データを XML に変換したり、XML をアプリケーションのバイナリー・データに変換したりするために必要な成果物の作成を支援する提供ユーティリティーです。XML 支援機能では、成果物を z/OS UNIX® 上のバンドル・ディレクトリーまたは指定された別の場所に作成することができます。

1. XML 支援機能を使用して、マッピングを作成します。
2. CICS でリソースを作成して、マッピングを使用可能にします。
3. アプリケーション・プログラムを作成または更新して、TRANSFORM API コマンドを使用します。 アプリケーションでは、チャネル・ベースのインターフェースを使用する必要があります。

© Copyright IBM Corp. 1989, 2009

4. アプリケーションを実行して、変換が意図したとおりに行われるかどうかをテストします。 妥当性検査をオンにして、CICS がデータを正しく変換するかどうかを確認することができます。

次のタスク

これらのステップの詳細は、以下のトピックで説明されています。

CICS XML 支援機能

CICS XML 支援機能は、 XML と高水準言語構造との間の相互変換を支援する一連のバッチ・ユーティリティーです。この支援機能により、 XML 処理を実行するアプリケーションを最小限のプログラミングで迅速にデプロイすることが可能になります。

CICS 用の XML 支援機能を使用すると、 XML の解析または構築のために作成する必要のあるコードの量を軽減できます。 CICS では、XML フラグメントとアプリケーション・プログラムのデータ構造との間でデータが相互に変換されます。

XML 支援機能では、単純な言語構造からの XML スキーマの作成や既存の XML スキーマからの言語構造の作成が可能であり、COBOL、C/C++、および PL/I をサポートしています。また、 CICS が実行時に XML データをバイナリー・アプリケーション・データに自動的に変換する (または逆の変換を行う) ために使用するメタデータも生成されます。このメタデータは XML バインディングで定義され、 z/OS UNIX 上に保管されます。XML バインディング用のスキーマは、z/OS UNIX 上の /usr/lpp/cicsts/cicsts41/schemas/xmltransform/ディレクトリーにあります。

CICS XML 支援機能は以下の 2 つのユーティリティー・プログラムで構成されています。

DFHLS2SC
: このユーティリティーは言語構造から XML スキーマおよびバインディングを生成します。

DFHSC2LS
: このユーティリティーはアプリケーション・プログラムで使用できる XML バインディングおよび言語構造を生成します。WSDL 文書または XML スキーマを入力データとして使用できます。

hlq.XDFHINST ライブラリーには、両方のプログラムを実行するための JCL プロシージャーがあります (*hlq* は CICS インストールの高位修飾子です)。

関連タスク

500 ページの『言語構造からのマッピングの生成』
XML をアプリケーション・データから作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータおよび XML を変換する方法を記述する必要があります。これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または DB2 レコードから開始できます。

502 ページの『XML スキーマからのマッピングの生成』
既存の XML スキーマに準拠する XML からアプリケーション・データを作成

したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータを変換する方法を記述する必要があります。これは、XML スキーマまたは WSDL 文書から開始できます。

関連情報

配置エラーの診断

DFHLS2SC: 高水準言語から XML スキーマへの変換

カタログ式プロシージャーの DFHLS2SC は、XML スキーマと XML バインディング・ファイルを高水準言語構造から生成します。XML の構文解析または作成が可能な CICS プログラムを作成する場合は、DFHLS2SC を使用してください。

このプロシージャーを使用する際には、DFHLS2SC のジョブ制御ステートメント、そのシンボリック・パラメーター、入力パラメーターとその記述、およびジョブ例が役に立ちます。

DFHLS2SC のジョブ制御ステートメント

JOB ジョブを開始します。

EXEC プロシージャー名 (DFHLS2SC) を指定します。

INPUT.SYSUT1 DD
入力を指定します。入力パラメーターは通常、入力ストリームで指定します。ただし、データ・セットまたは区分データ・セットのメンバーで定義することもできます。

シンボリック・パラメーター

DFHLS2SC では以下のシンボリック・パラメーターが定義されています。

JAVADIR=*path*
DFHLS2SC によって使用される Java ディレクトリーの名前を指定します。このパラメーターの値が `/usr/lpp/` に付加されることにより、完全なパス名 `/usr/lpp/path` が形成されます。

通常、このパラメーターは指定しません。デフォルト値は、**JAVADIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に提供された値です。

PATHPREF=*prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する、オプションの接頭部を指定します。デフォルトは空ストリングです。

通常、このパラメーターは指定しません。デフォルト値は、**JAVADIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に提供された値です。

SERVICE=*value*
このパラメーターは、IBM サポートから指示された場合にのみ、使用してください。

TMPDIR=*tmpdir*
DFHLS2SC が z/OS UNIX 内で一時ワークスペースとして使用するディレクトリーの場所を指定します。ジョブを実行するユーザー ID には、このディレクトリーへの読み取りおよび書き込み権限がある必要があります。

デフォルト値は `/tmp` です。

TMPFILE=*tmpprefix*

一時ワークスペース・ファイルの名前を構成するときに DFHLS2SC によって使用される接頭部を指定します。

デフォルト値は SC2WS です。

USSDIR=*path*

z/OS UNIX ファイル・システム内の CICS TS ディレクトリーの名前を指定します。このパラメーターの値は、/usr/lpp/cicsts/ に付加されて、/usr/lpp/cicsts/*path* の完全なパス名が形成されます。

通常、このパラメーターは指定しません。デフォルト値は、**USSDIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に提供された値です。

一時ワークスペース

DFHLS2SC により、実行時に以下の 3 つの一時ファイルが作成されます。

tmpdir/*tmpprefix*.in

tmpdir/*tmpprefix*.out

tmpdir/*tmpprefix*.err

ここで、

tmpdir は、**TMPDIR** パラメーターで指定された値です。

tmpprefix は、**TMPFILE** パラメーターで指定された値です。

ファイルのデフォルト名 (**TMPDIR** および **TMPFILE** が指定されていない場合) は、次のとおりです。

/tmp/LS2SC.in

/tmp/LS2SC.out

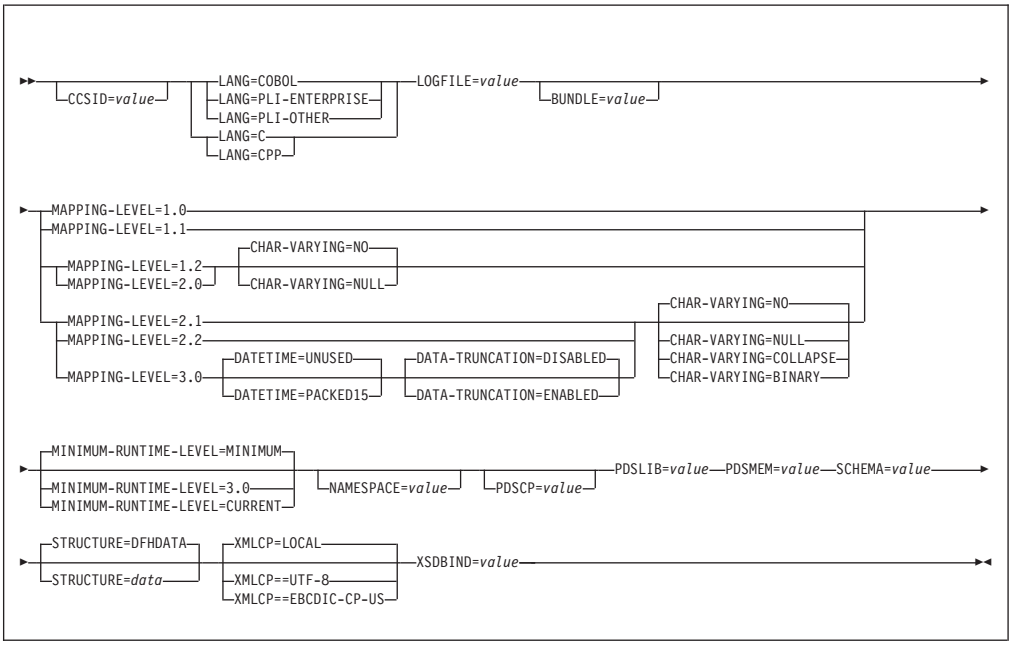
/tmp/LS2SC.err

重要: DFHLS2SC は、生成された z/OS UNIX ファイル名へのアクセスをロックしません。したがって、DFHLS2SC の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することはできないため、予測不能な障害が生じます。

そのため、このような状況を回避する命名規則と操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーターの **SYSUID** を使用して、個々のユーザーにとって固有のワークスペース・ファイル名を生成することができます。

これらの一時ファイルは、ジョブが終了する前に削除されます。

DFHLS2SC の入力パラメーター



パラメーターの使用法

- 入力パラメーターは任意の順序で指定できます。
- 各パラメーターは、新しい行から開始する必要があります。
- パラメーター (および使用する場合は、その継続文字) は、72 桁を超えて拡張することはできません。73 桁から 80 桁の間はブランクにする必要があります。
- パラメーターが単一行に収まらないほど長い場合は、行の最後にアスタリスク (*) 文字を使用して、パラメーターが次の行に続くことを示します。アスタリスクの前のすべての部分 (スペースを含む) は、パラメーターの一部とみなされます。例を以下に示します。

  ```
  XSDBIND=xsdbinddir*
  /app1
  ```

  は

  XSDBIND=xsdbinddir/app1 と同等です。
- 行の先頭文字位置にある # 文字は、コメント文字です。この行は無視されます。

パラメーターに関する説明

BUNDLE=*value*

z/OS UNIX 上のバンドル・ディレクトリーのパスと名前を指定します。この値を指定する場合、XML 支援機能によって、XML スキーマとバインディングがバンドル・ディレクトリーで生成され、バンドル・マニフェストが作成されます。このパラメーターのパス情報は、**SCHEMA** パラメーターと **XSDBIND** パラメーター上のパス情報を指定変更します。

ディレクトリー名の代わりにアーカイブ・ファイルを指定することもできます。XML 支援機能では .zip および .jar アーカイブがサポートされています。ただし、BUNDLE リソースをインストールする前にアーカイブを圧縮解除する必要があります。

このパラメーターを指定しない場合、CICS によって、XML スキーマとバインディングがこれらのパラメーターで指定されている場所に配置されます。

CCSID=*value*
アプリケーション・データ構造で文字データをエンコードするために実行時に使用する CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。*value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります。このパラメーターを指定しない場合、アプリケーション・データ構造はシステム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用することができます。

CHAR-VARYING=NO|NULL|COLLAPSE|BINARY
マッピング・レベルが 1.2 以上の場合に、言語構造内の文字フィールドがどのようにマップされるのかを指定します。COBOL の文字フィールドは、タイプ X のピクチャー節 (`PIC(X) 10` など) です。C/C++ の文字フィールドは文字配列です。このパラメーターは、Enterprise および他の PL/I 言語構造には適用されません。以下のオプションを選択できます。

NO 文字フィールドは `xsd:string` にマップされ、固定長のフィールドとして処理されます。データの最大長は、フィールドの長さと同じです。NO は、マッピング・レベル 2.0 以前での、COBOL および PL/I の **CHAR-VARYING** パラメーターのデフォルト値です。

NULL 文字フィールドは `xsd:string` にマップされ、ヌル終了ストリングとして処理されます。CICS は、XML スキーマからの変換時に、終了を表すヌル文字を追加します。文字ストリングの最大長は、言語構造で示された長さより 1 文字少ない長さとして計算されます。NULL は、C/C++ での **CHAR-VARYING** パラメーターのデフォルト値です。

COLLAPSE
文字フィールドは `xsd:string` にマップされます。フィールド内の末尾の空白文字は XML スキーマには含まれません。COLLAPSE は、マッピング・レベル 2.1 以上での、COBOL および PL/I の **CHAR-VARYING** パラメーターのデフォルト *value* です。

BINARY
文字フィールドは `xsd:base64binary` データ型としてマップされ、固定長フィールドとして処理されます。**CHAR-VARYING** パラメーター上の BINARY *value* は、マッピング・レベル 2.1 以上でのみ使用可能です。

DATA-TRUNCATION=DISABLED|ENABLED
固定長フィールド構造で可変長データが許容されるかどうかを指定します。

DISABLED
予期されている固定長よりデータが短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
予期されている固定長よりデータが短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME=UNUSED|PACKED15
高水準言語構造内の日時フィールド (CICS ABSTIME 値を含む) がタイム・スタンプとしてマップされるかどうかを指定します。

PACKED15
すべての日時フィールドがタイム・スタンプとしてマップされます。

UNUSED
いかなる日時フィールドもタイム・スタンプとしてマップされません。このマッピングはデフォルトです。

このパラメーターはマッピング・レベル 3.0 で設定できます。

LANG=COBOL
高水準言語構造のプログラミング言語が COBOL であることを指定します。

LANG=PLI-ENTERPRISE
高水準言語構造のプログラミング言語が Enterprise PL/I であることを指定します。

LANG=PLI-OTHER
高水準言語構造のプログラミング言語が Enterprise PL/I 以外のレベルの PL/I であることを指定します。

LANG=C
高水準言語構造のプログラミング言語が C であることを指定します。

LANG=CPP
高水準言語構造のプログラミング言語が C++ であることを指定します。

LOGFILE=*value*
DFHLS2SC によって自身のアクティビティー・ログが書き込まれ、情報がトレースされるファイルの、完全修飾された z/OS UNIX 名。このファイルが存在しない場合、DFHLS2SC はこのファイルを作成しますが、ディレクトリー構造は作成しません。

通常、このファイルを使用することはありませんが、DFHLS2SC で問題が発生した場合は、IBM サービス組織に要求される場合があります。

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0}
XML バインディングおよび言語構造の生成時に支援機能で使用するマッピング・レベルを指定します。使用可能な最新のマッピング・レベルを使用することをお勧めします。Atom フィードの XML バインディングを作成している場合は、マッピング・レベル 3.0 を使用する必要があります。

MINIMUM-RUNTIME-LEVEL={MINIMUM|3.0|CURRENT}
XML バインディングをデプロイすることが可能な、CICS の最小ランタイム環

境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。選択可能なオプションは以下のとおりです。

MINIMUM
選択可能な CICS の最小ランタイム・レベルが、指定したパラメーターに基づいて自動的に割り振られます。

3.0 CICS XML 支援機能を使用して、高機能のデータ・マッピングを利用する場合は、ランタイム・レベル 3.0 以上を指定します。

CURRENT
バインディング・ファイルの生成にランタイム環境の使用を設定している CICS 領域に、生成されたバインディング・ファイルをデプロイする場合、このランタイム・レベルを使用します。

NAMESPACE=*value*
生成された XML スキーマで使用する CICS の名前空間を指定します。Atom フィードの場合、CICS は Atom 名前空間とともに Atom フィードでこの名前空間を提供します。

このパラメーターを指定しない場合、CICS では名前空間が自動的に生成されます。

PDSCP=*value*
区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。このパラメーターを指定しない場合、z/OS UNIX システム・サービス・コード・ページが使用されます。例えば、`PDSCP=037` を指定することができます。

PDSLIB=*value*
処理される高水準言語データ構造が含まれる区分データ・セットの名前を指定します。

制約事項: 区分データ・セット内のレコードは、80 バイトの固定長である必要があります。

PDSMEM=*value*
処理される高水準言語構造を含む区分データ・セット・メンバーの名前を指定します。

SCHEMA=*value*
XML スキーマが書き込まれるファイルの、完全修飾された z/OS UNIX 名。XML スキーマは WSDL 2.0 仕様に準拠しています。このファイルが存在しない場合、DFHLS2SC はこのファイルを作成しますが、ディレクトリー構造は作成しません。

STRUCTURE=*data*
C および C++ での最上位データ構造の名前。デフォルトは DFHDATA です。

XMLCP=LOCAL|UTF-8|EBCDIC-CP-US
XML バインディングを生成するために使用されるコード・ページを指定します。

LOCAL
この値はデフォルトです。XML がローカル・コード・ページを使用して生成され、XML スキーマでエンコード・タグが生成されないことを指定します。

UTF-8 XML が UTF-8 コード・ページを使用して生成されることを指定します。 XML スキーマでエンコード・タグが生成されます。このオプションを指定する場合、XML スキーマが異なるプラットフォーム間でコピーされるときにエンコードが正しく維持されるようにする必要があります。

EBCDIC-CP-US
XML が US EBCDIC コード・ページを使用して生成されることを指定します。 XML スキーマでエンコード・タグが生成されます。

XSDBIND=*value*
XSD バインディングの完全修飾された z/OS UNIX 名。このファイルが存在しない場合、DFHLS2SC はこのファイルを作成しますが、ディレクトリー構造は作成しません。ファイル拡張子は .xsdbind です。

関連タスク

500 ページの『言語構造からのマッピングの生成』
XML をアプリケーション・データから作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータおよび XML を変換する方法を記述する必要があります。これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または DB2 レコードから開始できます。

DFHSC2LS: XML スキーマから高水準言語への変換

カタログ式プロシージャーの DFHSC2LS では、高水準言語データ構造および XML バインディングが XML スキーマまたは WSDL 文書から生成されます。XML の構文解析または作成が可能な CICS プログラムを作成する場合は、DFHSC2LS を使用してください。

このプロシージャーを使用する際には、DFHSC2LS のジョブ制御ステートメント、そのシンボリック・パラメーター、入力パラメーターとその記述、およびサンプル・ジョブが役に立ちます。

DFHSC2LS のジョブ制御ステートメント

JOB ジョブを開始します。

EXEC プロシージャー名 (DFHSC2LS) を指定します。

INPUT.SYSUT1 DD
入力を指定します。入力パラメーターは通常、入力ストリームで指定します。ただし、データ・セットまたは区分データ・セットのメンバーで定義することもできます。

シンボリック・パラメーター

DFHSC2LS では以下のシンボリック・パラメーターが定義されています。

JAVADIR=*path*
DFHSC2LS によって使用される Java ディレクトリーの名前を指定します。このパラメーターの値が `/usr/lpp/` に付加されることにより、完全なパス名 `/usr/lpp/path` が形成されます。

通常、このパラメーターは指定しません。デフォルト値は、**JAVADIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に提供された値です。

PATHPREF=*prefix*
他のパラメーターで使用される z/OS UNIX ディレクトリー・パスを拡張する、オプションの接頭部を指定します。デフォルトは空ストリングです。

通常、このパラメーターは指定しません。デフォルト値は、**JAVADIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に提供された値です。

TMPDIR=*tmpdir*
DFHSC2LS が一時ワークスペースとして使用する z/OS UNIX 内のディレクトリーの場所を指定します。ジョブを実行するユーザー ID には、このディレクトリーへの読み取りおよび書き込み権限がある必要があります。

デフォルト値は `/tmp` です。

TMPFILE=*tmpprefix*
DFHSC2LS が一時ワークスペース・ファイルの名前を構成するために使用する接頭部を指定します。

デフォルト値は `SC2LS` です。

USSDIR=*path*
z/OS UNIX ファイル・システム内の CICS TS ディレクトリーの名前を指定します。このパラメーターの値は、`/usr/lpp/cicsts/` に付加されて、`/usr/lpp/cicsts/path` の完全なパス名が形成されます。

通常、このパラメーターは指定しません。デフォルト値は、**USSDIR** パラメーターで CICS インストール・ジョブ (DFHISTAR) に提供された値です。

SERVICE=*value*
このパラメーターは、IBM サポートから指示された場合にのみ、使用してください。

一時ワークスペース

DFHSC2LS により、実行時に以下の 3 つの一時ファイルが作成されます。

tmpdir/*tmpprefix*.in

tmpdir/*tmpprefix*.out

tmpdir/*tmpprefix*.err

ここで、

tmpdir は、**TMPDIR** パラメーターで指定された値です。

tmpprefix は、**TMPFILE** パラメーターで指定された値です。

ファイルのデフォルト名 (**TMPDIR** および **TMPFILE** が指定されていない場合) は、次のとおりです。

/tmp/SC2LS.in

/tmp/SC2LS.out

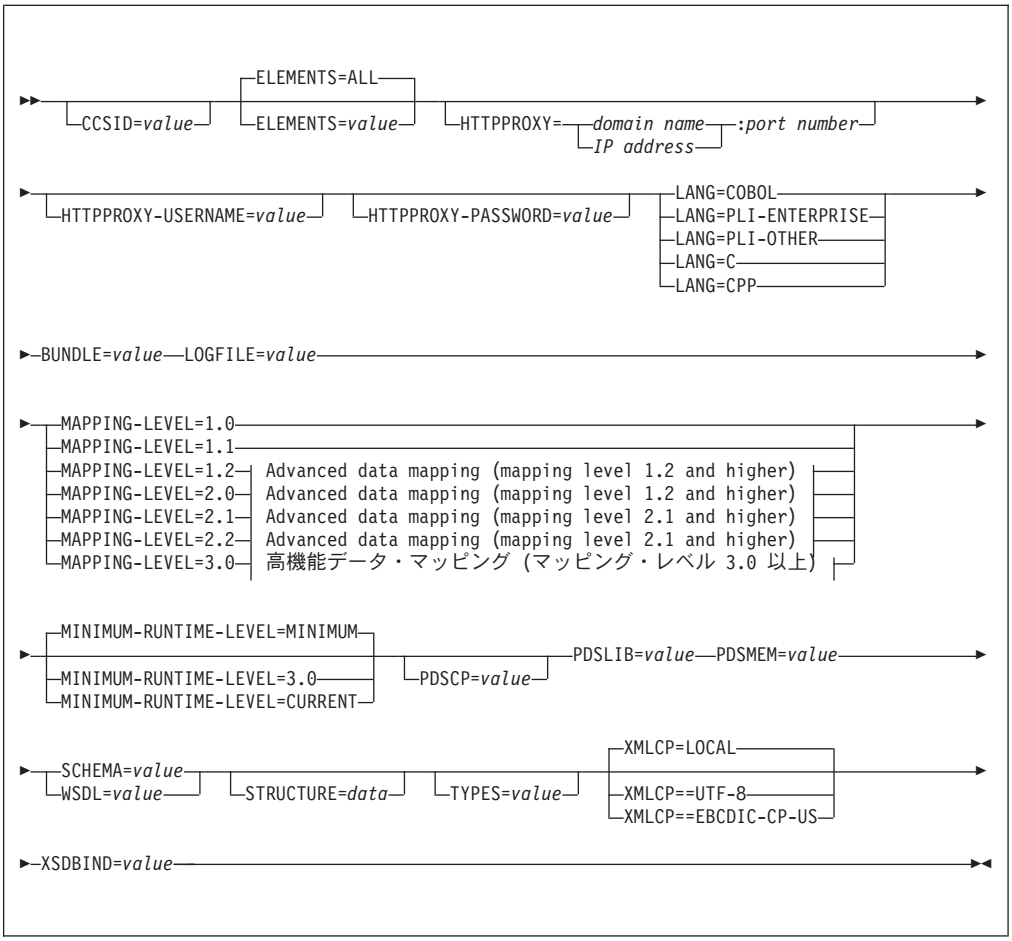
/tmp/SC2LS.err

重要: DFHSC2LS は、生成された z/OS UNIX ファイル名へのアクセスをロックしません。したがって、DFHSC2LS の複数のインスタンスが同時に実行され、同じ一時ワークスペース・ファイルを使用する場合、あるジョブがワークスペース・ファイルを使用しているときに、他のジョブがそれらのファイルを上書きするのを防止することができないため、予測不能な障害が生じます。

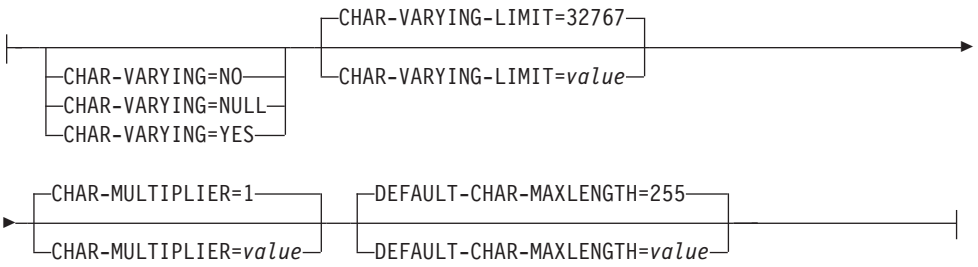
そのため、このような状況を回避する命名規則と操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーターの **SYSUID** を使用して、個々のユーザーにとって固有のワークスペース・ファイル名を生成することができます。

これらの一時ファイルは、ジョブが終了する前に削除されます。

DFHSC2LS の入力パラメーター

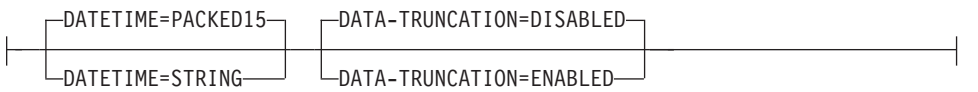


Advanced data mapping (mapping level 1.2 and higher):



Advanced data mapping (mapping level 2.1 and higher):

高機能データ・マッピング (マッピング・レベル 3.0):



パラメーターの使用法

- 入力パラメーターは任意の順序で指定できます。
- 各パラメーターは、新しい行から開始する必要があります。
- パラメーター (および使用する場合は、その継続文字) は、72 桁を超えて拡張することはできません。73 桁から 80 桁の間はブランクにする必要があります。
- パラメーターが単一行に収まらないほど長い場合は、行の最後にアスタリスク (*) 文字を使用して、パラメーターが次の行に続くことを示します。アスタリスクの前のすべての部分 (スペースを含む) は、パラメーターの一部とみなされます。例を以下に示します。

  ```
  XSDBIND=xsdbinddir*
  /app1
  ```

  は

  XSDBIND=xsdbinddir/app1 と同等です。
- 行の先頭文字位置にある # 文字は、コメント文字です。この行は無視されます。

パラメーターに関する説明

BUNDLE=*value*

z/OS UNIX 上のバンドル・ディレクトリーのパスと名前を指定します。この値を指定する場合、XML 支援機能によって、XML バインディングがバンドル・ディレクトリー内に生成され、バンドル・マニフェストが作成されます。このパ

ラメーターのパス情報は、**XSDBIND** パラメーター上のパス情報を指定変更します。このパラメーターを指定しない場合、CICS によって、XML バインディングが指定の場所に配置されます。

CCSID=*value*
アプリケーション・データ構造で文字データをエンコードするために実行時に使用する CCSID を指定します。このパラメーターの値は、**LOCALCCSID** システム初期設定パラメーターの値を指定変更します。*value* は、Java および z/OS 変換サービスによってサポートされている EBCDIC CCSID である必要があります。このパラメーターを指定しない場合、アプリケーション・データ構造はシステム初期設定パラメーターで指定された CCSID を使用してエンコードされます。

このパラメーターは任意のマッピング・レベルで使用することができます。ただし、生成されたファイルを CICS TS 3.1 領域に配置する場合は、APAR PK23547 を適用して Web サービスのバインディング・ファイルをインストールするためにコードの最小ランタイム・レベルを達成する必要があります。

CHAR-MULTIPLIER=1|*value*
マッピング・レベルが 1.2 以上の場合に各文字に許可されるバイト数を指定します。このパラメーターの *value* は、1 から 2 147 483 647 の範囲の正整数で指定できます。すべての非数字ベースのマッピングが、この乗数の対象になります。 2 進、数値、ゾーン、パック 10 進数の各フィールドは、この乗数の対象外です。

例えば、DBCS 文字を使用していて、実行時にすべての 2 バイト文字を潜在的なシフトアウトおよびシフトイン文字で囲むためのスペースを使用できるよう 3 の乗数を選択するときは、このパラメーターが便利です。

CHAR-VARYING=NO|NULL|YES
マッピング・レベルが 1.2 以上の場合に可変長文字データがどのようにマップされるのかを指定します。可変長 2 進データ型は常に、コンテナーまたは可変構造のいずれかにマップされます。このパラメーターを指定しない場合、デフォルトのマッピングは、指定された言語によって決まります。以下のオプションを選択できます。

NO 可変長文字データは、固定長ストリングとしてマップされます。

NULL 可変長文字データは、ヌル終了ストリングにマップされます。

YES PL/I の場合、可変長文字データは CHAR VARYING データ型にマップされます。COBOL、C、および C++ 言語の場合、可変長文字データは、 2 つの関連エレメント (データ長およびデータ) で構成される同等の表現にマップされます。

CHAR-VARYING-LIMIT=32767|*value*
マッピング・レベルが 1.2 以上の場合に、言語構造にマップされる 2 進データおよび可変長文字データの最大サイズを指定します。文字または 2 進データが、このパラメーターで指定されている値よりも大きい場合は、コンテナーにマップされ、生成された言語構造でそのコンテナー名が使用されます。 *value* は 0 からデフォルトの 32 767 バイトの範囲で指定できます。

DATA-TRUNCATION=DISABLED|ENABLED
固定長フィールド構造で可変長データが許容されるかどうかを指定します。

DISABLED
予期されている固定長よりデータが短い場合に、CICS は切り捨てられたデータを拒否してエラー・メッセージを発行します。

ENABLED
予期されている固定長よりデータが短い場合に、CICS は切り捨てられたデータを許容して、欠落データをヌル値として処理します。

DATETIME=PACKED15|STRING
xsd:dateTime フィールドが CICS ABSTIME データ形式またはテキストにマップされることを指定します。

PACKED15
xsd:dateTime フィールドは CICS ABSTIME 形式にマップされます。

STRING
xsd:dateTime フィールドはテキストにマップされます。このマッピングはこれまでのすべてのマッピング・レベルと同じです。

このパラメーターはマッピング・レベル 3.0 で使用できます。

DEFAULT-CHAR-MAXLENGTH=255|*value*
マッピング・レベルが 1.2 以上で、XML スキーマ文書または WSDL 文書で長さが指定されていない場合に、マッピングする文字の文字データのデフォルトの配列長を指定します。このパラメーターの *value* は、1 から 2 147 483 647 の範囲の正整数で指定できます。

ELEMENTS=ALL|*value*
有効にするグローバル・エレメントのローカル名のリストを定義します。デフォルト値 ALL は、すべてのグローバル・エレメントが有効であることを示します。

HTTPPROXY={*domain name*|*IP address*}:*port number*
XML スキーマまたは WSDL 文書に、インターネット上にある他の XML スキーマまたは WSDL ファイルへの参照が含まれていて、DFHSC2LS を実行しているシステムがインターネットへのアクセスにプロキシー・サーバーを使用している場合は、そのプロキシー・サーバーのドメイン名か IP アドレス、およびポート番号を指定します。例:

```
HTTPPROXY=proxy.example.com:8080
```

それ以外の場合、このパラメーターは不要です。

HTTPPROXY-USERNAME=*value*
DFHSC2LS を実行しているシステムが HTTP プロキシー・サーバーを使用してインターネットにアクセスし、HTTP プロキシー・サーバーが基本認証を使用している場合に、**HTTPPROXY-PASSWORD** と共に使用する HTTP プロキシー・ユーザー名を指定します。このパラメーターは、**HTTPPROXY** も指定する場合にのみ使用できます。

HTTPPROXY-PASSWORD=*value*
DFHSC2LS を実行しているシステムが HTTP プロキシー・サーバーを使用してインターネットにアクセスし、HTTP プロキシー・サーバーが基本認証を使用し

ている場合に、**HTTPPROXY-USERNAME** と共に使用する HTTP プロキシー・パスワードを指定します。このパラメーターは、**HTTPPROXY** も指定する場合にのみ使用できます。

INLINE-MAXOCCURS-LIMIT=1|*value*
インラインの可変反復内容を XML 属性の maxOccurs 属性に基づいて使用するかどうかを指定します。

INLINE-MAXOCCURS-LIMIT パラメーターは、マッピング・レベル 2.1 以上でのみ使用できます。**INLINE-MAXOCCURS-LIMIT** の *value* は、0 から 32 767 の範囲の正整数で指定できます。値が 0 の場合は、インライン・マッピングが使用されないことを示します。値が 1 の場合は、オプションのエレメントがインラインでマップされることを示します。 maxOccurs 属性の *value* が **INLINE-MAXOCCURS-LIMIT** の *value* より大きい場合は、コンテナー・ベースのマッピングが使用されます。それ以外の場合はインライン・マッピングが使用されます。

可変反復リストをインラインでマップする場合は、繰り返しデータの 1 つの項目の長さを考慮してください。長いインスタンスがほとんどない場合は、コンテナー・ベースのマッピングの方が望ましい方法です。短いインスタンスが多数ある場合は、インライン・マッピングの方が望ましい可能性があります。

LANG=COBOL
高水準言語構造のプログラミング言語が COBOL であることを指定します。

LANG=PLI-ENTERPRISE
高水準言語構造のプログラミング言語が Enterprise PL/I であることを指定します。

LANG=PLI-OTHER
高水準言語構造のプログラミング言語が Enterprise PL/I 以外のレベルの PL/I であることを指定します。

LANG=C
高水準言語構造のプログラミング言語が C であることを指定します。

LANG=CPP
高水準言語構造のプログラミング言語が C++ であることを指定します。

LOGFILE=*value*
DFHSC2LS がアクティビティー・ログとトレース情報を書き込むファイルの、完全修飾された z/OS UNIX 名。DFHSC2LS はこのファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。

通常、このファイルを使用することはありませんが、DFHSC2LS で問題が発生した場合は、IBM サービス組織によって使用が指示される場合があります。

MAPPING-LEVEL={1.0|1.1|1.2|2.0|2.1|2.2|3.0}
XML バインディングおよび言語構造の生成時に支援機能で使用するマッピング・レベルを指定します。利用可能な最新のマッピング・レベルを使用することをお勧めします。DFHSC2LS の場合、マッピング・レベル 3.0 以上を使用することをお勧めします。

3.0 xsd:dateTime データ型は CICS ASKTIME フォーマットにマップされます。

MINIMUM-RUNTIME-LEVEL={MINIMUM|3.0|CURRENT}
XML バインディングをデプロイすることが可能な、CICS の最小ランタイム環境を指定します。指定した他のパラメーターと一致しないレベルを選択すると、エラー・メッセージが送信されます。選択可能なオプションは以下のとおりです。

MINIMUM
選択可能な CICS の最小ランタイム・レベルが、指定したパラメーターに基づいて自動的に割り振られます。

3.0 CICS XML 支援機能を使用して、高機能のデータ・マッピングを利用する場合は、ランタイム・レベル 3.0 以上を指定します。

CURRENT
生成された XML バインディングを、ランタイム環境が XML バインディングの生成に使用された領域と同じである CICS 領域にデプロイする場合、このランタイム・レベルを使用します。

PDSCP=*value*
区分データ・セット・メンバーで使用されるコード・ページを指定します。ここで、*value* は CCSID 番号または Java コード・ページ番号です。このパラメーターを指定しない場合は z/OS UNIX システム・サービスのコード・ページが使用されます。例えば、`PDSCP=037` を指定することができます。

PDSLIB=*value*
生成された高水準言語が含まれる区分データ・セットの名前を指定します。

PDSMEM=*value*
高水準言語構造が含まれる区分データ・セット・メンバーの名前を生成するために DFHSC2LS が使用する、1 から 6 文字の文字接頭部を指定します。

DFHSC2LS は、操作ごとに 1 つの区分データ・セット・メンバーを生成します。メンバー名は、接頭部に 2 桁の数値を付加することで生成されます。

SCHEMA=*value*
XML スキーマの読み込み元になるファイルの、完全修飾された z/OS UNIX 名。DFHSC2LS はこのファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。

XML スキーマまたは WSDL 文書のいずれかを、 DFHSC2LS への入力データとして使用できます。入力データのソースを指定するため、このパラメーターまたは **WSDL** パラメーターのいずれかを指定する必要があります。ただし、両方を同時に指定することはできません。

STRUCTURE=*data*
C および C++ での、最上位データ構造の名前。デフォルト値は **PDSMEM** パラメーターの値です。

TYPES=*value*
有効にするグローバル・タイプのローカル名のリストを定義します。*value* が ALL である場合は、すべてのグローバル・タイプが有効であることを示します。デフォルトでは、グローバル・タイプは有効化されていません。

WSDL=*value*
WSDL 文書の完全修飾された z/OS UNIX 名。

XML スキーマまたは WSDL 文書のいずれかを、 DFHSC2LS への入力データとして使用できます。入力データのソースを指定するため、このパラメーターまたは **SCHEMA** パラメーターのいずれかを指定する必要があります。ただし、両方を同時に指定することはできません。

XMLCP=LOCAL|UTF-8|EBCDIC-CP-US
XML バインディングを生成するために使用されるコード・ページを指定します。

LOCAL
この値はデフォルトです。XML がローカル・コード・ページを使用して生成され、XML スキーマでエンコード・タグが生成されないことを指定します。

UTF-8 XML が UTF-8 コード・ページを使用して生成されることを指定します。 XML スキーマでエンコード・タグが生成されます。このオプションを指定する場合、XML スキーマが異なるプラットフォーム間でコピーされるときにエンコードが正しく維持されるようにする必要があります。

EBCDIC-CP-US
XML が US EBCDIC コード・ページを使用して生成されることを指定します。 XML スキーマでエンコード・タグが生成されます。

XSDBIND=*value*
XML バインディングの完全修飾された z/OS UNIX 名。DFHSC2LS はこのファイルを作成しますが、存在していないディレクトリー構造を作成することはありません。ファイル拡張子は .xsdbind です。

CICS 支援機能用のマッピング・レベル

マッピングとは、言語構造と XML スキーマ間の情報の変換方法を指定する一連の規則です。現在使用可能で最も高度なマッピングを活用するには、CICS 支援機能の **MAPPING-LEVEL** パラメーターを最新レベルに設定することをお勧めします。

マッピングの各レベルでは、以前のマッピング機能が継承され、利用可能な最良の機能を提供する最高レベルのマッピングを使用できます。マッピングの最高レベルでは、実行時のデータ変換がより強力に制御され、特定のデータ型や XML エレメントへのサポートの制限が取り外されています。

以前のレベルで有効化されていたアプリケーションを再デプロイする場合は、**MAPPING-LEVEL** パラメーターをそのレベルに設定できます。

マッピング・レベル 3.0

マッピング・レベル 3.0 は、CICS TS 4.1 領域と互換性があります。

このマッピング・レベルでは、以下のサポートが提供されます。

- DFHSC2LS および DFHWS2LS は、`xsd:dateTime` データ型を CICS ASKTIME フォーマットにマップします。
- DFHLS2WS は、WSDL 文書および Web サービス・バインディングを、1 つのコンテナーのみでなく、多数のコンテナーを使用するアプリケーションから生成できます。

- 固定長データ構造で記述される切り捨てられたデータの許容。この動作は、CICS 支援機能で **DATA-TRUNCATION** パラメーターを使用することによって設定できます。

マッピング・レベル 2.2 以上

マッピング・レベル 2.2 以上は、APAR PK69738 以上が適用された CICS TS 3.2 領域と互換性があります。

マッピング・レベル 2.2 以上の場合、DFHSC2LS と DFHWS2LS は以下の XML マッピングをサポートします。

- エレメントの固定値
- 代替グループ
- 抽象データ型
- XML スキーマ <sequence> エレメントは、<choice> エレメント内にネストできます。

DFHSC2LS および DFHWS2LS は、以下の XML マッピングへの拡張サポートも提供します。

- 抽象エレメント
- XML スキーマ <choice> エレメント

マッピング・レベル 2.1 以上

マッピング・レベル 2.1 は、APAR PK59794 以上が適用された CICS TS 3.2 領域と互換性があります。

このマッピング・レベルでは、新しい **INLINE-MAXOCCURS-LIMIT** パラメーター、**CHAR-VARYING** パラメーターの新しい値を使用した変数内容の処理方法をより強力に制御できます。

マッピング・レベル 2.1 以上の場合、DFHSC2LS と DFHWS2LS は XML マッピングに対し、以下の新規サポートまたは強化されたサポートを提供します。

- XML スキーマ <any> エレメント
- `xsd:anyType` タイプ
- 抽象エレメントの許容
- **INLINE-MAXOCCURS-LIMIT** パラメーター
- `minOccurs` 属性

INLINE-MAXOCCURS-LIMIT パラメーターは、可変反復リストをインラインにマップするかどうかを指定します。インラインへの可変反復内容のマップの詳細については、479 ページの『エレメントの可変配列』を参照してください。

`minOccurs` 属性のサポートは、XML スキーマ <sequence> エレメント、 <choice> エレメントおよび <all> エレメントで拡張されています。`minOccurs="0"` の場合、CICS 支援機能はこれらのエレメントを、`minOccurs="0"` 属性もそのすべての子エレメントの属性であるかのように扱います。

マッピング・レベル 2.1 以上の場合、DFHLS2SC と DFHLS2WS は以下の XML マッピングをサポートします。

- COBOL および PL/I 内の FILLER フィールドは無視されます
- **CHAR-VARYING** パラメーターの COLLAPSE の値
- **CHAR-VARYING** パラメーターの BINARY の値

COBOL および PL/I の FILLER フィールドは無視され、生成された XML スキーマには表示されず、実行時に、データ構造に適切なギャップが残ります。

COLLAPSE により、CICS はテキスト・フィールドの末尾スペースを無視します。

BINARY は、バイナリー・フィールドへのサポートを提供します。この値は、COBOL を XML スキーマに変換する際に役立ちます。このオプションは SBCS 文字配列のみで使用可能です。このオプションにより、文字配列を `xsd:string` フィールドではなく、固定長の `xsd:base64Binary` フィールドにマップできます。

マッピング・レベル 1.2 以上

マッピング・レベル 1.2 は、CICS TS 3.1 領域以上と互換性があります。

実行時に、バッチ・ツールのこれらの追加パラメーターを使用して文字とバイナリー・データを変換する際、その方法をより強力に制御できます。

- **CHAR-VARYING**
- **CHAR-VARYING-LIMIT**
- **CHAR-MULTIPLIER**
- **DEFAULT-CHAR-MAXLENGTH**

DFHSC2LS または DFHWS2LS で **CHAR-MULTIPLIER** パラメーターを使用する場合は、このパラメーターの値が文字データに必要なスペース量の計算に使用された後、以下の規則が適用されることに注意してください。

- DFHSC2LS および DFHWS2LS はこれらのマッピングを提供します。
  - 最大長が 32 767 バイトより大きい可変長文字データ型がコンテナーにマップされます。**CHAR-VARYING-LIMIT** パラメーターを使用して、下限を設定できます。言語構造内に 16 バイトのフィールドが作成され、コンテナー名が保管されます。実行時に、文字データがコンテナーに保管され、コンテナー名が言語構造内に入れられます。
  - 最大長が 32 768 バイト未満の可変長文字データ型は、C/C++ および Enterprise PL/I 以外のすべての言語の VARYING 構造にマップされます。C/C++ の場合、これらのデータ型はヌル終了ストリングにマップされ、Enterprise PL/I では VARYINGZ 構造にマップされます。**CHAR-VARYING** パラメーターを使用して、可変長文字データのマッピング方法を選択できます。
  - 最大長が 32 768 バイト未満の可変長バイナリー・データ型は、すべての言語の VARYING 構造にマップされます。最大長が 32 768 バイト以上の場合、データはコンテナーにマップされます。言語構造内に 16 バイトのフィールドが作成され、コンテナー名が保管されます。実行時に、バイナリー・データがコンテナーに保管され、コンテナー名が言語構造内に入れられます。

これらに関連付けられた長さを持たない文字データ型が XML スキーマに存在する場合は、DFHWS2LS または DFHSC2LS の **DEFAULT-CHAR-MAXLENGTH** パラメーターを使用して、デフォルトの長さを割り当てることができます。

DFHLS2SC および DFHLS2WS はこれらのマッピングを提供します。

- 文字フィールドは `xsd:string` データ型にマップされ、実行時に固定長フィールドまたはヌル終了ストリングとして処理できます。PL/I 以外のすべての言語で、**CHAR-VARYING** パラメーターを使用して、実行時に可変長文字データを処理する方法を選択できます。
- データの最大長が 32 767 バイトより大きいか、または長さが定義されていない場合、Base64Binary データ型はコンテナーにマップします。データ長が 32 767 以下の場合、base64Binary データ型はすべての言語の VARYING 構造にマップされます。

マッピング・レベル 1.1 以上

マッピング・レベル 1.1 は、CICS TS 3.1 領域以上と互換性があります。

このマッピング・レベルでは、XML 文字とバイナリー・データ型のマッピングが強化されています(特に、XML スキーマで異なる値で定義された `maxLength` 属性と `minLength` 属性を持つ可変長のデータをマップする場合)。データは、以下のように扱われます。

- 固定長が 16 MB より大きい文字とバイナリー・データ型は、PL/I 以外のすべての言語のコンテナーにマップされます。PL/I の場合、32 767 バイトより大きい固定長の文字とバイナリー・データ型はコンテナーにマップされます。言語構造内に 16 バイトのフィールドが作成され、コンテナー名が保管されます。実行時に、固定長データがコンテナーに保管され、コンテナー名が言語構造内に入れられます。

  コンテナーの長さは可変的なので、コンテナーにマップされた固定長データと、XML スキーマや Web サービス記述で指定された固定長が一致するように、スペースやヌルが埋め込まれたり切り捨てられたりすることはありません。データ長が重要な場合、アプリケーションに書き込んでデータ長を確認するか、または CICS 領域で妥当性検査をオンにすることができます。 SOAP および XML の両方の妥当性検査は、パフォーマンスに大きな影響を及ぼします。
- XML スキーマ <list> データ型および <union> データ型は、文字フィールドにマップされます。
- スキーマ定義された XML 属性は、無視されずにマップされます。各 XML エレメントは、最大 255 の属性を持つことができます。詳しくは、484 ページの『XML 属性のサポート』を参照してください。
- `xsi:nil` 属性がサポートされています。詳しくは、484 ページの『XML 属性のサポート』を参照してください。

マッピング・レベル 1.1 のみ

マッピング・レベル 1.1 は、CICS TS 3.1 領域以上と互換性があります。

このマッピング・レベルでは、XML 文字とバイナリー・データ型のマッピングが強化されています(特に、XML スキーマで異なる値で定義された maxLength 属性と minLength 属性を持つ可変長のデータをマップする場合)。データは、以下のように扱われます。

- 可変長バイナリー・データ型は、コンテナーにマップされます。言語構造内に 16 バイトのフィールドが作成され、コンテナー名が保管されます。実行時に、バイナリー・データがコンテナーに保管され、コンテナー名が言語構造内に入れられます。
- 最大長が 32 767 バイトより大きい可変長文字データ型は、コンテナーにマップされます。言語構造内に 16 バイトのフィールドが作成され、コンテナー名が保管されます。実行時に、文字データがコンテナーに保管され、コンテナー名が言語構造内に入れられます。
- 固定長が 16 MB 未満の文字とバイナリー・データ型は、PL/I 以外のすべての言語の固定長フィールドにマップされます。PL/I の場合、32 767 バイト以下の固定長の文字とバイナリー・データ型は固定長フィールドにマップされます。
- CICS は、hexBinary 形式のデータのエンコードとデコードは行いますが、base64Binary 形式のデータのエンコードとデコードは行いません。XML スキーマの Base64Binary データ型は、言語構造内のフィールドにマップされます。フィールドのサイズは、式: 4×(ceil(*z*/3)) を使用して計算されます。ここで、
  - *z* は XML スキーマ内のデータ型の長さです。
  - ceil(*x*) は、*x* より大きいか等しい最小の整数です。

  z の長さが 24 566 バイトより大きい場合、結果として生成される言語構造はコンパイルに失敗します。24 566 バイトより大きい base64Binary データの場合、マッピング・レベル 1.2 を使用することをお勧めします。マッピング・レベル 1.2 では、言語構造内のフィールドを使用する代わりに、base64Binary データをコンテナーにマップできます。

マッピング・レベル 1.0 のみ

マッピング・レベル 1.0 は、CICS TS 3.1 領域以上と互換性があります。

以下の制限に注意してください。これらの制限は、1.0 より上のマッピング・レベルでは変更されています。

- DFHSC2LS および DFHWS2LS は、XML スキーマ内の文字およびバイナリー・データ型を、言語構造内の固定長フィールドにマップします。この部分的な XML スキーマを見てください。

```
<xsd:element name="example">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="33000"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

  この部分的な XML スキーマは、 COBOL 言語構造で次のように表示されます。

```
15 example     PIC X(33000)
```

- CICS は、hexBinary 形式のデータのエンコードとデコードは行いますが、base64Binary 形式のデータのエンコードとデコードは行いません。DFHSC2LS お

よび DFHWS2LS は、Base64Binary データを固定長文字フィールドにマップします。フィールドの内容は、アプリケーション・プログラムによってエンコードまたはデコードされる必要があります。

- DFHSC2LS および DFHWS2LS は処理中、XML 属性を無視します。
- DFHLS2SC および DFHLS2WS は、言語構造内の文字フィールドとバイナリー・フィールドを固定長フィールドとして解釈し、これらのフィールドを、maxLength 属性を持つ XML エレメントにマップします。実行時に、十分なデータが存在しない場合、言語構造内のフィールドはスペースまたはヌルで埋められます。

関連資料

484 ページの『XML 属性のサポート』
XML スキーマは、XML で許可されているか、または必要な属性を指定できます。CICS 支援機能ユーティリティーである DFHWS2LS および DFHSC2LS は XML 属性を無視するようにデフォルト設定されています。XML スキーマで定義されている XML 属性を処理するには、**MAPPING-LEVEL** パラメーターの値を 1.1 以上に設定する必要があります。

451 ページの『CICS 支援機能を使用する場合のデータ・マッピングの制限』
CICS では、高水準言語と XML スキーマや WSDL 文書 (WSDL バージョン 1.1 または 2.0 に準拠) との間の双方向データ・マッピングがサポートされますが、特定の制限があります。これらの制限は DFHWS2LS ツールと DFHSC2LS ツールのみに適用され、マッピング・レベルによって異なります。

高水準言語および XML スキーマのマッピング

CICS 支援機能は、高水準言語構造と XML スキーマまたは WSDL 文書との間のマッピングを生成する場合に使用します。また、CICS 支援機能は、高水準言語データ構造から XML スキーマまたは WSDL 文書を生成したり、 XML スキーマまたは WSDL 文書から高水準言語データ構造を生成したりします。

ユーティリティー・プログラムである DFHSC2LS および DFHLS2SC は、まとめて CICS XML 支援機能として知られています。ユーティリティー・プログラムである DFHWS2LS および DFHLS2WS はまとめて CICS Web サービス支援機能として知られています。

- DFHLS2SC と DFHLS2WS は、高水準言語構造をそれぞれ、XML スキーマおよび WSDL 文書にマップします。
- DFHSC2LS と DFHWS2LS は、XML スキーマと WSDL 文書を高水準言語構造にマップします。

2 つのマッピングはシンメトリックではありません。

- 言語データ構造を DFHLS2SC または DFHLS2WS で処理した後、生成された XML スキーマまたは WSDL 文書を補完的なユーティリティー・プログラム (それぞれ、DFHSC2LS または DFHWS2LS) で処理する場合、最終的なデータ構造が、処理開始時のデータ構造と同じになるとは想定しないでください。ただし、最終的なデータ構造は、論理上は処理開始時のデータ構造と同じです。
- XML スキーマまたは WSDL 文書を DFHSC2LS または DFHWS2LS で処理した後、生成された言語構造を補完的なユーティリティー・プログラム (それぞれ、

DFHLS2SC または DFHLS2WS) で処理する場合、最終的な XML スキーマ内の XML スキーマまたは WSDL 文書が、処理開始時のものと同じになるとは想定しないでください。

- 場合によっては、DFHSC2LS と DFHWS2LS は、DFHLS2SC と DFHLS2WS でサポートされない言語構造を生成することがあります。

DFHLS2SC と DFHLS2WS によって処理された言語構造は、 CICS によってサポートされる言語コンパイラーで実装されているように、言語の規則に従ってコード化する必要があります。

関連タスク

500 ページの『言語構造からのマッピングの生成』
XML をアプリケーション・データから作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータおよび XML を変換する方法を記述する必要があります。これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または DB2 レコードから開始できます。

CICS 支援機能を使用する場合のデータ・マッピングの制限

CICS では、高水準言語と XML スキーマや WSDL 文書 (WSDL バージョン 1.1 または 2.0 に準拠) との間の双方向データ・マッピングがサポートされますが、特定の制限があります。これらの制限は DFHWS2LS ツールと DFHSC2LS ツールのみに適用され、マッピング・レベルによって異なります。

全マッピング・レベルでの制限

- リテラル・エンコード方式を使用する SOAP バインディングのみがサポートされます。したがって、`use` 属性は `literal` の値に設定する必要があります。`use="encoded"` はサポートされていません。
- データ型定義は、XML スキーマ定義言語 (XSD) を使用してエンコードする必要があります。スキーマ内の SOAP メッセージで使用されたデータ型は、明示的に宣言する必要があります。
- Web サービス記述内の一部のキーワードの長さは制限されます。例えば、操作、バインディング、および部品の名前は、255 文字に制限されます。場合によっては、操作名の最大長はわずかに短くなることがあります。
- Web サービス記述で定義された SOAP 障害はすべて無視されます。サービス・プロバイダー・アプリケーションで SOAP 障害メッセージを送信する場合は、EXEC CICS SOAPFAULT コマンドを使用します。
- DFHWS2LS および DFHSC2LS では、特定の有効範囲内の単一の <xsd:any> エレメントのみがサポートされます。例えば、以下のスキーマ・フラグメントはサポートされません。

```
<xsd:sequence>
 <xsd:any/>
 <xsd:any/>
</xsd:sequence>
```

この `<xsd:any>` は、必要に応じて、`minOccurs` および `maxOccurs` を指定できます。例えば、以下のスキーマ・フラグメントがサポートされます。

```
<xsd:sequence>
 <xsd:any minOccurs="2" maxOccurs="2"/>
</xsd:sequence>
```

- 循環型参照はサポートされていません。例えば、タイプ A がタイプ B を含み、タイプ B がタイプ A を含む場合などです。
- グループ・エレメントでは、再帰はサポートされません (<xsd:choice> エレメント、<xsd:sequence> エレメント、 <xsd:group> エレメント、または <xsd:all> エレメントなど)。例えば、以下のスキーマ・フラグメントはサポートされません。

```
<xsd:choice maxOccurs="2">
   <xsd:element name="name1" type="string"/>
</xsd:choice>
```

  ただし、マッピング・レベル 2.1 以上である (`maxOccurs="1"` と `minOccurs="0"` がこれらのエレメントでサポートされている) 場合は例外です。
- DFHSC2LS および DFHWS2LS では、`xsi:type` 属性または代替グループのいずれかの XML スキーマで宣言されたデータ型とエレメントから派生した SOAP メッセージのデータ型とエレメントはサポートされません。ただし、マッピング・レベル 2.2 以上で、親エレメントまたはタイプが抽象として定義されている場合は例外です。
- <xsd:choice> エレメント内の組み込み <xsd:sequence> エレメントおよび <xsd:group> エレメントは、マッピング・レベル 2.2 より前のレベルではサポートされません。<xsd:choice> エレメント内の組み込み<xsd:choice> エレメントおよび <xsd:all> エレメントはサポートされません。

マッピング・レベル 1.1 以上で強化されたサポート

マッピング・レベルが 1.1 以上のとき、DFHWS2LS は以下の XML エレメントとエレメント・タイプをサポートします。

- <xsd:list> エレメント。
- <xsd:union> エレメント。
- `xsd:anySimpleType` タイプ。
- <xsd:attribute> エレメント。マッピング・レベル 1.0 ではこのエレメントは無視されます。

マッピング・レベル 2.1 以上で強化されたサポート

マッピング・レベルが 2.1 以上のとき、DFHWS2LS は以下の XML エレメントとエレメント属性をサポートします。

- <xsd:any> エレメント。
- `xsd:anyType` タイプ。
- 抽象エレメント。以前のマッピング・レベルの抽象エレメントは、継承の階層での非端末タイプとしてのみサポートされます。
- <xsd:all> エレメント、<xsd:choice> エレメント、および <xsd:sequence> エレメント上の `maxOccurs` 属性および `minOccurs` 属性 (`maxOccurs="1"` および `minOccurs="0"` の場合のみ)
- COBOL 内の「FILLER」フィールド、および PL/I 内の「*」フィールドは抑制されます。これらのフィールドは、生成された WSDL には表示されず、実行時に、データ構造に適切なギャップが保持されます。

マッピング・レベル 2.2 以上で強化されたサポート

マッピング・レベルが 2.2 以上の場合、 DFHSC2LS と DFHWS2LS による <xsd:choice> エレメントのサポートが強化され、最大 255 個の <xsd:choice> エレメントのオプションがサポートされます。 <xsd:choice> サポートの詳細については、 489 ページの『<xsd:choice>のサポート』を参照してください。

マッピング・レベル 2.2 以上では、CICS 支援機能により、以下の XML マッピングがサポートされます。

- 代替グループ
- エレメントの固定値
- 抽象データ型

<xsd:choice> エレメント内の組み込み <xsd:sequence> エレメントおよび <xsd:group> エレメントは、マッピング・レベル 2.2 以上でサポートされます。例えば、以下のスキーマ・フラグメントがサポートされます。

```
<xsd:choice>
   <xsd:element name="name1" type="string"/>
   <xsd:sequence/>
</xsd:choice>
```

SOAP メッセージの親エレメントやタイプが抽象として定義されている場合、DFHSC2LS および DFHWS2LS は、XML スキーマで宣言されたデータ型とエレメントから派生したデータ型とエレメントをサポートします。

マッピング・レベル 3.0 以上で強化されたサポート

マッピング・レベルが 3.0 以上の場合、CICS 支援機能により以下のマッピングの改良がサポートされています。

- DFHSC2LS および DFHWS2LS は、`xsd:dateTime` データ型を CICS ASKTIME フォーマットにマップします。
- DFHLS2WS は、WSDL 文書および Web サービス・バインディングを、1 つのコンテナーのみでなく、多数のコンテナーを使用するアプリケーションから生成できます。
- 固定長データ構造で記述される切り捨てられたデータの許容。この動作は、CICS 支援機能で **DATA-TRUNCATION** パラメーターを使用することによって設定できます。

  関連資料

  445 ページの『CICS 支援機能用のマッピング・レベル』
  マッピングとは、言語構造と XML スキーマ間の情報の変換方法を指定する一連の規則です。現在使用可能で最も高度なマッピングを活用するには、CICS 支援機能の **MAPPING-LEVEL** パラメーターを最新レベルに設定することをお勧めします。

COBOL から XML スキーマへのマッピング

DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは、 COBOL データ構造体と XML スキーマ定義間のマッピングをサポートします。

COBOL の名前は、以下の規則に従って、XML の名前に変換されます。

1. 重複する名前は、1 つ以上の数値の桁が付加されて、固有の名前にされます。

   例えば、year の 2 つのインスタンスは year および year1 になります。
2. ハイフンは、下線文字に置き換わります。連続するハイフンのストリングは、連続する下線に置き換わります。

   例えば、current-user--id は current_user__id となります。
3. ハイフンで区切られ、大文字のみを含む名前のセグメントは、小文字に変換されます。

   例えば、CA-REQUEST-ID は ca_request_id となります。
4. 数値で始まる名前には、先行する下線文字が追加されます。

   例えば、9A-REQUEST-ID は _9a_request_id となります。

CICS は以下の表に従って、COBOL データ記述エレメントをスキーマ・エレメントにマップします。表に示されていない COBOL データ記述エレメントは、DFHLS2SC または DFHLS2WS ではサポートされません。以下の制限も適用されます。

- レベル番号が 66 から 77 のデータ記述項目はサポートされません。レベル番号が 88 のデータ記述項目は無視されます。
- データ記述項目の以下の節はサポートされません。

  OCCURS DEPENDING ON

  OCCURS INDEXED BY

  REDEFINES

  RENAMES。これはレベル 66 です。

  DATE FORMAT
- データ記述項目の以下の節は無視されます。

  BLANK WHEN ZERO

  JUSTIFIED

  VALUE
- SIGN 節の SIGN TRAILING はサポートされています。SIGN 節の SIGN LEADING は、DFHLS2SC または DFHLS2WS で指定されたマッピング・レベルが 1.2 以上の場合のみ、サポートされます。
- SEPARATE CHARACTER は、SIGN TRAILING 節と SIGN LEADING 節の両方で、マッピング・レベルが 1.2以上の場合にサポートされます。
- 以下の USAGE 節の句はサポートされていません。

  OBJECT REFERENCE

  POINTER

  FUNCTION-POINTER

  PROCEDURE-POINTER
- 以下の USAGE 節の句は、マッピング・レベル 1.2 以上でサポートされます。

  COMPUTATIONAL-1

  COMPUTATIONAL-2

- DISPLAY および COMPUTATIONAL-5 データ記述項目でサポートされている PICTURE 文字は、9、S、および Z のみです。
- PACKED-DECIMAL データ記述項目でサポートされている PICTURE 文字は、9、S、V、および Z です。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが NULL に設定されている場合、文字配列は xsd:string にマップされ、ヌル終了ストリングとして処理されます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが BINARY に設定されている場合、文字配列は xsd:base64Binary にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが COLLAPSE に設定されている場合、ストリングの末尾空白文字は無視されます。

COBOL データ記述	スキーマ simpleType
PIC X(*n*) PIC A(*n*) PIC G(*n*) DISPLAY-1 PIC N(*n*)	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxlength value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> ここで、*m*=*n*
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY	<xsd:simpleType> <xsd:restriction base="xsd:short"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、「9」の文字のパターンによって表すことができる最大値です。
PIC S9(*z*) DISPLAY ここで、 5 ≤ *z* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:int"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、「9」の文字のパターンによって表すことができる最大値です。
PIC S9(*z*) DISPLAY ここで、 9 < *z*	<xsd:simpleType> <xsd:restriction base="xsd:long"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、「9」の文字のパターンによって表すことができる最大値です。

COBOL データ記述	スキーマ simpleType
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、「9」の文字のパターンによって表すことができる最大値です。
PIC 9(*z*) DISPLAY ここで、 5 ≤ *z* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、「9」の文字のパターンによって表すことができる最大値です。
PIC 9(*z*) DISPLAY ここで、 9 < *z*	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、「9」の文字のパターンによって表すことができる最大値です。
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY ここで、*n* ≤ 4。	<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY ここで、5 ≤ *n* ≤ 9。	<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>
PIC S9(*n*) COMP PIC S9(*n*) COMP-4 PIC S9(*n*) COMP-5 PIC S9(*n*) BINARY ここで、9 <*n*。	<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY ここで、*n* ≤ 4。	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY ここで、5 ≤ *n* ≤ 9。	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>

COBOL データ記述	スキーマ simpleType
PIC 9(*n*) COMP PIC 9(*n*) COMP-4 PIC 9(*n*) COMP-5 PIC 9(*n*) BINARY ここで、9 <*n*。	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>
PIC S9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*p* = *m* + *n*。
PIC 9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> <xsd:minInclusive value="0"/> </xsd:restriction> </xsd:simpleType> ここで、*p* = *m* + *n*。
PIC S9(*m*) COMP-3 DATETIME=PACKED15 の場合にマッピング・レベル 3.0 でサポート	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> タイム・スタンプの形式は CICS ABSTIME です。
PIC S9(*m*)V9(*n*) DISPLAY マッピング・レベル 1.2 以上でサポート	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*p* = *m* + *n*。
COMP-1 マッピング・レベル 1.2 以上でサポート	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
COMP-2 マッピング・レベル 1.2 以上でサポート	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

関連資料

458 ページの『XML スキーマと COBOL とのマッピング』
DFHSC2LS および DFHWS2LS ユーティリティー・プログラムでは、 XML スキーマ定義と COBOL データ構造との間のマッピングをサポートしています。

494 ページの『COBOL での可変反復内容の処理方法』
COBOL では、ポインター演算を使用してデータの各インスタンスをアドレス指定し、可変反復内容を処理することはできません。他のプログラミング言語には、このような制限はありません。この例では、COBOL で Web サービス・アプリケーション用の可変反復内容を処理する方法を示します。

XML スキーマと COBOL とのマッピング

DFHSC2LS および DFHWS2LS ユーティリティー・プログラムでは、 XML スキーマ定義と COBOL データ構造との間のマッピングをサポートしています。

CICS 支援機能は以下の規則を使用して、スキーマのエレメント名から COBOL 変数の固有かつ有効な名前を生成します。

1. COBOL 予約語には接頭部 'X' が付きます。

   例えば、DISPLAY は XDISPLAY になります。
2. A から Z、a から z、0 から 9、またはハイフンは、'X' に置き換えられます。

   例えば、monthly_total は monthlyXtotal になります。
3. 最後の文字がハイフンの場合は、'X' に置き換えられます。

   例えば、ca-request- は ca-requestX になります。
4. 変数の基数が可変であることがスキーマで指定されている (つまり、minOccurs および maxOccurs が xsd:element で異なる値によって指定されている) 場合、23 文字より長いスキーマのエレメント名は、23 文字に切り捨てられます。

   変数の基数が固定であることがスキーマで指定されている場合、28 文字より長いスキーマのエレメント名は、28 文字に切り捨てられます。
5. 同じスコープ内で重複する名前は、1 桁または 2 桁の数値が名前の 2 番目以降のインスタンスに付加されて固有の名前になります。

   例えば、year の 3 つのインスタンスは year、year1、 および year2 になります。
6. -cont または -num ストリング用に 5 つの文字が予約されており、変数の基数が可変であることがスキーマで指定されている (つまり、minOccurs と maxOccurs が異なる値で指定されている) 場合に使用されます。

   詳しくは、 479 ページの『エレメントの可変配列』を参照してください。
7. 属性の場合、以前の規則がエレメント名に適用されます。エレメント名に接頭部 attr- が追加され、その後に -value または -exist が続きます。合計長が 28 文字より長い場合、エレメント名は切り捨てられます。詳しくは、 484 ページの『XML 属性のサポート』を参照してください。

   nillable 属性には特別な規則が適用されます。接頭部 attr- が追加されますが、nil- もエレメント名の最初に追加されます。エレメント名の後には -value が続きます。合計長が 28 文字より長い場合、エレメント名は切り捨てられます。

最終的な名前の合計長は 30 文字以下になります。

DFHSC2LS および DFHWS2LS は以下の表に従い、指定されたマッピング・レベルを使用して、スキーマ型を COBOL データ記述エレメントにマップします。以下のことに注意してください。

- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが NULL に設定されている場合、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符については追加の文字が割り振られます。

- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが YES に設定されている場合、可変長文字データは関連する 2 つのエレメント (長さフィールドおよびデータ・フィールド) にマップされます。例:

```
<xsd:simpleType name="VariableStringType">
   <xsd:restriction base="xsd:string">
       <xsd:minLength value="1"/>
       <xsd:maxLength value="10000"/>
   </xsd:restriction>
</xsd:simpleType>
<xsd:element name="textString" type="tns:VariableStringType"/>
```

これは次のようにマップされます。

```
15 textString-length PIC S9999 COMP-5 SYNC
15 textString        PIC X(10000)
```

スキーマ単純型	COBOL データ記述
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 2.0 以下: サポート対象外 マッピング・レベル 2.1: サポート対象
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1 以上: PIC X(255)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> <xsd:length value="*z*"/> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY hexBinary	すべてのマッピング・レベル: PIC X(*z*)

スキーマ単純型	COBOL データ記述
<xsd:simpleType> <xsd:restriction base="xsd:*type*" </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 duration date time gDay gMonth gYear gMonthDay gYearMonth	すべてのマッピング・レベル: PIC X(32)
<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.2 以下: PIC X(32) マッピング・レベル 2.0 以上: PIC X(40) マッピング・レベル 3.0 以上: PIC S9(15) COMP-3 形式は CICS ABSTIME です。
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 byte unsignedByte	すべてのマッピング・レベル: PIC X DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9999 COMP-5 SYNC または PIC S9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC 9999 COMP-5 SYNC または PIC 9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9(18) COMP-3
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9(9) COMP-5 SYNC または PIC S9(9) DISPLAY

スキーマ単純型	COBOL データ記述
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC 9(9) COMP-5 SYNC または PIC 9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC S9(18) COMP-5 SYNC または PIC S9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC 9(18) COMP-5 SYNC または PIC 9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*m*" <xsd:fractionDigits value="*n*" </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC 9(*p*)V9(*n*) COMP-3 ここで、*p* = *m* - *n* です。
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: PIC X DISPLAY
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1 以上: PIC X(255)
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1 以上: PIC X(255)
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="*z*"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> </xsd:restriction> </xsd:simpleType> この場合、長さは定義されていません。	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1: PIC X(*y*) ここで、*y* =4×(ceil(*z*/3)) です。 ceil(*x*) は、*x* より大きいか等しい最小の整数です。 マッピング・レベル 1.2 以上: PIC X(*z*) この場合、長さは固定されています。 PIC X(16) この場合、長さは定義されていません。このフィールドは、2 進データが保管されるコンテナーの 16 バイトの名前を保持します。

スキーマ単純型	COBOL データ記述
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.1 以下: PIC X(32) マッピング・レベル 1.2 以上: COMP-1
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.1 以下: PIC X(32) マッピング・レベル 1.2 以上: COMP-2

表で示されているスキーマ・タイプの一部は、minInclusive ファセットおよび maxInclusive ファセットに指定された値 (存在する場合) に応じて、COMP-5 SYNC または DISPLAY の COBOL フォーマットにマップされます。

- 符号あり型 (short、int、および long) の場合は、以下が指定されているときに DISPLAY が使用されます。

  ```
  <xsd:minInclusive value="-a"/>
  <xsd:maxInclusive value="a"/>
  ```

  ここで、*a* は '9' で構成されるストリングです。
- 符号なし型 (unsignedShort、unsignedInt、および unsignedLong) の場合は、以下が指定されているときに DISPLAY が使用されます。

  ```
  <xsd:minInclusive value="0"/>
  <xsd:maxInclusive value="a"/>
  ```

  ここで、*a* は '9' で構成されるストリングです。

他の値が指定されている場合や値が指定されていない場合は、COMP-5 SYNC が使用されます。

関連資料

453 ページの『COBOL から XML スキーマへのマッピング』
DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは、 COBOL データ構造体と XML スキーマ定義間のマッピングをサポートします。

494 ページの『COBOL での可変反復内容の処理方法』
COBOL では、ポインター演算を使用してデータの各インスタンスをアドレス指定し、可変反復内容を処理することはできません。他のプログラミング言語には、このような制限はありません。この例では、COBOL で Web サービス・アプリケーション用の可変反復内容を処理する方法を示します。

486 ページの『<xsd:any> と xsd:anyType のサポート』
DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

C および C++ から XML スキーマへのマッピング

DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは、 C および C++ データ型と、XML スキーマ定義間のマッピングをサポートします。

C および C++ の名前は、以下の規則に従って、XML の名前に変換されます。

1. XML エレメント名で無効な文字は、「X」に置き換えられます。

   例えば、monthly-total は monthlyXtotal になります。
2. 重複する名前は、1 つ以上の数値の桁が付加されて、固有の名前にされます。

   例えば、year の 2 つのインスタンスは year および year1 になります。

DFHLS2SC および DFHLS2WS は以下の表に従って、C および C++ データ型をスキーマ・エレメントにマップします。表に示されていない C および C++ 型は、DFHLS2SC または DFHLS2WS ではサポートされません。以下の制限も適用されます。

- ヘッダー・ファイルには、最上位の struct インスタンスを含める必要があります。
- 自身をメンバーとして含める構造化型を宣言することはできません。
- 以下の C および C++ データ型はサポートされていません。

  ```
  decimal
  long double
  wchar_t (C++ のみ)
  ```
- 以下は、ヘッダー・ファイル内に存在する場合、無視されます。

  ストレージ・クラス指定子

  ```
  auto
  register
  static
  extern
  mutable
  ```

  修飾子

  ```
  const
  volatile
  _Export (C++ のみ)
  _Packed (C のみ)
  ```

  関数指定子

  ```
  inline (C++ のみ)
  virtual (C++ のみ)
  ```

  初期値
- ヘッダー・ファイルに以下の項目を含めることはできません。

  和集合

  クラス宣言

  列挙型データ型

  ポインター型変数

  テンプレート宣言

事前定義マクロ。つまり、2 つの下線文字 (__) で開始および終了する名前のマクロ

線継続シーケンス (改行文字がすぐ後に続く ¥ 記号)

プロトタイプ関数宣言子

プリプロセッサー・ディレクティブ

ビット・フィールド

__cdecl (または _cdecl) キーワード (C++ のみ)

- アプリケーション・プログラマーは、32 ビット・コンパイラーを使用して、int が必ず 4 バイトにマップするようにする必要があります。
- 以下の C++ 予約キーワードはサポートされていません。

  explicit

  using

  namespace

  typename

  typeid

- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが NULL に設定されている場合、文字配列は xsd:string にマップされ、ヌル終了ストリングとして処理されます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが BINARY に設定されている場合、文字配列は xsd:base64Binary にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが COLLAPSE に設定されている場合、<xsd:whiteSpace value="collapse"/> がストリング用に生成されます。

C および C++ データ型	スキーマ simpleType
char[*z*]	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="*z*"/> </xsd:restriction> </xsd:simpletype>
char[8] DATETIME=PACKED15 の場合にマッピング・レベル 3.0 以上でサポート	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> タイム・スタンプの形式は CICS ABSTIME です。
char	<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpletype>
unsigned char	<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpletype>
short	<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpletype>

C および C++ データ型	スキーマ simpleType
unsigned short	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpletype>
int long	<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpletype>
unsigned int unsigned long	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpletype>
long long	<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpletype>
unsigned long long	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpletype>
bool (C++ のみ)	<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpletype>
float マッピング・レベル 1.2 以上でサポート	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
double マッピング・レベル 1.2 以上でサポート	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

関連資料

『XML スキーマと C および C++ とのマッピング』
DFHSC2LS および DFHWS2LS ユーティリティー・プログラムは、各 Web サービス記述に含まれる XML スキーマ定義と C および C++ データ型との間のマッピングをサポートしています。

XML スキーマと C および C++ とのマッピング

DFHSC2LS および DFHWS2LS ユーティリティー・プログラムは、各 Web サービス記述に含まれる XML スキーマ定義と C および C++ データ型との間のマッピングをサポートしています。

CICS 支援機能は以下の規則を使用して、スキーマのエレメント名から C 変数および C++ 変数の固有かつ有効な名前を生成します。

1. A から Z、a から z、0 から 9、または _ 以外の文字は、'X' に置き換えられます。

   例えば、monthly-total は monthlyXtotal になります。
2. 最初の文字が英字でない場合、先行する 'X' に置き換えられます。

例えば、_monthlysummary は Xmonthlysummary になります。

3. スキーマのエレメント名が 50 文字より長い場合は、50 文字に切り捨てられます。
4. 同じスコープ内で重複する名前は、1 つ以上の桁の数値が付加されて固有の名前になります。

   例えば、year の 2 つのインスタンスは year および year1 になります。
5. _cont または _num ストリング用に 5 種類の文字が予約されており、変数の基数が可変であることがスキーマで指定されている (つまり、 minOccurs と maxOccurs が xsd:element で指定されている) 場合に使用されます。

   詳しくは、 479 ページの『エレメントの可変配列』を参照してください。
6. 属性の場合、以前の規則がエレメント名に適用されます。エレメント名に接頭部 attr_ が追加され、その後に _value または _exist が続きます。合計長が 28 文字より長い場合、エレメント名は切り捨てられます。

   nillable 属性には特別な規則が適用されます。接頭部 attr_ が追加されますが、nil_ もエレメント名の最初に追加されます。エレメント名の後には _value が続きます。合計長が 28 文字より長い場合、エレメント名は切り捨てられます。

最終的な名前の合計長は 57 文字以下になります。

DFHSC2LS および DFHWS2LS は以下の表に従って、スキーマ型を C および C++ データ型にマップします。以下の規則も適用されます。

- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが NULL に設定されている場合、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符については追加の文字が割り振られます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが YES に設定されている場合、可変長文字データは関連する 2 つのエレメント (長さフィールドおよびデータ・フィールド) にマップされます。

スキーマ simpleType	C および C++ データ型
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 2.0 以下: サポート対象外 マッピング・レベル 2.1 以上: サポート対象
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1 以上: char[255]

スキーマ simpleType	C および C++ データ型
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> <xsd:length value="*z*"/> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY hexBinary	すべてのマッピング・レベル: char[*z*]
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 duration date decimal time gDay gMonth gYear gMonthDay gYearMonth	すべてのマッピング・レベル: char[32]
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.2 以下: char[32] マッピング・レベル 2.0 以上: char[40] マッピング・レベル 3.0 以上: char[8] タイム・スタンプの形式は CICS ABSTIME です。
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: signed char

スキーマ simpleType	C および C++ データ型
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: char
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: short
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: unsigned short
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: char[33]
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: int
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: unsigned int
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: long long
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: unsigned long long
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpletype>	すべてのマッピング・レベル: bool (C++ のみ) short (C のみ)
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1 以上: char[255]
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1 以上: char[255]

スキーマ simpleType	C および C++ データ型
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="z"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64binary"> </xsd:restriction> </xsd:simpletype> この場合、長さは定義されていません。	マッピング・レベル 1.1 以下: char[*y*] ここで、*y* =4×(ceil(*z*/3)) です。ceil(*x*) は、*x* より大きいか等しい最小の整数です。 マッピング・レベル 1.2 以上: char[*z*] この場合、長さは固定されています。 char[16] は、長さが定義されていない場合に 2 進データが保管されるコンテナーの名前です。
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.1 以下: char[32] マッピング・レベル 1.2 以上: float(*)
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.0 以下: char[32] マッピング・レベル 1.2 以上: double(*)

関連資料

463 ページの『C および C++ から XML スキーマへのマッピング』
DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは、 C および C++ データ型と、XML スキーマ定義間のマッピングをサポートします。

486 ページの『<xsd:any> と xsd:anyType のサポート』
DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

PL/I から XML スキーマへのマッピング

DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは、 PL/I データ構造と XML スキーマ定義との間のマッピングをサポートします。Enterprise PL/I コンパイラーと以前の PL/I コンパイラーには相違があるため、 PLI-ENTERPRISE および PLI-OTHER の 2 種類の言語オプションがサポートされています。

PL/I の名前は、以下の規則に従って、XML の名前に変換されます。

1. XML エレメント名で無効な文字は、「xに置き換えられます。

   例えば、monthly$total は monthlyxtotal となります。
2. 重複する名前は、1 つ以上の数値の桁が付加されて、固有の名前にされます。

   例えば、year の 2 つのインスタンスは year および year1 になります。

DFHLS2SC および DFHLS2WS は以下の表に従って、PL/I データ型をスキーマ・エレメントにマップします。表に示されていない PL/I 型は、DFHLS2SC または DFHLS2WS ではサポートされません。以下の制限も適用されます。

- COMPLEX 属性を持つデータ項目はサポートされていません。
- FLOAT 属性を持つデータ項目は、マッピング・レベル 1.2 以上でサポートされます。Enterprise PL/I FLOAT IEEE はサポートされていません。
- VARYING および VARYINGZ ピュア DBCS ストリングは、マッピング・レベル 1.2 以上でサポートされます。
- DECIMAL(*p*,*q*) として指定されたデータ項目は、$p \geq q$ の場合にのみサポートされます。
- BINARY(*p*,*q*) として指定されたデータ項目は、$q = 0$ の場合にのみサポートされます。
- データ項目に PRECISION 属性が指定されている場合、そのデータ項目は無視されます。
- PICTURE ストリングはサポートされていません。
- ORDINAL データ項目は、FIXED BINARY(7) データ型として扱われます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが NULL に設定されている場合、文字配列は `xsd:string` にマップされ、ヌル終了ストリングとして処理されます。このマッピングは、Enterprise PL/I には適用されません。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが BINARY に設定されている場合、文字配列は `xsd:base64Binary` にマップされ、バイナリー・データとして処理されます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが COLLAPSE に設定されている場合、`<xsd:whiteSpace value="collapse"/>` がストリング用に生成されます。

DFHLS2SC および DFHLS2WS には、PL/I の埋め込みアルゴリズムが完全実装されていません。したがって、埋め込みバイトをデータ構造で明示的に宣言する必要があります。DFHLS2SC および DFHLS2WS は、埋め込みバイトが欠落していることを検出すると、メッセージを発行します。各最上位構造は、ダブルワード境界で開始する必要があり、構造内の各バイトは正しい境界にマップする必要があります。このコード・フラグメントについて考えます。

```
 3 FIELD1 FIXED BINARY(7),
 3 FIELD2 FIXED BINARY(31),
 3 FIELD3 FIXED BINARY(63);
```

この例では、次のようになります。

- FIELD1 は 1 バイト長で、すべての境界で位置合わせできます。
- FIELD2 は 4 バイト長で、フルワード境界で位置合わせする必要があります。
- FIELD3 は 8 バイト長で、ダブルワード境界で位置合わせする必要があります。

Enterprise PL/I コンパイラーは、以下の順序でフィールドを位置合わせします。

1. FIELD3 には、最も厳しい境界要件があるため、最初に位置合わせされます。
2. FIELD2 は、FIELD3 の直前に、フルワード境界で位置合わせされます。
3. FIELD1 は、FIELD3 の直前に、バイト境界で位置合わせされます。

最後に、コンパイラーは、構造全体がフルワード境界で位置合わせされるように、FIELD1 の直前に 3 つの埋め込みバイトを挿入します。

DFHLS2WS は同等の埋め込みバイトを挿入しないため、構造が DFHLS2WS によって処理される前に、埋め込みバイトを明示的に宣言する必要があります。例:

```
3 PAD1   FIXED BINARY(7),
3 PAD2   FIXED BINARY(7),
3 PAD3   FIXED BINARY(7),
3 FIELD1 FIXED BINARY(7),
3 FIELD2 FIXED BINARY(31),
3 FIELD3 FIXED BINARY(63);
```

または、構造を変更して全フィールドを位置合わせされないものとして宣言し、構造を使用するアプリケーションを再コンパイルすることができます。PL/I 構造の記憶域位置合わせの要件の詳細については、「*Enterprise PL/I 言語解説書*」を参照してください。

PL/I データ記述	スキーマ
FIXED BINARY (*n*) ここで、*n* ≤ 7	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
FIXED BINARY (*n*) ここで、 8 ≤ *n* ≤ 15	<xsd:simpleType> <xsd:restriction base="xsd:short"/> </xsd:simpleType>
FIXED BINARY (*n*) ここで、 16 ≤ *n* ≤ 31	<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>
FIXED BINARY (*n*) ここで、 32 ≤ *n* ≤ 63 **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) ここで、*n* ≤ 8 **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) ここで、 9 ≤ *n* ≤ 16 **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) ここで、 17 ≤ *n* ≤ 32 **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) ここで、33 ≤ *n* ≤ 64 **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"/> </xsd:simpleType>
FIXED DECIMAL(*n*,*m*)	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>
FIXED DECIMAL(15) DATETIME=PACKED15 の場合にマッピング・レベル 3.0 以上でサポート	<xsd:simpleType> <xsd:restriction base="xsd:dateTime" </xsd:restriction> </xsd:simpleType> タイム・スタンプの形式は CICS ABSTIME です。

PL/I データ記述	スキーマ
BIT(*n*) ここで、*n* は 8 の倍数です。他の値はサポートされません。	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> ここで、*m* = *n*/8
CHARACTER(*n*) VARYING および VARYINGZ は、マッピング・レベル 1.2 以上でもサポートされます。 **制約事項:** VARYINGZ は、Enterprise PL/I のみでサポートされます。	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>
GRAPHIC(*n*) VARYING および VARYINGZ は、マッピング・レベル 1.2 以上でもサポートされます。 **制約事項:** VARYINGZ は、Enterprise PL/I のみでサポートされます。	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> マッピング・レベル 1.0 および 1.1 で、 *m* = 2**n* の場合 <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> マッピング・レベル 1.2 以上の場合
WIDECHAR(*n*) **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> マッピング・レベル 1.0 および 1.1 で、 *m* = 2**n* の場合 <xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*n*"/> </xsd:restriction> </xsd:simpleType> マッピング・レベル 1.2 以上の場合
ORDINAL **制約事項:** Enterprise PL/I のみ	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
BINARY FLOAT(*n*) ここで、 *n* <= 21 マッピング・レベル 1.2 以上でサポートされます。	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>

PL/I データ記述	スキーマ
BINARY FLOAT(*n*) ここで、21 < *n* <= 53 53 より大きい値はサポートされません。 マッピング・レベル 1.2 以上でサポートされます。	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>
DECIMAL FLOAT(*n*) ここで、 *n* <= 6 マッピング・レベル 1.2 以上でサポートされます。	<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>
DECIMAL FLOAT(*n*) ここで、6 < *n* <= 16 16 より大きい値はサポートされません。 マッピング・レベル 1.2 以上でサポートされます。	<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>

関連資料

『XML スキーマと PL/I とのマッピング』
DFHSC2LS および DFHWS2LS ユーティリティー・プログラムでは、 XML スキーマ定義と PL/I データ構造との間のマッピングをサポートしています。 Enterprise PL/I コンパイラーと以前の PL/I コンパイラーには相違があるため、 PLI-ENTERPRISE および PLI-OTHER の 2 種類の言語オプションがサポートされています。

XML スキーマと PL/I とのマッピング

DFHSC2LS および DFHWS2LS ユーティリティー・プログラムでは、 XML スキーマ定義と PL/I データ構造との間のマッピングをサポートしています。Enterprise PL/I コンパイラーと以前の PL/I コンパイラーには相違があるため、 PLI-ENTERPRISE および PLI-OTHER の 2 種類の言語オプションがサポートされています。

CICS 支援機能は以下の規則を使用して、スキーマのエレメント名から PL/I 変数の固有かつ有効な名前を生成します。

1. A から Z、a から z、0 から 9、@、#、または $ 以外の文字は、'X' に置き換えられます。

   例えば、monthly-total は monthlyXtotal になります。
2. 変数の基数が可変であることがスキーマで指定されている (つまり、minOccurs 属性および maxOccurs 属性が xsd:element で異なる値によって指定されている) 場合、24 文字より長いスキーマのエレメント名は、24 文字に切り捨てられます。

   変数の基数が固定であることがスキーマで指定されている場合、29 文字より長いスキーマのエレメント名は、29 文字に切り捨てられます。

3. 同じスコープ内で重複する名前は、1 つ以上の桁の数値が名前の 2 番目以降のインスタンスに付加されて固有の名前になります。

   例えば、year の 3 つのインスタンスは year、year1、 および year2 になります。

4. _cont または _num ストリング用に 5 つの文字が予約されており、変数の基数が可変であることがスキーマで指定されている (つまり、minOccurs 属性と maxOccurs 属性が異なる値で指定されている) 場合に使用されます。

   詳しくは、479 ページの『エレメントの可変配列』を参照してください。

5. 属性の場合、以前の規則がエレメント名に適用されます。エレメント名に接頭部 attr- が追加され、その後に -value または -exist が続きます。合計長が 28 文字より長い場合、エレメント名は切り捨てられます。詳しくは、484 ページの『XML 属性のサポート』を参照してください。

   nillable 属性には特別な規則が適用されます。接頭部 attr- が追加されますが、nil- もエレメント名の最初に追加されます。エレメント名の後には -value が続きます。合計長が 28 文字より長い場合、エレメント名は切り捨てられます。

最終的な名前の合計長は 31 文字以下になります。

DFHSC2LS および DFHWS2LS は以下の表に従って、スキーマ型を PL/I データ型にマップします。以下のことにも注意してください。

- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが NULL に設定されている場合、可変長文字データはヌル終了ストリングにマップされ、ヌル終止符については追加の文字が割り振られます。
- **MAPPING-LEVEL** パラメーターが 1.2 以上に設定され、**CHAR-VARYING** パラメーターが指定されていない場合、デフォルトでは可変長文字データは Enterprise PL/I の場合に VARYINGZ データ型に、その他の PL/I の場合は VARYING データ型にマップされます。
- 可変長 2 進データは、32 768 バイトより小さい場合は VARYING データ型に、32 768 バイトより大きい場合はコンテナーにマップされます。

スキーマ	PL/I データ記述
<xsd:simpleType> <xsd:restriction base="xsd:anyType"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 2.0 以下: サポート対象外 マッピング・レベル 2.1 以上: サポート対象
<xsd:simpleType> <xsd:restriction base="xsd:anySimpletype"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以上:CHAR(255)

スキーマ	PL/I データ記述
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> <xsd:maxLength value="*z*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY	すべてのマッピング・レベル:CHARACTER(*z*)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 duration date time gDay gMonth gYear gMonthDay gYearMonth	すべてのマッピング・レベル:CHAR(32)
<xsd:simpleType> <xsd:restriction base="xsd:dateTime"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.2 以下: CHAR(32) マッピング・レベル 2.0 以上: CHAR(40) マッピング・レベル 3.0 以上: FIXED DECIMAL(15) タイム・スタンプの形式は CICS ABSTIME です。
<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*y*"/> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下: BIT(*z*) ここで、*z* = 8 ×*y* および *z* < 4095 バイトです。 CHAR(*z*) ここで、*z* = 8 ×*y* および *z* > 4095 バイトです。 マッピング・レベル 1.2 以上: CHAR(*y*)

スキーマ	PL/I データ記述
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** SIGNED FIXED BINARY (7) **その他の PL/I** FIXED BINARY (7)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** UNSIGNED FIXED BINARY (8) **その他の PL/I** FIXED BINARY (8)
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** SIGNED FIXED BINARY (15) **その他の PL/I** FIXED BINARY (15)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** UNSIGNED FIXED BINARY (16) **その他の PL/I** FIXED BINARY (16)
<xsd:simpleType> <xsd:restriction base="xsd:integer"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** FIXED DECIMAL(31,0) **その他の PL/I** FIXED DECIMAL(15,0)
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル: **Enterprise PL/I** SIGNED FIXED BINARY (31) **その他の PL/I** FIXED BINARY (31)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下: **Enterprise PL/I** UNSIGNED FIXED BINARY(32) マッピング・レベル 1.2 以上: **Enterprise PL/I** CHAR(*y*) ここで、*y* は 16 MB より小さい固定長です。 すべてのマッピング・レベル: **その他の PL/I** BIT(64)

スキーマ	PL/I データ記述
<xsd:simpleType> <xsd:restriction base="xsd:long"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下: **Enterprise PL/I** SIGNED FIXED BINARY(63) マッピング・レベル 1.2 以上: **Enterprise PL/I** CHAR(*y*) ここで、*y* は 16 MB より小さい固定長です。 すべてのマッピング・レベル: **その他の PL/I** BIT(64)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下: **Enterprise PL/I** UNSIGNED FIXED BINARY(64) マッピング・レベル 1.2 以上: **Enterprise PL/I** CHAR(*y*) ここで、*y* は 16 MB より小さい固定長です。 すべてのマッピング・レベル: **その他の PL/I** BIT(64)
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </xsd:restriction> </xsd:simpleType>	マッピング・レベル 1.1 以下: **Enterprise PL/I** SIGNED FIXED BINARY (7) **その他の PL/I** FIXED BINARY (7) マッピング・レベル 1.2 以上: **Enterprise PL/I** BIT(7) BIT(1) **その他の PL/I** BIT(7) BIT(1) ここで、BIT(7) は桁合わせ用に提供され、BIT(1) にはブール値によるマップ値が含まれます。
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>	すべてのマッピング・レベル:FIXED DECIMAL(*n*,*m*)

スキーマ	PL/I データ記述
<xsd:simpleType> <xsd:list> <xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType> </xsd:list> </xsd:simpleType>	すべてのマッピング・レベル:CHAR(255)
<xsd:simpleType> <xsd:union memberTypes="xsd:int xsd:string"/> </xsd:simpleType>	すべてのマッピング・レベル:CHAR(255)
<xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> <xsd:length value="*y*"/> </xsd:restriction> </xsd:simpleType> <xsd:simpleType> <xsd:restriction base="xsd:base64Binary"> </xsd:restriction> </xsd:simpleType> この場合、長さは定義されていません。	マッピング・レベル 1.0: サポート対象外 マッピング・レベル 1.1: CHAR(*z*) ここで、*z* =4×(ceil(*y*/3)) です。 ceil(*x*) は、*x* より大きいか等しい最小の整数です。 マッピング・レベル 1.2 以上: CHAR(*y*) この場合、長さは固定されています。 CHAR(16) この場合、長さは定義されていません。このフィールドは、2 進データが保管されるコンテナーの 16 バイトの名前を保持します。
<xsd:simpleType> <xsd:restriction base="xsd:float"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.0 および 1.1: CHAR(32) マッピング・レベル 1.2 以上: **Enterprise PL/I** DECIMAL FLOAT(6) HEXADEC **その他の PL/I** DECIMAL FLOAT(6)
<xsd:simpleType> <xsd:restriction base="xsd:double"> </xsd:restriction> </xsd:simpletype>	マッピング・レベル 1.0 および 1.1: CHAR(32) マッピング・レベル 1.2 以上: **Enterprise PL/I** DECIMAL FLOAT(16) HEXADEC **その他の PL/I** DECIMAL FLOAT(16)

関連資料

469 ページの『PL/I から XML スキーマへのマッピング』
DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは、 PL/I データ構造と XML スキーマ定義との間のマッピングをサポートします。Enterprise

PL/I コンパイラーと以前の PL/I コンパイラーには相違があるため、PLI-ENTERPRISE および PLI-OTHER の 2 種類の言語オプションがサポートされています。

486 ページの『<xsd:any> と xsd:anyType のサポート』
DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

エレメントの可変配列

XML には、可変数のエレメントの配列を含めることができます。通常、可変数のエレメントが含まれる WSDL 文書と XML スキーマを単一の高水準言語データ構造に効率的にマップすることはできません。CICS は、コンテナー・ベースのマッピングまたはインライン・マッピングを使用して、XML 内の可変数のエレメントを扱います。

可変数のエレメントを持つ配列は、XML スキーマ内ではエレメント宣言で minOccurs 属性および maxOccurs 属性を使用して表されます。

- minOccurs 属性は、エレメントが発生できる最小回数を指定します。0 または任意の正整数を指定できます。
- maxOccurs 属性は、エレメントが発生できる最大回数を指定します。minOccurs 属性の値より大きいか等しい任意の正整数を指定できます。また、unbounded の値を指定して、エレメントが発生できる回数に上限が適用されないことを示すこともできます。
- どちらの属性もデフォルト値は 1 です。

この例では、オプションである 8 バイト・ストリングを示しています。つまり、このストリングはアプリケーション XML または SOAP メッセージでゼロ回または 1 回出現できます。

```
<xsd:element name="component" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

以下の例では、少なくとも 1 回出現する必要のある 8 バイト・ストリングを示します。

```
<xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

通常、可変数のエレメントが含まれる WSDL 文書を単一の高水準言語データ構造に効率的にマップすることはできません。CICS ではこのような場合に対処するた

め、アプリケーション・プログラムに一連のコンテナーで渡される一連の関連データ構造が使用されます。これらの構造は、アプリケーションの入出力として使用されます。

- CICS は XML をアプリケーション・データに変換するときに、アプリケーション・データをこれらの構造に取り込み、アプリケーションはそれらの構造を読み取ります。
- CICS はアプリケーション・データを XML に変換するときに、アプリケーションによって取り込まれた構造内のアプリケーション・データを読み取ります。

これらのデータ構造のフォーマットを理解するには、複数の例を参照するのが一番です。XML は SOAP メッセージまたはアプリケーションから生成できます。これらの例では、単純な 8 バイト・フィールドの配列を使用します。ただし、このモデルでは、複合データ型の配列や他の配列を含むデータ型の配列がサポートされています。

固定数のエレメント

最初の例では、3 回だけ出現するエレメントを示します。

```
<xsd:element name="component" minOccurs="3" maxOccurs="3">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

この例では、エレメントの出現回数が事前に分かっているので、単純な COBOL 宣言 (または他の言語における同等部分) で固定長配列として表すことができます。

```
05 component PIC X(8) OCCURS 3 TIMES
```

マッピング・レベルが 2 以下である可変数のエレメント

この例では、1 から 5 回出現できる必須エレメントを示します。

```
<xsd:element name="component" minOccurs="1" maxOccurs="5">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:length value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

主データ構造には、2 つのフィールドの宣言が含まれています。CICS が XML をバイナリー・データに変換するとき、最初のフィールド component-num にはエレメントが XML で出現する回数が含まれ、2 番目のフィールド component-cont にはコンテナー名が含まれます。

```
05 component-num PIC S9(9) COMP-5
05 component-cont PIC X(16)
```

2 つ目のデータ構造には、エレメント自体の宣言が含まれます。

```
01 DFHWS-component
  02 component PIC X(8)
```

エレメントの出現回数を確認するには、component-num (1 から 5 の範囲の値が含まれる) の値を調べる必要があります。エレメントの内容は、component-cont で指定されたコンテナー内にあります。このコンテナーは、各エレメントが DFHWS-component データ構造によってマップされるエレメントの配列を保持します。

minOccurs="0" および maxOccurs="1" の場合、このエレメントはオプションです。アプリケーション・プログラムでデータ構造を処理するには、component-num の値を調べる必要があります。

- 値が 0 の場合、メッセージに component エレメントはなく、component-cont の内容は未定義になっています。
- 値が 1 の場合、component エレメントは component-cont で指定されたコンテナー内にあります。

コンテナーの内容は、DFHWS-component データ構造によってマップされます。

注: SOAP メッセージが 1 つの繰り返しエレメントによって構成されている場合、DFHWS2LS によって 2 つの言語構造が生成されます。主要な言語構造には、配列内のエレメント数と、そのエレメントの配列が保持されているコンテナーの名前が含まれています。 2 番目の言語構造は、繰り返しエレメントの 1 つのインスタンスをマップします。

マッピング・レベルが 2.1 以上である可変数のエレメント

マッピング・レベル 2.1 では、CICS 支援機能で **INLINE-MAXOCCURS-LIMIT** パラメーターを使用できます。**INLINE-MAXOCCURS-LIMIT** パラメーターは、可変数のエレメントの処理方法を指定します。可変数のエレメント用のマッピング・オプションは、コンテナー・ベースのマッピング (480 ページの『マッピング・レベルが 2 以下である可変数のエレメント』で説明されています) またはインライン・マッピングです。このパラメーターの *value* は、0 から 32767 の範囲の正整数で指定できます。

- **INLINE-MAXOCCURS-LIMIT** のデフォルト値は 1 です。この場合、オプションのエレメントは必ずインラインでマップされます。
- **INLINE-MAXOCCURS-LIMIT** パラメーターの値に 0 を指定すると、インライン・マッピングは行われません。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値より小さいか等しい場合、インライン・マッピングが使用されます。
- maxOccurs が **INLINE-MAXOCCURS-LIMIT** の値より大きい場合、コンテナー・ベースのマッピングが使用されます。

可変数のエレメントをインラインでマッピングすると、配列 (出現数が固定されている上述の例と同様) およびカウンターの両方が生成されます。component-num フィールドは存在するエレメントのインスタンス数を示し、これらのインスタンスは配列が指します。 480 ページの『マッピング・レベルが 2 以下である可変数のエレメント』で示されている例の場合、**INLINE-MAXOCCURS-LIMIT** が 5 より小さいか等しいとき、生成されるデータ構造はこのようになります。

```
05 component-num PIC S9(9) COMP-5 SYNC.
05 component OCCURS 5 PIC X(8).
```

最初のフィールド component-num は、前のセクションで参照したコンテナー・ベースのマッピング例における出力と同じです。2 番目のフィールドには、長さが 5 (生成することが可能なエレメントの最大数を収めるのに十分な大きさ) である配列が含まれます。

インライン・マッピングではすべてのデータが現行コンテナーに保管されるので、エレメントの出現回数およびデータの保管先であるコンテナー名が保管されるコンテナー・ベースのマッピングとは異なります。一般に、データを現行コンテナーに保管するとパフォーマンスが改善されるため、インライン・マッピングの方が望ましい選択肢になります。

ネストされた可変配列

複合型の WSDL 文書と XML スキーマには、繰り返し数が可変の繰り返しエレメントを含め、そのエレメント内にも繰り返し数が可変の繰り返しエレメントを含めることができます。この場合、記述される構造は、例で記述された 2 つのレベルを超えて拡張されます。

この例では、<component1> という必須エレメント (この必須エレメントは 1 から 5 回出現できます) 内でネストされた <component2> というオプションのエレメントを示します。

```
<xsd:element name="component1" minOccurs="1" maxOccurs="5">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="component2" minOccurs="0" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:length value="8"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

最上位データ構造は、前の例とまったく同じです。

```
05 component1-num PIC S9(9) COMP-5
05 component1-cont PIC X(16)
```

ただし、2 番目のデータ構造には以下のエレメントが含まれます。

```
01 DFHWS-component1
  02 component2-num PIC S9(9) COMP-5
  02 component2-cont PIC X(16)
```

3 番目のレベルの構造には以下のエレメントが含まれます。

```
01 DFHWS-component2
  02 component2 PIC X(8)
```

最外部のエレメント <component1> の出現回数は、component1-num に入っています。

component1-cont で指定されたコンテナーには、配列と共に、 2 番目のデータ構造 DFHWS-component1 のインスタンス数が含まれます。

component2-cont の各インスタンスは異なるコンテナーを指定し、それぞれのコンテナーには、第 3 レベルの構造 DFHWS-component2 によってマップされるデータ構造が含まれます。

この構造を示すため、例に一致する以下の XML フラグメントを考慮します。

```
<component1><component2>string1</component2></component1>
<component1><component2>string2</component2></component1>
<component1></component1>
```

<component1> は 3 回出現します。最初の 2 つにはそれぞれ <component2> のインスタンスが含まれますが、3 番目のインスタンスには含まれません。

最上位データ構造では、component1-num に値 3 が含まれます。component1-cont で指定されたコンテナーには、DFHWS-component1 の 3 つのインスタンスが含まれます。

1. 最初のインスタンスでは、 component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string1* を保持しています。
2. 2 番目のインスタンスでは、component2-num の値は 1 で、component2-cont で指定されたコンテナーは *string2* を保持しています。
3. 3 番目のインスタンスでは、component2-num の値は 0 で、component2-cont の内容は未定義になっています。

このインスタンスの場合、完全なデータ構造が全部で以下の 4 つのコンテナーによって表されます。

- コンテナー DFHWS-DATA 内のルート・データ構造
- component1-cont で指定されたコンテナー
- component2-cont の最初の 2 つのインスタンスで指定された 2 つのコンテナー

オプションの構造および xsd:choice

マッピング・レベルが 2.1 以上の場合のみ、DFHWS2LS および DFHSC2LS は、<xsd:sequence> エレメント、<xsd:choice> エレメント、および <xsd:all> エレメントでの maxOccurs と minOccurs の使用をサポートします。その際、minOccurs 属性および maxOccurs 属性は、minOccurs="0" および maxOccurs="1" に設定されます。

支援機能により、それぞれの子エレメントがオプションになっているものとしてこれらのエレメントを扱うマッピングが生成されます。これらのエレメントを使用してアプリケーションを実装する場合は、アプリケーションによって無効なオプションの組み合わせが生成されないように注意してください。各エレメントは、生成された言語構造内に独自の count フィールドがあり、これらのフィールドはすべて "0" または "1" のいずれかに設定する必要があります。<xsd:choice> エレメントの場合を除き、他の組み合わせの値は無効です。

<xsd:choice> エレメントは、エレメント内の 1 つのオプションのみが使用可能であることを示します。これは、すべてのマッピング・レベルでサポートされています。支援機能は、<xsd:choice> 内の各オプションを、minOccurs="0" および maxOccurs="1" が指定された <xsd:sequence> エレメント内にある場合と同じように扱います。<xsd:choice> エレメントを使用してアプリケーションを実装する場合は、アプリケーションによって無効なオプションの組み合わせが生成されないよう

に注意してください。各エレメントは、言語構造内に独自の count フィールドがあり、その中の 1 つを必ず '1' に、その他のすべてのフィールドを '0' に設定する必要があります。<xsd:choice> エレメント自体がオプションである (すべてのフィールドを '0' に設定できます) 場合を除き、これ以外の値の組み合わせは無効です。

XML 属性のサポート

XML スキーマは、XML で許可されているか、または必要な属性を指定できます。CICS 支援機能ユーティリティーである DFHWS2LS および DFHSC2LS は XML 属性を無視するようにデフォルト設定されています。XML スキーマで定義されている XML 属性を処理するには、**MAPPING-LEVEL** パラメーターの値を 1.1 以上に設定する必要があります。

オプションの属性

属性にはオプションのものと必須のものがあり、SOAP メッセージまたはアプリケーションの XML のどのエレメントにも関連付けることができます。スキーマで定義されたオプションの属性ごとに、以下の 2 つのフィールドが適切な言語構造で生成されます。

1. 存在フラグ: このフィールドは、ブール・データ型として扱われ、通常は 1 バイト長です。
2. 値: このフィールドは、同等の型の XML エレメントと同様にマップされます。例えば、タイプ NMTOKEN の属性は、タイプ NMTOKEN の XML エレメントと同じようにマップされます。

属性の存在フィールドと値フィールドは、関連付けされたエレメントのフィールドの前に、生成された言語構造で表示されます。インスタンス文書に表れた予期しない属性は無視されます。

例えば、以下のスキーマの属性定義を考えます。

```
<xsd:attribute name="age" type="xsd:short" use="optional" />
```

このオプションの属性は、以下の COBOL 構造にマップします。

```
05 attr-age-exist  PIC X DISPLAY
05 attr-age-value  PIC S9999 COMP-5 SYNC
```

オプションの属性のランタイム処理

オプションの属性について、以下のランタイム処理が実行されます。

- 属性が存在する場合、存在フラグが設定され、値がマップされます。
- 属性が存在しない場合、存在フラグは設定されません。
- 属性にデフォルト値が設定されていて存在する場合、値がマップされます。
- 属性にデフォルト値が設定されていて存在しない場合、デフォルト値がマップされます。

デフォルト値が設定されたオプションの属性は、必須属性として扱われます。

CICS がデータを XML に変換する際、以下のランタイム処理が実行されます。

- 存在フラグが設定されている場合、属性は変換され、XML に組み込まれます。
- 存在フラグが設定されていない場合、属性は XML に組み込まれません。

必須属性およびランタイム処理

必要な属性ごとに、値フィールドのみが適切な言語構造で生成されます。

属性が XML 内に存在する場合、値がマップされます。属性が存在しない場合、以下の処理が発生します。

- アプリケーションが Web サービス・プロバイダーの場合、CICS はクライアントの SOAP メッセージ内に、エラーを示す SOAP 障害メッセージを生成します。
- アプリケーションが Web サービス要求側の場合、CICS はメッセージを発行して、13 の RESP2 コード付きの変換エラー応答をアプリケーションに返します。
- アプリケーションが TRANSFORM XMLTODATA コマンドを使用している場合、CICS はメッセージを発行して、3 の RESP2 コード付きの無効な要求応答をアプリケーションに返します。

CICS が COMMAREA またはコンテナーの内容に基づいて SOAP メッセージを生成するときに、属性は変換され、メッセージに組み込まれます。アプリケーションが TRANSFORM DATATOXML コマンドを使用する場合、CICS は属性も変換し、XML に組み込みます。

nillable 属性

nillable 属性は、XML スキーマの `xsd:element` に表すことが可能な特別な属性です。`xsi:nil` 属性が XML 内のエレメントに対して有効であることを指定します。エレメントに `xsi:nil` 属性が指定されている場合は、エレメントが存在するものの値がないこと、そのため関連付けられている内容がないことを示します。

XML スキーマで、nillable 属性が TRUE と定義されている場合、nillable 属性はブール値をとる必須属性としてマップされます。

CICS が SOAP メッセージを受信するか、または `xsi:nil` 属性を含むアプリケーション用の XML を変換する必要がある場合、属性の値は TRUE か FALSE です。値が TRUE の場合、アプリケーションは `xsi:nil` 属性の有効範囲内のエレメントまたはネストされたエレメントの値を無視する必要があります。

`xsi:nil` 属性の値が TRUE になっている COMMAREA またはコンテナーの内容に基づいて、CICS が SOAP メッセージまたは XML を生成するとき、以下の処理が発生します。

- `xsi:nil` 属性が、XML または SOAP メッセージ内に生成されます。
- 関連付けられたエレメントの値は無視されます。
- エレメント内の、ネストされたエレメントはすべて無視されます。

SOAP メッセージの例

WSDL 文書の一部になることが可能な以下の XML スキーマの例について考えます。

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="root" nillable="true">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element nillable="true" name="num" type="xsd:int"  maxOccurs="3" minOccurs="3"/>
```

```
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

このスキーマに準拠する SOAP メッセージの一部の例を以下に示します。

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <num xsi:nil="true"/>
 <num>15</num>
 <num xsi:nil="true"/>
</root>
```

COBOL の場合、この SOAP メッセージはこれらのエレメントにマップします。

```
05    root
10    attr-nil-root-value  PIC X DISPLAY
10    num                  OCCURS 3
15    num1                 PIC S9(9) COMP-5 SYNC
15    attr-nil-num-value   PIC X DISPLAY
10    filler               PIC X(3)
```

関連資料

445 ページの『CICS 支援機能用のマッピング・レベル』
マッピングとは、言語構造と XML スキーマ間の情報の変換方法を指定する一連の規則です。現在使用可能で最も高度なマッピングを活用するには、CICS 支援機能の **MAPPING-LEVEL** パラメーターを最新レベルに設定することをお勧めします。

<xsd:any> と xsd:anyType のサポート

DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

<xsd:any> および xsd:anyType を CICS 支援機能で使用するには、以下のパラメーターを設定する必要があります。

- **MAPPING-LEVEL** パラメーターを 2.1 以上に設定します。
- Web サービス・プロバイダー・アプリケーションでは、**PGMINT** パラメーターを CHANNEL に設定します。

<xsd:any> の例

この例では、<xsd:any> を使用して、 "Customer" タグ内の "Surname" タグの後に、いくつかのオプションの非構造 XML 内容を記述します。

```
<xsd:element name="Customer">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Title" type="xsd:string"/>
   <xsd:element name="FirstName" type="xsd:string"/>
   <xsd:element name="Surname" type="xsd:string"/>
   <xsd:any minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

この XML スキーマに準拠する SOAP メッセージの例は以下のとおりです。

```
<xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Customer xmlns="http://www.example.org/anyExample">
      <Title xmlns="">Mr</Title>
      <FirstName xmlns="">John</FirstName>
      <Surname xmlns="">Smith</Surname>
      <ExtraInformation xmlns="http://www.example.org/ExtraInformation">
        <!-- この「ExtraInformation」タグは、XML スキーマのオプションの xsd:any に関連付けられています。
             正しく形成されたすべての XML を含めることができます。-->
        <ExampleField1>one</ExampleField1>
        <ExampleField2>two</ExampleField2>
      </ExtraInformation>
    </Customer>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

この SOAP メッセージが CICS に送信されると、CICS は以下の XML データを Customer-xml-cont コンテナーに入力します。

```
<ExtraInformation xmlns="http://www.example.org/ExtraInformation">
  <!-- この「ExtraInformation」タグは、XML スキーマのオプションの xsd:any に関連付けられています。
       正しく形成されたすべての XML を含めることができます。-->
  <ExampleField1>one</ExampleField1>
  <ExampleField2>two</ExampleField2>
</ExtraInformation>
```

CICS は、有効範囲内の以下の XML 名前空間宣言も Customer-xmlns-cont コンテナーに入力します。これらの宣言はスペースで区切られています。

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns="http://www.example.org/anyExample"
```

xsd:anyType の例

xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容の制限はありません。データ型を指定しない場合、デフォルトの xsd:anyType に設定されます。例えば、これらの 2 つの XML フラグメントは同等になります。

```
<xsd:element name="Name" type="xsd:anyType"/>
```

```
<xsd:element name="Name"/>
```

生成された言語構造

<xsd:any> または xsd:anyType 用に生成された言語構造の形式は、COBOL の以下の形式および他の言語の同等の形式です。

elementName-xml-cont PIC X(16)

ロー XML を保持するコンテナーの名前。CICS は、着信 SOAP メッセージを処理するときに、<xsd:any> または xsd:anyType によって定義される SOAP メッセージのサブセットをこのコンテナーに格納します。アプリケーションは、XML データをネイティブにのみ処理できます。アプリケーションは XML を生成し、このコンテナーにデータを取り込み、そのコンテナー名を提供する必要があります。

このコンテナーはテキスト・モードでデータを取り込む必要があります。CICS がこのコンテナーにデータを取り込む場合、Web サービスで使用するように定義されているのと同じ EBCDIC バリアントを使用します。ターゲ

ット EBCDIC コード・ページに存在しない文字は、コンテナーが UTF-8 のアプリケーションで読み取られる場合でも、置換文字に置き換えられます。

elementName-xmlns-cont PIC X(16)
有効範囲内の名前空間接頭部宣言を保持するコンテナーの名前。このコンテナーの内容は、DFHWS-XMLNS コンテナーの内容と似ています。ただし、SOAP Envelope タグの単なるサブセットではなく、有効範囲内に存在する関連性のある名前空間宣言がすべて含まれる点が異なります。

このコンテナーはテキスト・モードでデータを取り込む必要があります。CICS がこのコンテナーにデータを取り込む場合、Web サービスで使用するように定義されているのと同じ EBCDIC バリアントを使用します。ターゲット EBCDIC コード・ページに存在しない文字は、コンテナーが UTF-8 のアプリケーションで読み取られる場合でも、置換文字に置き換えられます。

このコンテナーは、CICS に送信された SOAP メッセージを処理する場合にのみ使用されます。出力 SOAP メッセージが生成されたときに、アプリケーションがコンテナーに名前空間宣言の提供を試行すると、コンテナーおよびその内容は、CICS によって無視されます。CICS は、アプリケーションによって提供された XML が、名前空間宣言について必要なものを完備していることを要求します。

<xsd:any> エレメントが含まれている XML エレメントの名前は、<xsd:any> エレメントのために生成される変数名に含まれます。 <xsd:any> の例では、<xsd:any> エレメントは <xsd:element name="Customer"> エレメント内にネストされていて、<xsd:any> エレメントのために生成される変数名は `Customer-xml-cont PIC X(16)` と `Customer-xmlns-cont PIC X(16)` です。

`xsd:anyType` 型の場合、直接の XML エレメント名が使用されます。上記の `xsd:anyType` の例では、変数名は `Name-xml-cont PIC X(16)` および `Name-xmlns-cont PIC X(16)` です。

関連タスク

509 ページの『<xsd:any> データ型の処理』
1 つ以上の <xsd:any> データ型が含まれる XML スキーマを処理する場合は、XML 支援機能でデータ型を 1 対の CICS コンテナーにマップすることができます。コンテナー内の XML を解析するアプリケーション・プログラムを作成できます。

関連資料

458 ページの『XML スキーマと COBOL とのマッピング』
DFHSC2LS および DFHWS2LS ユーティリティー・プログラムでは、 XML スキーマ定義と COBOL データ構造との間のマッピングをサポートしています。

465 ページの『XML スキーマと C および C++ とのマッピング』
DFHSC2LS および DFHWS2LS ユーティリティー・プログラムは、各 Web サービス記述に含まれる XML スキーマ定義と C および C++ データ型との間のマッピングをサポートしています。

473 ページの『XML スキーマと PL/I とのマッピング』
DFHSC2LS および DFHWS2LS ユーティリティー・プログラムでは、 XML ス

キーマ定義と PL/I データ構造との間のマッピングをサポートしています。Enterprise PL/I コンパイラーと以前の PL/I コンパイラーには相違があるため、PLI-ENTERPRISE および PLI-OTHER の 2 種類の言語オプションがサポートされています。

<xsd:choice>のサポート

<xsd:choice> エレメントは、エレメントの 1 つのオプションのみが使用可能であることを示します。CICS 支援機能は <xsd:choice> エレメントを、さまざまな深度とマッピング・レベルでサポートします。

マッピング・レベル 2.2 以上での <xsd:choice>のサポート

マッピング・レベル 2.2 以上では、DFHWS2LS および DFHSC2LS が <xsd:choice> エレメントをさらに強力にサポートします。支援機能は、<xsd:choice> エレメントに関連付けられた値を格納する新規コンテナーを生成します。また、新規コンテナーおよび予備のフィールドを格納する言語構造を生成します。

fieldname-**enum**
: <xsd:choice> エレメントが使用するオプションを示すための識別フィールド。

fieldname-**cont**
: 使用するオプションを格納するコンテナーの名前。オプションの値をマップするために、さらに言語構造が生成されます。

以下の XML スキーマ・フラグメントには、<xsd:choice> エレメントが含まれます。

```
<xsd:element name="choiceExample">
   <xsd:complexType>
      <xsd:choice>
         <xsd:element name="option1" type="xsd:string" />
         <xsd:element name="option2" type="xsd:int" />
         <xsd:element name="option3" type="xsd:short" maxOccurs="2" minOccurs="2" />
      </xsd:choice>
   </xsd:complexType>
</xsd:element>
```

この XML スキーマ・フラグメントがマッピング・レベル 2.2 以上で処理された場合、支援機能によって以下の COBOL 言語構造が生成されます。

```
   03 choiceExample.
      06 choiceExample-enum         PIC X DISPLAY.
          88 empty                     VALUE X'00'.
          88 option1                   VALUE X'01'.
          88 option2                   VALUE X'02'.
          88 option3                   VALUE X'03'.
      06 choiceExample-cont         PIC X(16).


01 Example-option1.
   03 option1-length                PIC S9999 COMP-5 SYNC.
   03 option1                       PIC X(255).

01 Example-option2.
   03 option2                       PIC S9(9) COMP-5 SYNC.
```

```
01 Example-option3.
   03 option3 OCCURS 2             PIC S9999 COMP-5 SYNC.
```

マッピング・レベル 2.2 以上での <xsd:choice> への制限

DFHSC2LS および DFHWS2LS では、ネストされた <xsd:choice> エレメントはサポート対象外です。例えば、次の XML はサポートされません。

```
<xsd:choice>
   <xsd:element name ="name1" type="string"/>
   <xsd:choice>
      <xsd:element name ="name2a" type="string"/>
      <xsd:element name ="name2b" type="string"/>
   </xsd:choice>
</xsd:choice>
```

DFHSC2LS および DFHWS2LS では、繰り返し発生する <xsd:choice> エレメントはサポート対象外です。例えば、次の XML はサポートされません。

```
<xsd:choice maxOccurs="2">
   <xsd:element name ="name1" type="string"/>
</xsd:choice>
```

DFHSC2LS および DFHWS2LS では、最大 255 個の <xsd:choice> エレメントのオプションがサポートされます。

マッピング・レベル 2.1 以下での <xsd:choice>のサポート

マッピング・レベル 2.1 以下では、<xsd:choice> エレメントへの DFHWS2LS のサポートは制限されます。DFHWS2LS は、<xsd:choice> エレメントの各オプションを、最大 1 回発生する <xsd:sequence> エレメントであるかのように扱います。

<xsd:choice> エレメントでは 1 つのオプションのみを使用できます。したがって、<xsd:choice> エレメントを使用してアプリケーションを実装する場合は、有効な組み合わせのオプションのみを使用するように注意してください。各エレメントで生成された言語構造内には独自の count フィールドがあり、その中の 1 つは必ず 1 に、その他すべてのフィールドは 0 に設定する必要があります。これ以外の値の組み合わせは無効です。ただし、<xsd:choice> 自体がオプションのとき、すべてのフィールドを 0 に設定した場合は有効になります。

関連資料

486 ページの『<xsd:any> と xsd:anyType のサポート』
DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

492 ページの『抽象エレメントと抽象データ型のサポート』
CICS 支援機能は、抽象エレメントと抽象データ型をマッピング・レベル 2.2 以上でサポートします。CICS 支援機能は、抽象エレメントと抽象データ型を、代替グループと同じようにマップします。

『代替グループのサポート』
代替グループを使用して、交換可能な XML エレメントのグループを定義することができます。CICS 支援機能は、 代替グループをマッピング・レベル 2.2 以上でサポートします。

代替グループのサポート

代替グループを使用して、交換可能な XML エレメントのグループを定義することができます。CICS 支援機能は、代替グループをマッピング・レベル 2.2 以上でサポートします。

マッピング・レベル 2.2 以上の場合、DFHSC2LS と DFHWS2LS は、<xsd:choice> エレメントで使用されるものに類似したマッピングを使用して、代替グループをサポートします。支援機能では、列挙型フィールドと新規コンテナー名が言語構造内に生成されます。

以下の XML スキーマ・フラグメントには、2 つの subGroupParent エレメントの配列が含まれ、それぞれは replacementOption1 または replacementOption2: に置き換えることができます。

```
<xsd:element name="subGroupExample" >
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="subGroupParent" maxOccurs="2" minOccurs="2" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="subGroupParent" type="xsd:anySimpleType" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="subGroupParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="subGroupParent" />
```

この XML フラグメントを支援機能で処理すると、以下の COBOL 言語構造が生成されます。

```
   03 subGroupExample.
      06 subGroupParent OCCURS2.
         09 subGroupExample-enum PIC X DISPLAY.
            88 empty              VALUE X '00'.
            88 replacementOption1  VALUE X '01'.
            88 replacementOption2  VALUE X '02'.
            88 subGroupParent      VALUE X '03'.
         09 subGroupExample-cont PIC X (16).



01 Example-replacementOption1.
   03 replacementOption1      PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
   03 replacementOption2      PIC S9999 COMP-5 SYNC.

01 Example-subGroupParent.
   03 subGroupParent-length   PIC S9999 COMP-5 SYNC.
   03 subGroupParent          PIC X(255).
```

代替グループの詳細については、「*W3C XML Schema Part 1: Structures Second Edition*」仕様 (http://www.w3.org/TR/xmlschema-1/#Elements_Equivalence_Class) を参照してください。

関連資料

486 ページの『<xsd:any> と xsd:anyType のサポート』
DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

489 ページの『<xsd:choice>のサポート』
<xsd:choice> エレメントは、エレメントの 1 つのオプションのみが使用可能であることを示します。CICS 支援機能は <xsd:choice> エレメントを、さまざまな深度とマッピング・レベルでサポートします。

『抽象エレメントと抽象データ型のサポート』
CICS 支援機能は、抽象エレメントと抽象データ型をマッピング・レベル 2.2 以上でサポートします。CICS 支援機能は、抽象エレメントと抽象データ型を、代替グループと同じようにマップします。

抽象エレメントと抽象データ型のサポート

CICS 支援機能は、抽象エレメントと抽象データ型をマッピング・レベル 2.2 以上でサポートします。CICS 支援機能は、抽象エレメントと抽象データ型を、代替グループと同じようにマップします。

マッピング・レベル 2.2 での抽象エレメントのサポート

マッピング・レベル 2.2 以上では、 DFHSC2LS および DFHWS2LS は、抽象エレメントを代替グループとほぼ同じ方法で扱います。ただし、抽象エレメントがグループの有効なメンバーではない場合は異なります。代替可能なエレメントがない場合、抽象エレメントは <xsd:any> エレメントとして扱われ、同じマッピングをマッピング・レベル 2.1 の <xsd:any> エレメントとして使用します。

以下の XML スキーマ・フラグメントは、抽象エレメントの代わりに使用可能な 2 つのオプションを指定します。抽象エレメント自体は有効なオプションではありません。

```
<xsd:element name="abstractElementExample" >
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="abstractElementParent" maxOccurs="2" minOccurs="2" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<xsd:element name="abstractElementParent" type="xsd:anySimpleType" abstract="true" />
<xsd:element name="replacementOption1" type="xsd:int" substitutionGroup="abstractElementParent" />
<xsd:element name="replacementOption2" type="xsd:short" substitutionGroup="abstractElementParent" />
```

この XML フラグメントを支援機能で処理すると、以下の COBOL 言語構造が生成されます。

```
03 abstractElementExample.
   06 abstractElementParent OCCURS 2.
      09 abstractElementExample-enum  PIC X DISPLAY.
         88 empty                          VALUE X '00'.
         88 replacementOption1             VALUE X '01'.
         88 replacementOption2             VALUE X '02'.
      09 abstractElementExample-cont  PIC X (16).
```

```
01 Example-replacementOption1.
   03 replacementOption1                 PIC S9(9) COMP-5 SYNC.

01 Example-replacementOption2.
   03 replacementOption2                 PIC S9999 COMP-5 SYNC.
```

抽象エレメントの詳細については、「*W3C XML Schema Part 0: Primer Second Edition*」仕様 (http://www.w3.org/TR/xmlschema-0/#SubsGroups) を参照してください。

マッピング・レベル 2.2 での抽象データ型のサポート

マッピング・レベル 2.2 以上では、DFHSC2LS および DFHWS2LS は抽象データ型を代替グループとして扱います。支援機能では、列挙型フィールドと新規コンテナー名が言語構造内に生成されます。

以下の XML スキーマ・フラグメントは、抽象型の代わりに使用可能な 2 つの代替物を指定します。

```
<xsd:element name="AbstractDataTypeExample" type="abstractDataType" />

<xsd:complexType name="abstractDataType" abstract="true">
   <xsd:simpleContent>
      <xsd:extension base="xsd:string" />
   </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option1">
   <xsd:simpleContent>
      <xsd:restriction base="abstractDataType">
         <xsd:length value="5" />
      </xsd:restriction>
   </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="option2">
   <xsd:simpleContent>
      <xsd:restriction base="abstractDataType">
         <xsd:length value="10" />
      </xsd:restriction>
   </xsd:simpleContent>
</xsd:complexType>
```

この XML フラグメントを支援機能で処理すると、以下の COBOL 言語構造が生成されます。

```
   03 AbstractDataTypeExamp-enum   PIC X DISPLAY.
     88 empty                         VALUE X'00'.
     88 option1                       VALUE X'01'.
     88 option2                       VALUE X'02'.
   03 AbstractDataTypeExamp-cont   PIC X(16).
```

言語構造は、個別のコピーブックに生成されます。option1 用に生成された言語構造は、1 つのコピーブックに生成されます。

```
   03 option1          PIC X(5).
```

option2 用の言語構造は、別のコピーブックに生成されます。

```
   03 option2          PIC X(10).
```

抽象データ型の詳細については、「*W3C XML Schema Part 0: Primer Second Edition specification*」: http://www.w3.org/TR/xmlschema-0/#SubsGroupsを参照してください。

関連資料

486 ページの『<xsd:any> と xsd:anyType のサポート』
DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

489 ページの『<xsd:choice>のサポート』
<xsd:choice> エレメントは、エレメントの 1 つのオプションのみが使用可能であることを示します。CICS 支援機能は <xsd:choice> エレメントを、さまざまな深度とマッピング・レベルでサポートします。

491 ページの『代替グループのサポート』
代替グループを使用して、交換可能な XML エレメントのグループを定義することができます。CICS 支援機能は、 代替グループをマッピング・レベル 2.2 以上でサポートします。

COBOL での可変反復内容の処理方法

COBOL では、ポインター演算を使用してデータの各インスタンスをアドレス指定し、可変反復内容を処理することはできません。他のプログラミング言語には、このような制限はありません。この例では、COBOL で Web サービス・アプリケーション用の可変反復内容を処理する方法を示します。

この技法は、TRANSFORM API コマンドを使用して XML をアプリケーション・データに変換する場合にも適用されます。以下の WSDL 文書の例では、可変的な回数で再発する 8 文字のストリングで構成されるアプリケーション・データを持つ Web サービスを示します。

```
<?xml version="1.0"?>
<definitions name="ExampleWSDL"
             targetNamespace="http://www.example.org/variablyRepeatingData/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:tns="http://www.example.org/variablyRepeatingData/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <types>
   <xsd:schema targetNamespace="http://www.example.org/variablyRepeatingData/">
     <xsd:element name="applicationData">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
             <xsd:simpleType>
               <xsd:restriction base="xsd:string">
                 <xsd:length value="8"/>
               </xsd:restriction>
             </xsd:simpleType>
           </xsd:element>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
   </xsd:schema>
 </types>
```

```
 <message name="exampleMessage">
   <part element="tns:applicationData" name="messagePart"/>
 </message>

 <portType name="examplePortType">
   <operation name="exampleOperation">
     <input message="tns:exampleMessage"/>
     <output message="tns:exampleMessage"/>
   </operation>
 </portType>

 <binding name="exampleBinding" type="tns:examplePortType">
   <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="exampleOperation">
     <soap:operation soapAction=""/>
     <input><soap:body parts="messagePart" encodingStyle="" use="literal"/></input>
     <output><soap:body parts="messagePart" encodingStyle="" use="literal"/></output>
   </operation>
 </binding>
</definitions>
```

この WSDL 文書を DFHWS2LS で処理すると、以下の COBOL 言語構造が生成されます。

```
          03 applicationData.

            06 component-num                PIC S9(9) COMP-5 SYNC.
            06 component-cont               PIC X(16).


        01 DFHWS-component.
          03 component                    PIC X(8).
```

8 文字の component フィールドは、DFHWS-component という個別の構造で定義されることに注意してください。主データ構造は、applicationData と呼ばれ、ここには component-num および component-cont の 2 つのフィールドが含まれます。component-num フィールドは、component データに存在するインスタンス数を示し、component-cont フィールドは、component フィールドの連結リストを保持するコンテナー名を示します。

以下の COBOL コードは、可変的な繰り返しデータのリストを処理する方法の 1 つを示します。linkage section 配列を利用して、後続データのインスタンスをアドレス指定します。これはそれぞれ、DISPLAY ステートメントを使用して表示されます。

```
IDENTIFICATION DIVISION.
       PROGRAM-ID.    EXVARY.

       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

      * working storage variables
       01 APP-DATA-PTR             USAGE IS POINTER.
       01 APP-DATA-LENGTH          PIC S9(8) COMP.
       01 COMPONENT-PTR            USAGE IS POINTER.
       01 COMPONENT-DATA-LENGTH    PIC S9(8) COMP.
       01 COMPONENT-COUNT          PIC S9(8) COMP-4 VALUE 0.
       01 COMPONENT-LENGTH         PIC S9(8) COMP.

       LINKAGE SECTION.

      * a large linkage section array
       01 BIG-ARRAY PIC X(659999).
```

```
* application data structures produced by DFHWS2LS
* this is normally referenced with a COPY statement
 01 DFHWS2LS-data.
    03 applicationData.
      06 component-num  PIC S9(9) COMP-5 SYNC.
      06 component-cont PIC X(16).

 01 DFHWS-component.
    03 component        PIC X(8).



 PROCEDURE DIVISION USING DFHEIBLK.
 A-CONTROL SECTION.
 A010-CONTROL.

* Get the DFHWS-DATA container
     EXEC CICS GET CONTAINER('DFHWS-DATA')
               SET(APP-DATA-PTR)
               FLENGTH(APP-DATA-LENGTH)
     END-EXEC
     SET ADDRESS OF DFHWS2LS-data TO APP-DATA-PTR

* Get the recurring component data
     EXEC CICS GET CONTAINER(component-cont)
               SET(COMPONENT-PTR)
               FLENGTH(COMPONENT-DATA-LENGTH)
     END-EXEC

* Point the component structure at the first instance of the data
     SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* Store the length of a single component
     MOVE LENGTH OF DFHWS-component TO COMPONENT-LENGTH

* process each instance of component data in turn
     PERFORM WITH TEST AFTER
           UNTIL COMPONENT-COUNT = component-num

* display the current instance of the data
        DISPLAY 'component value is: ' component

* address the next instance of the component data
        SET ADDRESS OF BIG-ARRAY TO ADDRESS OF DFHWS-component
        SET ADDRESS OF DFHWS-component
            TO ADDRESS OF BIG-ARRAY (COMPONENT-LENGTH + 1:1)
        ADD 1 TO COMPONENT-COUNT

* end the loop
     END-PERFORM.

* Point the component structure back at the first instance of
* of the data, for any further processing we may want to perform
     SET ADDRESS OF DFHWS-component TO COMPONENT-PTR

* return to CICS.

     EXEC CICS
          RETURN
     END-EXEC

     GOBACK.
```

上記のコードは、可変反復内容を処理する一般的な解決策となります。配列 BIG-ARRAY は、各コンポーネントの開始位置に順次移動し、データの開始位置に固

定され続けることはありません。次に、構成要素データ構造は移動し、次のコンポーネントの最初のバイトをポイントします。COMPONENT-PTR は、必要に応じて、構成要素データの開始位置を回復する場合に使用できます。

WSDL 文書に準拠する SOAP メッセージの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
   <applicationData xmlns="http://www.example.org/variablyRepeatingData/">
     <component xmlns="">VALUE1</component>
     <component xmlns="">VALUE2</component>
     <component xmlns="">VALUE3</component>
   </applicationData>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

COBOL プログラムが SOAP メッセージを処理する際に生成する出力を示します。

```
CPIH 20080115103151 component value is: VALUE1
CPIH 20080115103151 component value is: VALUE2
CPIH 20080115103151 component value is: VALUE3
```

関連資料

453 ページの『COBOL から XML スキーマへのマッピング』
DFHLS2SC および DFHLS2WS ユーティリティー・プログラムは、 COBOL データ構造体と XML スキーマ定義間のマッピングをサポートします。

458 ページの『XML スキーマと COBOL とのマッピング』
DFHSC2LS および DFHWS2LS ユーティリティー・プログラムでは、 XML スキーマ定義と COBOL データ構造との間のマッピングをサポートしています。

可変長の値と空白のサポート

CICS 支援機能の設定値を使用したり、XML スキーマに直接ファセットを追加したりすることにより、可変長の値と空白を処理する方法をカスタマイズできます。

通常、CICS XML 支援機能と CICS Web サービス支援機能は、データ・ストリングを固定長の文字配列にマップします。こうした配列では、スペースや NULL で埋め込む必要があります。可変長の値を固定長のデータ配列にマッピングすると、効率が悪く、ストレージを浪費する恐れがあります。使用するデータ長が可変の場合、こうしたマッピングを処理する方法をカスタマイズすることが推奨されています。

ある言語構造から XML スキーマまたは WSDL 文書に変換する場合には、XML スキーマで whiteSpace ファセットと maxLength ファセットを指定し、支援機能で **CHAR-VARYING-LIMIT** パラメーターを設定することが推奨されています。

XML スキーマまたは WSDL 文書からある言語構造に変換する場合には、支援機能で **CHAR-VARYING** パラメーターを適切な値に設定することが推奨されています。

注: XML 文書では、NULL 文字 ('x00') は無効です。 CICS が構文解析するアプリケーション・データの NULL 文字はストリングの末尾を示すためのもので、値は切り捨てられます。 CICS は、アプリケーション・データを生成する際に、

CHAR-VARYING パラメーターの値に従って生成します。例えば、**CHAR-VARYING=NULL** オプションが指定されると、 CICS が生成する可変長ストリングは NULL 文字で終了します。

XML から言語構造への可変長の値のマッピング

XML スキーマのファセットを使用して、または CICS 支援機能で特定のパラメーターを指定して、XML スキーマまたは WSDL 文書と言語構造間のマッピングを処理する方法をカスタマイズします。

ファセットを使用して XML データ・タイプを制限することができます。 XML 内の可変長データの処理方法をカスタマイズするには、長さファセット (`length`、`maxLength`、および `minLength`) および `whiteSpace` ファセットを使用します。

`length` データが固定長となるように指定します。

`maxLength`
データ・タイプの最大長を指定します。ストリング・ベースのデータ・タイプでこの値を設定しないと、最大長は無制限になります。

`minLength`
データ・タイプの最小長を指定します。ストリング・ベースのデータ・タイプでこの値を設定しないと、最小長は 0 になります。

`whiteSpace`
データ値の周囲にある空白の処理方法を指定します。空白には、スペース、タブ、および改行が含まれます。 `whiteSpace` ファセットは、`preserve`、`replace`、または `collapse` に設定できます。

- 値 `preserve` は、データ値内の空白をすべて保持します。
- 値 `replace` は、タブまたは改行が適切な数のスペースに置換されることを意味します。
- 値 `collapse` は、先行空白および後続空白は除去され、タブ、改行、および連続スペースはすべて単一のスペース文字に置き換えられることを意味します。

`whiteSpace` ファセットが設定されないと、空白は保持されます。

XML スキーマ・ファセットの詳細については、W3C 勧告スキーマ「*XML Schema Part 2: Datatypes Second Edition*」(http://www.w3.org/TR/xmlschema-2/#facets) を参照してください。

CICS 支援機能でパラメーター DFHSC2LS および DFHWS2LS を使用すると、XML スキーマから言語構造への可変長データのマッピング方法を変更できます。これらのパラメーターは、マッピング・レベル 1.2 以降で使用可能です。

DEFAULT-CHAR-MAXLENGTH
XML スキーマ文書または WSDL 文書で長さが指定されていない場合に、マッピングする文字の文字データのデフォルトの配列長を指定します。このパラメーターの値は、1 から 2 147 483 647 の範囲の正整数で指定できます。

ただし、XML スキーマまたは WSDL 文書で `maxLength` ファセットを使用して、DFHSC2LS または DFHWS2LS が使用する最大文字長を直接指定す

ることをお勧めします。 XML スキーマまたは WSDL 文書で最大長を直接指定すると、すべてのストリング・ベースのデータ・タイプに 1 つのグローバル・デフォルトが適用されることに関連した問題を回避できます。

CHAR-VARYING-LIMIT
言語構造にマップされる可変長文字データの最大サイズを指定します。文字データが、このパラメーターで指定されている値よりも大きい場合は、コンテナーにマップされ、生成された言語構造でそのコンテナー名が使用されます。値は 0 からデフォルトの 32 767 バイトの範囲で指定できます。

CHAR-VARYING
可変長文字データがどのようにマップされるのかを指定します。このパラメーターを指定しない場合、デフォルトのマッピングは、指定された言語によって決まります。以下のオプションを選択できます。

- **CHAR-VARYING=NO** は、可変長文字データが固定長ストリングとしてマップされるように指定します。
- **CHAR-VARYING=NULL** は、可変長文字データがヌル終了ストリングにマップされるように指定します。
- **CHAR-VARYING=YES** は、可変長文字データが PL/I で CHAR VARYING データ・タイプにマップされるように指定します。
  COBOL、C、および C++ 言語の場合、可変長文字データは、2 つの関連エレメント (データ長およびデータ) で構成される同等の表現にマップされます。

通常、**CHAR-VARYING=YES** に設定すると、最高のパフォーマンスが得られます。

言語構造から XML への可変長の値のマッピング

ご使用の言語構造と XML スキーマまたは WSDL 文書間におけるマッピングの処理方法をカスタマイズできます。 DFHLS2SC または DFHLS2WS の **CHAR-VARYING** パラメーターを COLLAPSE または NULL に設定して、文字配列が生成される方法を変更します。

CHAR-VARYING=NULL オプションを設定すると、CICS に XML を生成する際に各文字配列の末尾に NULL 文字を追加するように指示することになります。

CHAR-VARYING=COLLAPSE オプションを設定すると、CICS に XML を生成する際に文字配列の末尾から後続スペースを自動的に除去するように指示することになります。このオプションが有効なのはマッピング・レベルが 2.1 以上の場合だけで、C および C++ 以外のすべての言語ではマッピング・レベル 2.1 以降のデフォルト値が **CHAR-VARYING=COLLAPSE** であることに注意してください。

詳しくは、Support for white space and variable length values in CICS Web services (Technote) を参照してください。

言語構造からのマッピングの生成

XML をアプリケーション・データから作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータおよび XML を変換する方法を記述する必要があります。これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または DB2 レコードから開始できます。

始める前に

マッピングを作成する前に、以下の前提条件が完了していることを確認する必要があります。

- 区分データ・セット内に、アプリケーション・レコードを記述する言語構造が必要です。言語構造は、COBOL、PL/I、C、および C++ など、CICS XML 支援機能でサポートされている高水準言語のいずれであってもかまいません。このマッピングを Atom フィードで使用する場合で、(作成者名など) Atom エントリーのメタデータを提供するためにアプリケーション・レコードのいずれかのフィールドを使用する場合、そのフィールドが言語構造内でネストされていないことを確認してください。Atom エントリーの内容を提供するフィールド内では、ネストされているフィールドの構造を使用することができます。
- DFHLS2SC が実行されて z/OS UNIX を使用するユーザー ID を構成する必要があります。
- このユーザー ID には、言語構造にアクセスするための読み取り権限と、z/OS UNIX の適切なディレクトリーに出力するための書き込み権限が必要です。

このタスクについて

CICS XML 支援機能を使用して、アプリケーション・レコードのデータ・マッピングを作成します。CICS XML 支援機能でサポートされている各高水準言語では、いくつかのタイプのデータが制限されているかサポートされていません。CICS XML 支援機能では、言語構造内で識別された非サポートのアイテムに関して、エラー・メッセージが出されます。CICS XML 支援機能に関する参照情報では、各高水準言語に適用される制限がリストされています。

1. DFHLS2SC バッチ・ジョブを実行します。 DFHLS2SC には、特定のコード・ページや名前空間の選択など、要件を満たすために選択するオプション・パラメーターが用意されています。少なくとも以下のパラメーターを使用することをお勧めします。
   a. 言語構造の高水準言語を **LANG** パラメーターで指定します。
   b. データ・マッピングをバンドルでデプロイする場合、バンドル・リソース名を **BUNDLE** パラメーターで指定します。 Atom フィードの XML バインディングを作成している場合は、このパラメーターを指定しないでください。
   c. マッピング・レベルを **MAPPING-LEVEL** パラメーターで指定します。 Atom フィードの XML バインディングを作成している場合は、マッピング・レベル 3.0 を使用する必要があります。 他の状況ではどのマッピング・レベルも使用できますが、最も高機能のマッピング・オプションを取得するには、マッピング・レベル 3.0 の使用をお勧めします。
   d. オプション: Atom フィードの XML バインディングを作成していて、アプリケーション・データ・レコードに CICS ABSTIME 形式のタイム・スタンプ

が含まれている場合は、オプション・パラメーター **DATETIME=PACKED15** を指定してこれらのフィールドをタイム・スタンプとしてマップしてください。

e. **PDSMEM** パラメーターと **PDSCP** パラメーターで、アプリケーション・レコードを記述する言語構造の場所とコード・ページを指定します。

f. スキーマ・ファイルの名前と場所を **SCHEMA** パラメーターで指定します。ファイル拡張子は .xsd です。バンドルを作成している場合は場所を指定しないでください。DFHLS2SC は XML スキーマを作成しますが、ファイルがまだ存在していない場合、ディレクトリー構造は作成しません。

g. XML バインディングの名前と場所を **XSDBIND** パラメーターで指定します。 ファイル拡張子は .xsdbind です。バンドルを作成している場合は場所を指定しないでください。DFHLS2SC は XML バインディングを作成しますが、ファイルがまだ存在していない場合、ディレクトリー構造は作成しません。

ヒント: XML バインディングとスキーマを同じディレクトリー構造に格納して、妥当性検査を有効にします。妥当性検査は、アプリケーションを開発またはテスト環境でテストする場合に役立ちます。バンドルを作成している場合、CICS では同じディレクトリーにファイルが配置されます。

BUNDLE パラメーターを指定した場合、バッチ・ジョブでは z/OS UNIX 上にバンドル・ディレクトリー構造が作成されます。バンドル・ディレクトリーには、バンドル・マニフェストが含まれる META-INF サブディレクトリーがあります。バッチ・ジョブは、**SCHEMA** パラメーターと **XSDBIND** パラメーターで指定したファイル名を使用して、XML スキーマと XML バインディングもバンドル・ディレクトリー内に作成します。**BUNDLE** パラメーターを指定しない場合、バッチ・ジョブにより、XML スキーマと XML バインディングのみが、指定された場所に作成されます。

2. BUNDLE リソース、またはこの XML バインディングを指定する ATOMSERVICE リソースをインストールします。 BUNDLE リソースおよび ATOMSERVICE リソースにより、XML スキーマとバインディング・ファイルの場所を定義する XMLTRANSFORM リソースが動的に作成されます。

タスクの結果

言語構造からマッピングを生成する場合は、1 種類の XML の変換のみが可能です。

例

以下の例は、すべての必須パラメーターと推奨パラメーターが指定された状態の DFHLS2SC を示しています。

```
//LS2SC JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHLS2SC,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
```

```
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
SCHEMA=example.xsd
/*
```

次のタスク

アプリケーション・データを XML に変換したりその逆の変換を行ったりするためのアプリケーション・プログラムを作成します。両方の変換に同じマッピングを使用できます。Atom フィードの XML バインディングの作成を完了した後は、Atom フィードをセットアップするステップに進んでください。

関連概念

430 ページの『CICS XML 支援機能』
CICS XML 支援機能は、 XML と高水準言語構造との間の相互変換を支援する一連のバッチ・ユーティリティーです。この支援機能により、 XML 処理を実行するアプリケーションを最小限のプログラミングで迅速にデプロイすることが可能になります。

関連タスク

504 ページの『アプリケーション・データの XML への変換』
アプリケーション・プログラムを書き込み、データを XML に変換します。

506 ページの『アプリケーション・データへの XML の変換』
XML をアプリケーション・データに変換するためのアプリケーション・プログラムを作成します。変換前に XML を照会することもできます。

関連資料

431 ページの『DFHLS2SC: 高水準言語から XML スキーマへの変換』
カタログ式プロシージャーの DFHLS2SC は、XML スキーマと XML バインディング・ファイルを高水準言語構造から生成します。XML の構文解析または作成が可能な CICS プログラムを作成する場合は、DFHLS2SC を使用してください。

450 ページの『高水準言語および XML スキーマのマッピング』
CICS 支援機能は、高水準言語構造と XML スキーマまたは WSDL 文書との間のマッピングを生成する場合に使用します。また、CICS 支援機能は、高水準言語データ構造から XML スキーマまたは WSDL 文書を生成したり、 XML スキーマまたは WSDL 文書から高水準言語データ構造を生成したりします。

XML スキーマからのマッピングの生成

既存の XML スキーマに準拠する XML からアプリケーション・データを作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータを変換する方法を記述する必要があります。これは、XML スキーマまたは WSDL 文書から開始できます。

始める前に

有効な XML スキーマまたは WSDL 文書が必要です。マッピングを作成する前に、以下の前提条件が完了していることを確認する必要があります。

- 有効な XML スキーマまたは WSDL 文書が必要です。

- UNIX システム・サービスを使用するように、DFHSC2LS を実行するユーザー ID を構成する必要があります。
- このユーザー ID には、XML スキーマまたは WSDL 文書にアクセスするための読み取り権限と、z/OS UNIX の適切なディレクトリーに出力するための書き込み権限が必要です。
- このユーザー ID には、Java を実行するのに十分に大きなストレージが割り振られている必要もあります。

このタスクについて

CICS XML 支援機能を使用して、XML スキーマのデータ・マッピングを作成します。

1. DFHSC2LS バッチ・ジョブを実行します。 DFHSC2LS には、特定のコード・ページの選択や可変長文字データの処理方法の指定など、要件を満たすために選択できるオプション・パラメーターが用意されています。少なくとも以下のパラメーターを使用することをお勧めします。
   a. 入力ファイルの場所を指定します。 WSDL 文書または XML スキーマを使用できます。入力ファイルにインターネット上にある他のスキーマまたは文書への参照が含まれていて、システムがプロキシー・サーバーを使用している場合は、そのプロキシー・サーバーのドメイン名か IP アドレス、およびポート番号を指定します。
   b. 生成する高水準言語を指定します。 XML 支援機能では、COBOL、C、C++、および PL/I 言語がサポートされています。
   c. データ・マッピングをバンドル内に配置する場合は、**BUNDLE** パラメーターでバンドルの名前と場所を指定します。

   XML 支援機能により、サポートされる変換のライブラリーが XML バインディング内に作成されます。入力ファイル内のグローバル・エレメントごとに個別の変換が作成されます。

   BUNDLE パラメーターを指定した場合、バッチ・ジョブでは z/OS UNIX 上にバンドル・ディレクトリー構造が作成されます。バンドル・ディレクトリーには、バンドル・マニフェストが含まれる META-INF サブディレクトリーがあります。バッチ・ジョブではまた、バンドル・ディレクトリー内に XML バインディングが作成され、指定された場所に言語構造が保管されます。 XML 支援機能では、入力ファイルのコピーもバンドル・ディレクトリー内に保管されます。**BUNDLE** パラメーターを指定しない場合、バッチ・ジョブでは指定された場所に言語構造と XML バインディングのみが作成されます。
2. BUNDLE リソースをインストールします。 BUNDLE リソースによって XMLTRANSFORM リソースが動的に作成されます。このリソースにより、XML スキーマまたは WSDL 文書、XML バインディング、および言語構造のそれぞれの場所が定義されます。

タスクの結果

XML スキーマからマッピングを生成する場合、CICS ではスキーマに存在する各グローバル・エレメント用の言語構造が生成されます。

例

以下の例は、すべての必須パラメーターと推奨パラメーターが指定された DFHSC2LS を示しています。

```
//SC2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHSC2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
LANG=COBOL
BUNDLE=/u/exampleapp/bundle/test1
LOGFILE=/u/exampleapp/xsdbind/example.log
MAPPING-LEVEL=3.0
PDSLIB=//CICSHLQ.SDFHSAMP
PDSMEM=CPYBK2
XSDBIND=example.xsdbind
SCHEMA=example.xsd
/*
```

次のタスク

アプリケーション・データを XML に変換したりその逆の変換を行ったりするためのアプリケーション・プログラムを作成します。両方の変換に同じマッピングを使用できます。

関連概念

430 ページの『CICS XML 支援機能』
CICS XML 支援機能は、 XML と高水準言語構造との間の相互変換を支援する一連のバッチ・ユーティリティーです。この支援機能により、 XML 処理を実行するアプリケーションを最小限のプログラミングで迅速にデプロイすることが可能になります。

関連タスク

『アプリケーション・データの XML への変換』
アプリケーション・プログラムを書き込み、データを XML に変換します。

506 ページの『アプリケーション・データへの XML の変換』
XML をアプリケーション・データに変換するためのアプリケーション・プログラムを作成します。変換前に XML を照会することもできます。

アプリケーション・データの XML への変換

アプリケーション・プログラムを書き込み、データを XML に変換します。

始める前に

XML バインディングと XML スキーマを定義する、有効な XMLTRANSFORM リソースが必要です。

このタスクについて

XML 支援機能は、XML バインディング内のマッピングを生成します。言語構造から始めて DFHLS2SC を使用した場合、可能な XML への変換は 1 種類だけです。XML スキーマから始めた場合は、XML から言語構造への変換は複数の種類が可能であるため、生成する XML エレメントをアプリケーションで選択する必要があります。

1. アプリケーション・プログラムはチャネルを作成し、言語構造に関連付けられたデータをそのチャネルのビット・モード・コンテナーに格納する必要があります。
2. TRANSFORM DATATOXML API コマンドを使用して、データを XML に変換します。

```
EXEC CICS TRANSFORM DATATOXML
     XMLTRANSFORM('MyXmlTransformName')
     CHANNEL('MyChannelName')
     DATCONTAINER('SourceContainerName')
     XMLCONTAINER('TargetContainerName')
```

   XMLTRANSFORM リソースでは、サポートされている言語構造からの変換が 1 種類だけであるため、変換タイプをコマンドで指定する必要はありません。複数の種類の変換が可能である場合は、以下のオプションをアプリケーション・プログラムに追加してください。

```
ELEMNAME(elementName) ELEMNAMELEN (elementNameLength)
```

   これらの追加オプションは、アプリケーション・データが変換されて出力コンテナーに配置される XML エレメントを示します。
3. アプリケーション・プログラムをインストールします。

タスクの結果

アプリケーションが TRANSFORM DATATOXML コマンドを実行すると、CICS は XMLTRANSFORM リソースをチェックして XML バインディングのマッピングを検索し、チャネル上のコンテナーを使用してアプリケーション・バイナリー・データを XML に変換します。XML は、アプリケーションが XMLCONTAINER オプションで指定したコンテナーに格納されます。この XML は、XMLTRANSFORM リソースで定義されている XML スキーマに準拠しています。

次のタスク

同じマッピングを使用して XML をアプリケーション・データに変換することもできます。詳細については、506 ページの『アプリケーション・データへの XML の変換』を参照してください。

関連タスク

500 ページの『言語構造からのマッピングの生成』
XML をアプリケーション・データから作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータおよび XML を変換する方法を記述する必要があります。これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または DB2 レコードから開始できます。

502 ページの『XML スキーマからのマッピングの生成』
既存の XML スキーマに準拠する XML からアプリケーション・データを作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータを変換する方法を記述する必要があります。これは、XML スキーマまたは WSDL 文書から開始できます。

アプリケーション・データへの XML の変換

XML をアプリケーション・データに変換するためのアプリケーション・プログラムを作成します。変換前に XML を照会することもできます。

始める前に

XML バインディングと XML スキーマを定義する、有効な XMLTRANSFORM リソースが必要です。

このタスクについて

CICS の XML 支援機能は、XML バインディング内のマッピングを生成します。言語構造から始めて DFHLS2SC を使用した場合、可能な XML からの変換は 1 種類だけです。 XML スキーマから始めた場合は、XML から言語構造への変換は複数の種類が可能であるため、出力として使用する言語構造をアプリケーションで選択する必要があります。

1. チャネルを作成し、XML をそのチャネルでテキスト・モード・コンテナーに格納します。 XML がアプリケーションによって BIT モード・コンテナーに格納されると、 CICS はテキスト・データのエンコードを判別しようとしますが、コンテナーの処理に時間がかかり、エンコードが正しくならない場合があります。
2. TRANSFORM XMLTODATA API コマンドを使用します。 可能な XML の変換が 1 種類だけの場合は、以下のコマンドを使用することができます。

   ```
   EXEC CICS TRANSFORM XMLTODATA
         XMLTRANSFORM('MyXmlTransformName')
         CHANNEL('MyChannelName')
         XMLCONTAINER('SourceContainerName')
         DATCONTAINER('TargetContainerName')
   ```

   複数の種類の変換が可能な場合は、以下の追加のオプションを使用してください。

   ```
   EXEC CICS TRANSFORM XMLTODATA
         XMLTRANSFORM('MyXmlTransformName')
         CHANNEL('MyChannelName')
         XMLCONTAINER('SourceContainerName')
         DATCONTAINER('TargetContainerName')
         ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
   ```

   2 番目の例では、ELEMNAME オプション内で見つかったエレメントの名前が返されます。アプリケーションでそのエレメントの名前を使用して、ターゲット・コンテナーの内容を解釈するために使用する言語構造のライブラリーを判別できます。
3. アプリケーション・プログラムをインストールします。

タスクの結果

アプリケーションによって TRANSFORM XMLTODATA コマンドが実行されると、CICS は XMLTRANSFORM リソース内の詳細を使用して、 XML をチャネル上のコンテナーを使用するアプリケーション・バイナリー・データに変換します。

例

次のタスク

同じマッピングを使用してアプリケーション・データを XML に変換することもできます。詳細については、504 ページの『アプリケーション・データの XML への変換』を参照してください。

関連タスク

500 ページの『言語構造からのマッピングの生成』
XML をアプリケーション・データから作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータおよび XML を変換する方法を記述する必要があります。これは、どのアプリケーション・データ・レコードからでも開始できます。例えば、COMMAREA、VSAM ファイル、一時ストレージ・キュー、または DB2 レコードから開始できます。

502 ページの『XML スキーマからのマッピングの生成』
既存の XML スキーマに準拠する XML からアプリケーション・データを作成したりその逆を行ったりする場合には、マッピングを作成して、CICS が実行時にデータを変換する方法を記述する必要があります。これは、XML スキーマまたは WSDL 文書から開始できます。

アプリケーションからの XML の照会

XML のフラグメントを照会してからアプリケーション・データに変換するアプリケーション・プログラムを作成することができます。

このタスクについて

アプリケーションでさまざまなタイプの XML を処理する場合は、XML を照会して、アプリケーション・データへの変換に使用する XMLTRANSFORM リソースを判別することもできます。このコマンドは、XML に <xsd:any> エレメントが含まれている場合にも役立つことがあります。

1. アプリケーション・プログラムで XML を照会するには、以下のような TRANSFORM XMLTODATA API コマンドを使用します。

   ```
   EXEC CICS TRANSFORM XMLTODATA
        CHANNEL('MyChannelName')
        XMLCONTAINER('SourceContainerName')
        ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
   ```

   チャネルの名前と、XML を保持するコンテナーの名前を指定する必要があります。XML を照会するために XMLTRANSFORM リソースを指定する必要はありません。上記の例では、最初の XML エレメントの名前とその XML エレメントの長さが照会されます。最初の XML エレメントのタイプ、タイプの長さ、およびタイプの名前空間を照会することもできます。
2. オプション: アプリケーションで XML エレメントの名前空間が必要な場合は、CICS が ELEMNS 値を書き込めるデータ域を提供してください。
3. オプション: XML を照会した後は、XML をアプリケーション・データに変換するために使用する XMLTRANSFORM リソースを判別するアプリケーション・ロジックを作成できます。
4. CICS でアプリケーション・プログラムをインストールします。

タスクの結果

CICS は指定されたコンテナーを読み取り、XML エレメントについての情報をアプリケーション・プログラムに返します。

データ型による XML の取り扱い

XML スキーマにグローバル・データ型が含まれていて、そのうちの 1 つ以上が XML で参照される場合は、これらのグローバル・データ型をサポートするメタデータを生成した後、アプリケーション・プログラムで XML を解析または変換することができます。

始める前に

適切なメタデータを生成するには、TYPES=ALL パラメーターを指定して DFHSC2LS を実行する必要があります。

このタスクについて

1. アプリケーション・データをデータ型により XML に変換するには、以下のような TRANSFORM DATATOXML コマンドを使用します。

```
EXEC CICS TRANSFORM DATATOXML
     XMLTRANSFORM('MyXmlTransformName')
     CHANNEL('MyChannelName')
     DATCONTAINER('SourceContainerName')
     XMLCONTAINER('TargetContainerName')
     ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
     ELEMNS(elementNamespace) ELEMNSLEN(elementNamespaceLength)
     TYPENAME(typeName) TYPENAMELEN(typeNameLen)
     TYPENS(typeNamespace) TYPENSLEN(typeNamespaceLen)
```

   MyXmlTransformName は、XML バインディングおよびスキーマを指定する XMLTRANSFORM リソースの 32 文字の名前です。*MyChannelName* は、入力および出力コンテナーのあるチャネルの 16 文字の名前です。*SourceContainerName* は、アプリケーション・データを保持する入力コンテナーの 16文字の名前であり、*TargetContainerName* は、CICS が XML で追加する出力コンテナーの 16 文字の名前です。XML エレメント名、名前空間、データ型、および変換用のデータ型の名前空間を指定する必要もあります。

2. データ型により XML を解析するには、TRANSFORM XMLTODATA コマンドを使用します。 このコマンドで指定するオプションは XML に xsi:type 属性があるかどうかによって異なります。

   - XML で xsi:type 属性が使用されている場合、アプリケーション・プログラムで以下のコマンドを指定します。

```
EXEC CICS TRANSFORM XMLTODATA
     XMLTRANSFORM('MyXmlTransformName')
     CHANNEL('MyChannelName')
     XMLCONTAINER('SourceContainerName')
     DATCONTAINER('TargetContainerName')
     ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
     TYPENAME(typeName) TYPENAMELEN(typeNameLength)
```

     アプリケーションでデータ型の名前空間も必要とされる場合、API コマンドに TYPENS オプションを追加します。データ型は xsi:type 属性から TYPENAME に返されます。

- XML で xsi:type 属性が使用されていない場合、アプリケーション・プログラムでグローバル・データ型のローカル名および名前空間を指定することができます。アプリケーション・プログラムで以下のコマンドを使用してください。

```
EXEC CICS TRANSFORM XMLTODATA
     XMLTRANSFORM('MyXmlTransformName')
     CHANNEL('MyChannelName')
     XMLCONTAINER('SourceContainerName')
     DATCONTAINER('TargetContainerName')
     TYPENAME(typeName) TYPENAMELEN(typeNameLength)
     TYPENS(TypeNamespace) TYPENSLEN(typeNamespaceLen)
```

  アプリケーション・プログラムでは、名前および名前空間ではなく型名および名前空間を指定する必要があります。このコマンドは、アプリケーションが CICS に使用するデータ型を通知することを示しています。CICS が XML を生成している場合は、常に xsi:type 属性が追加されます。

3. CICS でアプリケーション・プログラムをインストールします。

次のタスク

アプリケーション・プログラムが XML を予期したとおりに生成および解析するかどうかをテストします。

<xsd:any> データ型の処理

1 つ以上の <xsd:any> データ型が含まれる XML スキーマを処理する場合は、XML 支援機能でデータ型を 1 対の CICS コンテナーにマップすることができます。コンテナー内の XML を解析するアプリケーション・プログラムを作成できます。

始める前に

マッピング・レベル 2.1 以上で DFHSC2LS または DFHWS2LS を使用して XML スキーマをマップする必要があります。

このタスクについて

CICS でデータを XML に変換すると、<xsd:any> データ型と関連付けられた XML が、最初のコンテナーと、2 番目のコンテナーの有効範囲内にある名前空間接頭部宣言に格納されます。

1. この XML データを解析するには、アプリケーション・プログラムで TRANSFORM XMLTODATA コマンドを使用します。

```
EXEC CICS TRANSFORM XMLTODATA
     XMLTRANSFORM('MyXmlTransformName')
     CHANNEL('MyChannelName')
     XMLCONTAINER('SourceContainerName')
     DATCONTAINER('TargetContainerName')
     NSCONTAINER('NamespacesContainerName')
     ELEMNAME(elementName) ELEMNAMELEN(elementNameLength)
```

  MyXmlTransformName は、XML バインディングおよびスキーマを指定する XMLTRANSFORM リソースの 32 文字の名前です。*MyChannelName* は、入力および出力コンテナーのあるチャネルの 16 文字の名前です。*SourceContainerName* は、XML を保持する入力コンテナーの 16文字の名前であり、*TargetContainerName* は、CICS がアプリケーション・データで追加する出力

コンテナーの 16 文字の名前です。*NamespacesContainerName* は、 CICS が名前空間接頭部宣言で追加するコンテナーの 16 文字の名前です。 ELEMNAME および ELEMNAMELEN オプションの初期値を提供してください。 CICS は XML エレメントと、ELEMNAME および ELEMNAMELEN オプション内にあるその長さを返します。

2. CICS でアプリケーション・プログラムをインストールします。

次のタスク

アプリケーション・プログラムが XML を正常に解析するかどうかをテストします。

関連資料

486 ページの『<xsd:any> と xsd:anyType のサポート』
DFHWS2LS と DFHSC2LS は、XML スキーマでの <xsd:any> と xsd:anyType の使用をサポートします。<xsd:any> XML スキーマ・エレメントを使用して、内容が未定義の XML 文書のセクションを記述できます。xsd:anyType は、すべての単素数データ型および複素数データ型が派生する基本データ型です。データ内容に対する制限や制約はありません。

XML 変換の妥当性検査

アプリケーション・データをマップするために CICS XML 支援機能を使用する場合、実行時に行う変換が妥当性検査されるように指定して、変換が XML バインディングに含まれるスキーマに準拠することを保証できます。XML からバイナリー・データまたはバイナリー・データから XML への変換に対して、妥当性検査を実行することができます。

始める前に

CICS アプリケーションの開発およびテストの間に完全な妥当性検査を行うと、XML の問題を検出するのに役立ちます。ただし、 XML の完全な妥当性検査ではかなりのオーバーヘッドが発生するため、完全にテストされた実動アプリケーションで XML を妥当性検査することはお勧めできません。

CICS では、スキーマに対して XML を妥当性検査するために Java プログラムが使用されます。したがって、妥当性検査を実行するには、CICS 領域で Java サポートが有効になっている必要があります。

このタスクについて

XML を妥当性検査する手順は以下のとおりです。

1. XML バインディングとスキーマが z/OS UNIX 上の同じ場所にあることを確認します。 XMLTRANSFORM リソースで、これらのファイルを CICS に定義します。INQUIRE XMLTRANSFORM コマンドを使用すると、各ファイルの場所を確認することができます。
2. アプリケーションの妥当性検査をオンにします。 CEMT または SPI コマンド SET XMLTRANSFORM(name) VALIDATION を使用します (*name* は XMLTRANSFORM リソースです)。

タスクの結果

システム・ログを確認して、XML 変換が有効であるかどうかを調べます。メッセージ DFHML0508 は XML が正常に妥当性検査されたことを示し、メッセージ DFHML0507 は妥当性検査が失敗したことを示します。

次のタスク

アプリケーション用に XML を妥当性検査する必要がなくなった場合は、 SET XMLTRANSFORM(name) NOVALIDATION コマンドを使用して妥当性検査をオフにしてください。

第 6 部 ビジネス・サービスおよびバンドル

CICS アプリケーションからサービス指向アーキテクチャーに参加するビジネス・サービスを作成してバンドル内に配置します。

© Copyright IBM Corp. 1989, 2009

第 29 章 CICS アプリケーションからのビジネス・サービスの作成

ビジネス・サービスであるアプリケーションは、サービス指向アーキテクチャー (SOA) に参加できます。ビジネス・サービス は、技術的な実装よりも、ビジネス・プロセスやモデルと連携するサービスです。既存または新規の CICS アプリケーションは、CICS で提供されるサポートを使用することで、ビジネス・サービスの一部として公開できます。

CICS アプリケーションから 2 種類のサービスを作成できます。

チャネル・ベース・サービス
このサービスは、Service Component Architecture (SCA) サポートを使用し、アプリケーションをサービス・コンポーネントとして公開します。このサービスのインターフェースはチャネルです。チャネル・ベース・サービスは、INVOKE SERVICE API コマンドを使用した他の CICS アプリケーションでのみ呼び出すことができます。

XML ベース・サービス
このサービスは、通常、XML を使用して他のアプリケーションとやり取りし、バインディングを使用してデータ変換を行う Web サービス・プロバイダーまたはリクエスターのアプリケーションです。SCA を使用して、Web サービス・アプリケーションをコンポーネントとして記述することもできます。このサービスのインターフェースは、データを変換するためのバインディングを含む XML です。XML ベース･サービスは、INVOKE SERVICE API コマンドを使用した他の CICS アプリケーション、または外部クライアントで呼び出すことができます。

XML ベース・サービスは、TRANSFORM API コマンドを使用して XML のアプリケーション・データをマップするアプリケーションになることもできます。XML 支援機能は、言語構造または XML スキーマを使用して、XML バインディングの生成に加え、バンドルの作成も行います。

両方のサービス・タイプで CICS のパイプライン・サポートが使用されます。

Service Component Architecture (SCA)

Service Component Architecture (SCA) は、Service-Oriented Architecture (SOA) を使用してアプリケーションおよびシステムを構築するためのプログラミング・モデルが記述された一連の仕様です。SCA は従来のサービス実装方法を拡張および補完するものであり、 Web サービスなどのオープン・スタンダードを基盤としています。

この仕様は、*複合アプリケーション*の作成方法を記述しています。複合アプリケーションは、 1 つ以上のコンポーネントを結合することで作成されます。これにより、新しいアプリケーションのビジネス・ロジックが実装されます。コンポーネントは、ビジネス・ロジックと構成情報を実装するアプリケーション・プログラムで構成されます。アプリケーション開発者は、同じアプリケーション・プログラムを

© Copyright IBM Corp. 1989, 2009

異なる構成で使用し、さまざまなコンポーネントを作成できます。コンポーネントは、サービス指向のインターフェースを使用して、他のコンポーネントにサービスを提供したり、またはその反対に他のサービスから提供される機能を取り込んだりします。

アプリケーション開発者は、コンポーネントをまとめて組み立てることで、特定のビジネス要件に合ったソリューションを作成できます。複合アプリケーションには、業務アプリケーション用に特別に作成した新規コンポーネントおよび他のアプリケーションを再利用した既存のコンポーネントの両方を含めることができます。

CICS は、「*SCA Assembly Model 1.0*」 仕様をサポートします。この仕様は、サービス・コンポーネントを組み立ててコンポジット を形成する方法を記述します。コンポジットとは、SCA における実装単位のことで、SCDL と呼ばれる XML 言語で記述されます。コンポジットには、コンポーネント、サービス、参照、プロパティー宣言、およびこれらのエレメント間の関連付けを記述するワイヤリングを含めることができます。コンポジットは、他のコンポジットとともにコンポーネントで使用することも可能なので、複合アプリケーションを階層的な構造にし、一連の下位サービスによって高位サービスを内部に実装することができます。

コンポーネントの構造

単純なタイプのコンポーネントには、1 つのサービスと 1 つの参照が含まれます。サービス は、1 つ以上の操作を含めることができる、コンポーネント用のアドレッシング可能なインターフェースです。参照 は、別のコンポーネントから提供されるサービスとの依存関係を表します。コンポーネントのバインディングはサービスと参照の両方で定義できます。

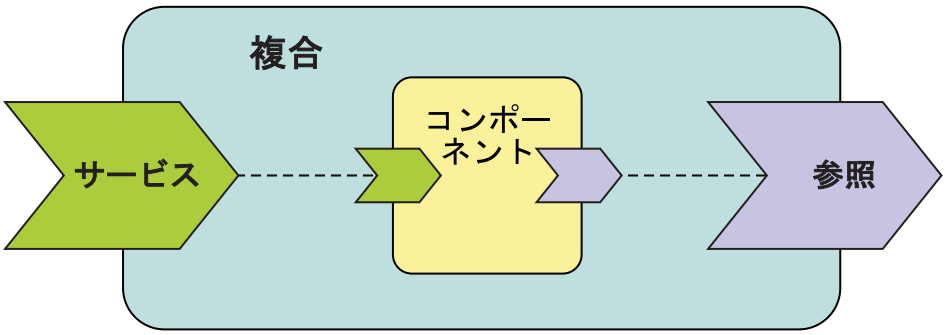
- コンポーネント・サービスはバインディングを使用し、他の内部コンポーネントがコンポーネントを呼び出すときに使用する必要のあるアクセス機構を記述します。
- コンポーネント参照はバインディングを使用し、他のコンポーネントの呼び出しに使用されるアクセス機構を記述します。

コンポーネント・サービスおよびコンポーネント参照は内部に実装されており、コンポーネント対コンポーネントの通信にのみ使用されます。外部インターフェースを作成するには、コンポーネントがコンポジット内部にデプロイされる必要があります。非常に単純なコンポジットには、1 つの外部サービスと 1 つの外部参照が含まれます。

- コンポジット・サービスはバインディングを使用し、外部クライアントがサービスを呼び出すときに使用する必要のあるアクセス機構を記述します。
- コンポジット参照はバインディングを使用し、別のサービスの呼び出しに使用されるアクセス機構を記述します。

コンポジット・サービスおよびコンポジット参照は、コンポーネント自体の一部ではありませんが、コンポーネントと他の外部クライアントまたは外部サービスとの間の境界の役割を果たします。

以下の図は、内部および外部の両方のサービスと参照のある単純なコンポーネントを含むコンポジットを示しています。



SCA コンポジットおよびワイヤリング

アプリケーション開発者は、サービスと参照をワイヤリングでまとめることで、SCA コンポジットを作成します。ワイヤーとは、コンポーネントからターゲットへ制御やデータを渡すコネクターです。

Rational Developer for System z を使用して、複数のコンポーネントを 1 つのコンポジットにワイヤリングするには、以下の 2 つの方法を使用できます。

プロモーション

コンポジット内のいずれかのコンポーネントで定義されているコンポーネント・サービスをプロモートすることで、コンポジット・サービスにワイヤリングできます。同様に、コンポジット参照をコンポーネント参照にプロモートできます。コンポジット・サービスを別のコンポジット・サービスにプロモートしたり、コンポジット参照を別のコンポジット参照にプロモートしたりすることはできません。サービスまたは参照にバインディングを追加すると、プロモートされたコンポジット・サービスまたは参照のバインディング属性が、コンポーネント・サービスまたは参照の属性に優先します。

外部サービスが内部サービスとは異なる種類のバインディングを使用している場合、その内部サービスをプロモートすることはできません。例えば、外部サービスが CICS バインディングを使用している場合、内部サービスは Web サービス・バインディングを使用することはできません。

ターゲット

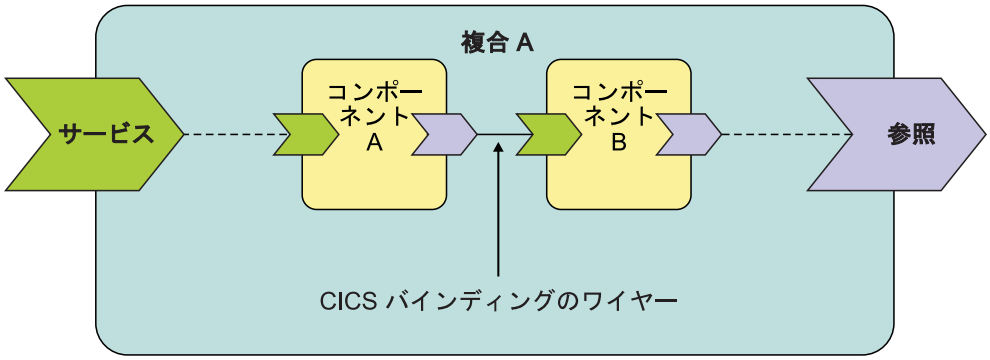
あるコンポーネント参照を別のコンポーネント・サービスに直接ワイヤリングすることができます。これを行うには、参照でターゲット属性を指定し、それら 2 つのコンポーネントをつないでコンポジットにまとめます。このワイヤリングは、コンポジットの内部で行われます。同様に、コンポジット参照でターゲット属性を指定することで、2 つのコンポジットをワイヤリングすることもできます。通常、この種のワイヤリングは複雑なコンポジットで行われます。

単純なコンポジット

コンポジットには、相互につながれた複数のコンポーネントを含めることができます。以下の例では、コンポジット A に 2 つのコンポーネントが含まれています。コンポーネント A は、サービス実行のためにコンポーネント B に依存しています。各コンポーネントには 1 つのサービスと 1 つの参照があります。コンポーネ

ント A の参照に定義されたバインディングは、コンポーネント B のサービスと互換性がある必要があります。コンポジット A は、2 つのコンポーネントをカプセル化し、コンポジット A で提供されるサービスを要求するアプリケーションから下位実装の詳細を隠します。

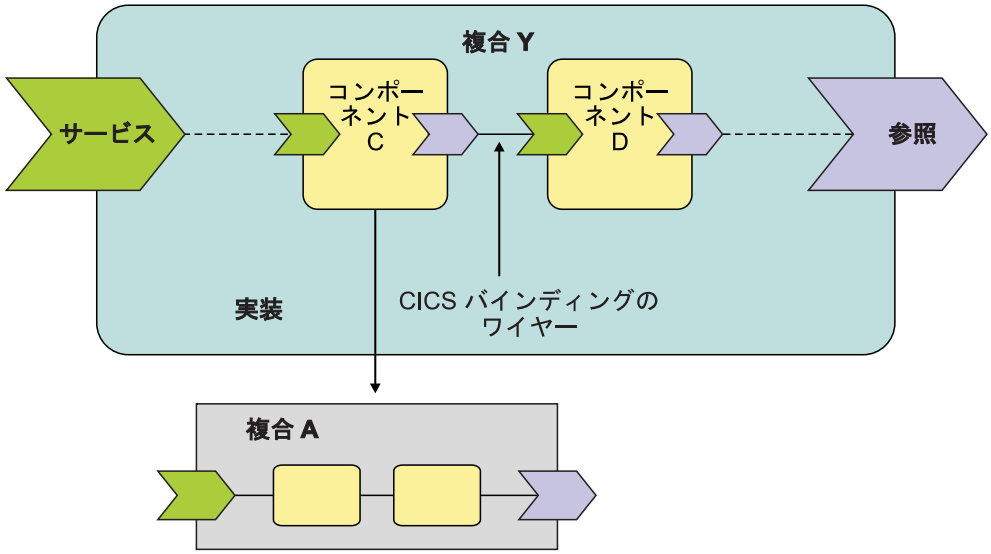
コンポジットの外部サービスと外部参照は、コンポーネント A のサービスとコンポーネント B の参照をプロモートすることで作成されます。コンポーネント A と B のワイヤーはターゲットであり、各インターフェースのバインディングは同じです。この例では、コンポーネントは CICS バインディングを使用しています。



複雑なコンポジット

コンポーネント内の実装内容をカプセル化するモデルを使用すると、階層構成の複合アプリケーションを極めて柔軟に作成することができます。以下の例では、コンポジット Y に 2 つのコンポーネントが含まれています。ただし、この例の場合、コンポーネント C の実装自体が、内部に 2 つのコンポーネントを含んだコンポジットとなっています。コンポジット Y によって提供されるサービスを呼び出すクライアントが、コンポジット C に含まれるコンポーネントを知る必要はありません。そのため、クライアントに影響を及ぼすことなく、基盤となる実装の詳細を変更できます。

コンポジット Y の外部サービスと外部参照は、コンポーネント C のサービスとコンポーネント D の参照をプロモートすることで形成されます。コンポーネント C と D のワイヤーはターゲットであり、各インターフェースのバインディングは同じです。この例では、コンポーネントは CICS バインディングを使用しています。



コンポジットの作成およびデプロイのベスト・プラクティス

コンポーネントを複合アプリケーションに組み立てるためのモデルは柔軟性に優れています。アプリケーション・コンポジットを複数の CICS 領域で再利用するときは、アプリケーションを開発およびテストから実稼働環境へ移動する場合でも、または実稼働環境下でアプリケーションのクローンを作成する場合でも、アプリケーション・ロジックをバインディングから分離することをお勧めします。

実装ロジックをコンポジット・サービスおよび参照から分離する

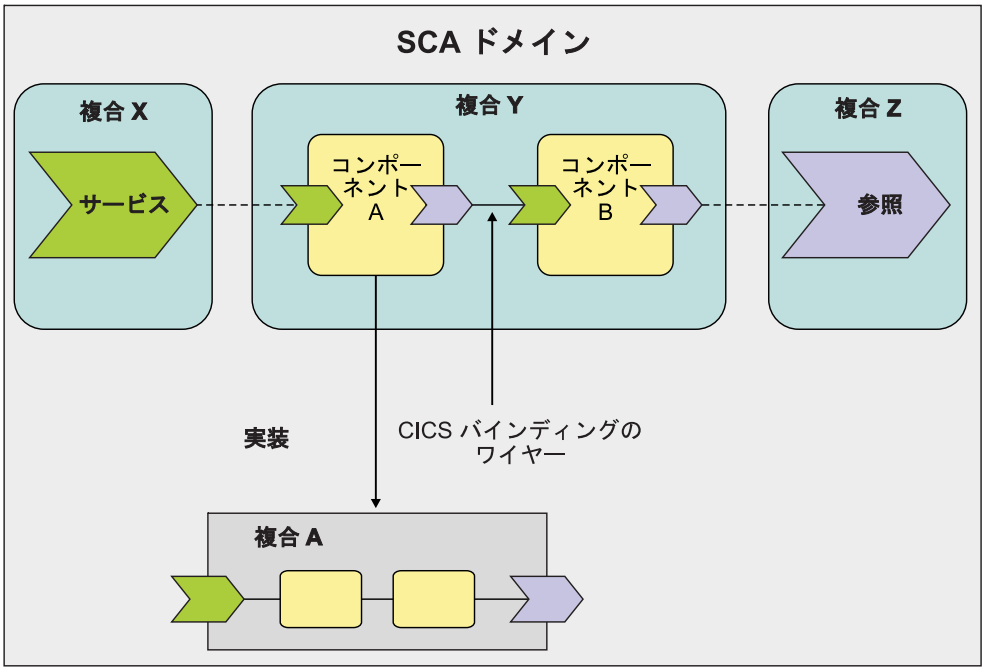
異なる CICS 領域で同じアプリケーション・ロジックをさらに簡単に再利用するには、コンポジット・サービスおよび参照をアプリケーションの実装ロジックから分離し、それらを個別のバンドルとしてデプロイします。コンポジット・サービスおよび参照には、特定のシステム・リソース・セットまたはトランザクション ID など、 CICS 領域固有の情報を入れることができます。この方法を使用することで、アプリケーション・コンポジットを再デプロイすることなく、コンポジット・サービスおよび参照の変更または更新ができます。

バンドルの SCA ドメインを定義する

バンドルは、BUNDLE リソースによって CICS に示されます。デフォルトでは、CICS にデプロイされたすべてのバンドルに同じ SCA ドメインがありますが、値は空です。BUNDLE リソースには、システム・プログラマーが SCA ドメインを表す絶対名の設定に使用できる、BASESCOPE というオプションの属性が含まれています。サービスと参照を複合アプリケーションから分離し、個別のバンドルとしてデプロイする推奨モデルに従う場合、バンドルが関連付けられていることを示すために、同じ BASESCOPE 値が各 BUNDLE リソースで使用されるように要求できます。

また、異なる SCA ドメインを BASESCOPE 属性に指定することで、同じバンドルを CICS 領域に何度もデプロイすることができます。CICS は、ランタイム処理中に SCA ドメインとコンポジットをともに使用してサービス

を特定します。要求を処理中のタスクは、サービスの有効範囲を使用できます。BASESCOPE の値には固有の URI を指定することをお勧めします。



チャネル・ベース・サービスの作成

チャネル・ベース・サービスとは、 Rational Developer for System z などのツールを使用してコンポーネントとして記述され、まとめてアセンブルされる CICS アプリケーションのことです。これらのサービスは、INVOKE SERVICE API コマンドを使用してコンテナー内のバイナリー・データをチャネルで渡す、他の CICS アプリケーションにのみ使用できます。

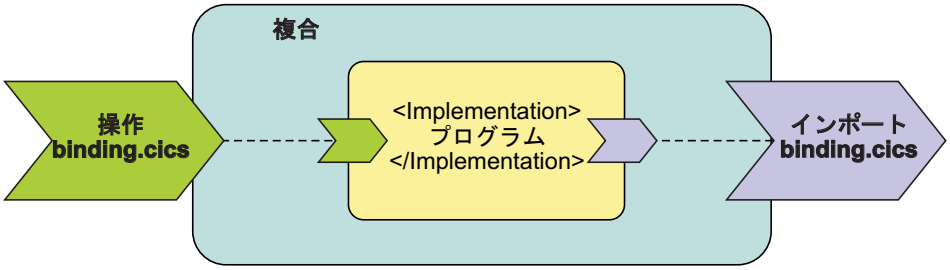
このタスクについて

SCA では、チャネル・ベース・サービスとして公開するアプリケーション・プログラムはビジネス・ロジックの実装となります。コンポーネントのサービスは、アプリケーション・プログラムへのインターフェースを WSDLで記述します。コンポーネントには、操作とバインディングを含むことのできるインターフェースが 1 つあります。チャネル・ベース・サービスでは、CICS バインディングがチャネルを記述します。このバインディングを選択すると、Rational Developer for System z により、SCDL に binding.cics セクションが追加されます。アプリケーション・インターフェースは、チャネルおよびバイナリー・データがコンテナーに含まれていることを想定しているため、チャネル・ベース・サービスにはデータ・マッピングがありません。

1. Rational Developer for System z を使用し、コンポジットを作成します。
   a. コンポーネントの実装として、アプリケーション・プログラム名を指定します。

b. ツールで direct マッピング・モードを選択することにより、適切なサービスまたは参照に CICS バインディングを追加します。
   - CICS バインディングで、コンポジット・サービスに接続する URI を指定すると、この URI によってサービスの外部名が設定されます。CICS では、この URI は相対パス名になります。例えば、サービスが http://myhost:port/myService として公開される場合、バインディングの URI は myService になります。これは、ホストとポートが複合アプリケーションによる制御の対象外であるためです。
   - CICS バインディングで、コンポジット参照に接続する URI を指定すると、URI はターゲット先サービスへの絶対パスになります。例えば、http://myhost:port/myService または cics://PROGRAM/prog1?user=user1 のようになります。

c. オプション: コンポジット参照でインポートを定義します。 インポートは、コンポーネントまたはコンポジットの依存関係を定義します。



CICS でコンポジットを作成およびデプロイするための推奨方法については、519 ページの『コンポジットの作成およびデプロイのベスト・プラクティス』を参照してください。

2. CICS に、コンポジットをバンドルとしてデプロイします。 Rational Developer for System z は、バンドル・マニフェストを生成し、SCDL やその他の成果物をパッケージ化します。マニフェストは、CICS が BUNDLE リソースを正常にインストールするために必要なすべてのリソースとメタデータを記述します。BUNDLE リソースは、CICS 領域のコンポジットを表します。マニフェストは、コンポジットを SCACOMPOSITE リソース・タイプとして定義し、相対パスによって SCDL の位置を参照します。
3. BUNDLE リソースを作成およびインストールします。 オプションで、BUNDLE リソース定義で SCA ドメインを設定できます。通常、SCA ドメインは、ビジネス機能の領域を提供するサービス・セットを表します。異なる SCA ドメインを使用して、同じバンドルをインストールすることができます。これは、CICS が SCA ドメインとサービス名を結合することでサービスを識別するためです。SCA ドメインの追加方法に関する詳細とその例については、「*CICS Resource Definition Guide*」を参照してください。

   BUNDLE リソースを正常にインストールするには、CICS 領域でバンドルのすべての前提条件が満たされていることを確認する必要があります。
4. INVOKE SERVICE API コマンドを使用し、チャネル・ベース・サービスを呼び出すアプリケーションを記述します。

```
EXEC CICS INVOKE SERVICE('servicename')
          CHANNEL(channel)
          OPERATION(operation)
```

servicename はサービスの外部名、*channel* はチャネルの 16 バイトの名前、*operation* は任意の値を指します。オペレーションはコマンドの必須オプションですが、その値はチャネル・ベース・サービスに使用されません。

タスクの結果

アプリケーションがチャネル・ベース・サービスを呼び出すと、CICS はサービスの名前を解決して EXEC CICS LINK コマンドを発行し、コンポジットで定義したアプリケーション・プログラムに対して指定のチャネルとコンテナーを渡します。バインディングで特定のリクエスター・パイプラインを指定した場合、CICS はそのリクエスター・パイプラインを使用して要求を実行します。バインディングでリクエスター・パイプラインを指定していない場合、CICS はその要求のリクエスター・パイプラインを動的に作成します。

次のタスク

サービスが予期したとおりに機能するか、テストおよび検証することができます。IBM CICS Explorer を使用し、BUNDLE リソースとその内容を表示できます。BUNDLE リソースを有効および無効にして、リソースすべてをまとめて管理することもできます。

XML ベース・サービスの作成

XML ベース・サービスは、XML を使用して他のアプリケーションとやり取りし、バインディングを使用してデータ変換を行う Web サービス・プロバイダーまたはリクエスターのアプリケーションです。XML ベース・サービスは、 INVOKE SERVICE API コマンドを使用する CICS アプリケーションまたは外部ネットワーク上のビジネス・サービスで使用できます。

このタスクについて

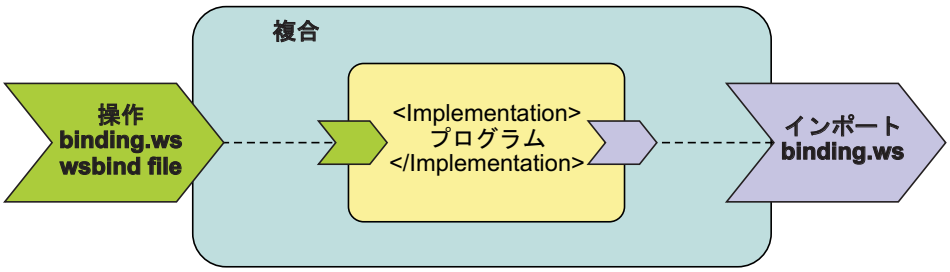
CICS の Web サービス・サポートを使用して Web サービスを作成することも、Rational Developer for System z を使用することもできます。Rational Developer for System z を使用する場合は、Web サービスから SCA コンポーネントも作成できます。Web サービスからコンポーネントを作成することには、以下のような利点があります。

- Rational Developer for System z を使用することで、既存のコンポーネントをさらに簡単に再利用し、新しい複合アプリケーションを迅速に開発できます。
- SCDL を使用して Web サービスを記述し、アプリケーションを変更せずに、アプリケーションからより変更しやすいメタデータに構成情報を移動できます。例えば、異なるトランザクションとユーザー ID で Web サービスを実行する場合、Web サービス・バインディング・ファイルを再生成することなく、SCDL を変更できます。

SCA では、Web サービスとして公開するアプリケーション・プログラムはビジネス・ロジックの実装となります。アプリケーション・プログラムは、コンポーネントの <Implementation> エレメントで定義されます。コンポーネントのサービスは、

アプリケーション・プログラムへのインターフェースを WSDLで記述します。コンポーネントには、多数の操作とバインディングを含むことのできるインターフェースが 1 つあります。

1. Rational Developer for System z を使用し、コンポジットを作成します。
   a. コンポーネントの実装として、アプリケーション・プログラム名を指定します。
   b. CICS 外でもサービスを使用できるようにするには、Web サービス・バインディングを適切なサービスまたは参照に追加します。サービスを使用できる対象を CICS アプリケーションだけにする場合、CICS バインディングを適切なサービスまたは参照に追加し、ツールで `mapped` マッピング・モードを選択します。 ツールは、SCDL でバインディングを記述します。SCDL には、XML を適切な高水準言語に変換するデータ・マッピングも含まれます。
   c. コンポジット参照でインポートを定義します。 インポートは、CICS 領域で満たされる必要があるコンポジットの依存関係を定義します。



CICS でコンポジットを作成およびデプロイするための推奨方法については、519 ページの『コンポジットの作成およびデプロイのベスト・プラクティス』を参照してください。

2. CICS に、コンポジットをバンドルとしてデプロイします。 Rational Developer for System z は、バンドル・マニフェストを生成し、SCDL やその他の成果物をパッケージ化します。マニフェストは、CICS が BUNDLE リソースを正常にインストールするために必要なすべてのリソースとメタデータを記述します。BUNDLE リソースは、CICS 領域のコンポジットを表します。マニフェストは、コンポジットを SCACOMPOSITE リソース・タイプとして定義し、相対パスによって SCDL の位置を参照します。
3. BUNDLE リソースを作成およびインストールします。 オプションで、BUNDLE リソース定義で SCA ドメインを設定できます。通常、SCA ドメインは、ビジネス機能の領域を提供するサービス・セットを表します。異なる SCA ドメインを使用して、同じバンドルをインストールすることができます。これは、CICS が SCA ドメインとサービス名を結合することでサービスを識別するためです。SCA ドメインの追加方法に関する詳細とその例については、 「*CICS Resource Definition Guide*」を参照してください。

   BUNDLE リソースを正常にインストールするには、CICS 領域でバンドルのすべての前提条件が満たされていることを確認する必要があります。 BUNDLE リソースが正常にインストールされると、CICS はマニフェストと SCDL からの情報を使用して WEBSERVICE および URIMAP リソースを作成します。また、CICS は、「参照 (Reference)」の「インポート (Imports)」セクションで定義され

たリソースが CICS 領域に存在しているかどうかも確認します。「インポート (Imports)」は、アプリケーションの前提条件を定義します。

タスクの結果

Web サービスが正常に CICS にインストールされます。

次のタスク

Web サービスが予測したとおりに機能するかどうかをテストおよび検証できます。IBM CICS Explorer を使用し、BUNDLE リソースとその内容を表示できます。BUNDLE リソースを有効および無効にして、リソースすべてをまとめて管理することもできます。

CICS によるサービスの処理

アプリケーションがサービスを呼び出す際 (Web サービス・リクエスターが SOAP メッセージを送信する場合や、CICS アプリケーションが INVOKE SERVICE API コマンドを使用する場合など)、CICS はパイプラインを使用して要求を処理します。

CICS は、サービス・リクエスターおよびサービス・プロバイダー・パイプラインの XML ベース・サービスを処理します。各パイプラインのタイプには、要求をネットワークで送信できるトランスポート・ハンドラーがあります。CICS アプリケーションは、外部クライアントに対しては Web サービス・リクエスターまたは Web サービス・プロバイダーになることができ、HTTP、HTTPS、JMS、または WMQ プロトコルを使用して要求を送受信します。

Web サービスがどちらも CICS アプリケーションである場合、サービス・リクエスターは INVOKE SERVICE コマンドおよび `cics://` で始まる URI を使用して、パイプライン処理を最適化できます。バイナリー・データを XML に変換するデータ変換は通常どおりに実行され、パイプラインはメッセージ・ハンドラーを介して実行されます。パイプライン・トランスポート・ハンドラーは、指定された URI のタイプに応じて別のプログラムにリンクするか、またはパイプラインを開始します。

例えば、URI `cics://SERVICE/`*`service`*`?targetServiceUri=`*`targetServiceUri`* を使用すると、ネットワークをわざわざ使用しなくても、プロバイダー・パイプラインを介して要求を実行できます。CICS は *service* 値および *targetServiceUri* 値を使用し、URIMAP で要求を解決します。

この URI は、リクエスター・アプリケーションおよびプロバイダー・アプリケーションが異なる言語で作成されているときや異なるマッピング・レベルを使用している場合、および異なるバイナリー・データを想定している場合に役立ちます。

CICS は、EXEC CICS LINK PROGRAM を使用し、SCDL でサービス向けに定義されているプログラムに、チャネルおよびコンテナーを渡すことで、チャネル・ベース・サービスへの呼び出しを処理します。呼び出しアプリケーションが `cics://` で始まる URI も指定する場合、CICS はリクエスター・パイプラインで追加の処理を実行できます。バインディングでリクエスター・パイプラインが指定されていない場合、CICS はその要求のリクエスター・パイプラインを動的に作成します。

URI フォーマット

URI が cics:// で開始する場合、パイプライン・トランスポート・ハンドラーはプログラムにリンクするか、別のリクエスターのパイプラインまたはプロバイダーのパイプラインを開始できます。URI には 3 つのタイプがあります。

cics://PROGRAM/*program*
: この URI タイプのパイプライン処理は、同じ CICS 領域内の Web サービス・リクエスターおよびプロバイダーに対して行われるローカル最適化に似ています。*program* は、トランスポート・ハンドラーが、リクエスター・パイプライン処理の終了時にリンクするプログラムの名前を定義します。トランスポート・ハンドラーは EXEC CICS LINK PROGRAM コマンドを使用して、チャネルとそのコンテナーを指定のプログラムに渡します。URI に追加のオプションが示されている場合、CICS はプログラムに COMMAREA を渡すこともできます。

cics://SERVICE/*service***?targetServiceUri=***targetServiceUri*
: この URI タイプのパイプライン処理は、ネットワーク経由の送信を行わないことで、要求を最適化します。ただし、要求は、リクエスターおよびプロバイダーの両方のパイプラインを介して実行されます。*service* 値は、指定のプログラムではなく、サービスの名前を定義します。*targetServiceUri* 値は、URIMAP リソースによって解決されるサービスのパスを定義します。トランスポート・ハンドラーは URIMAP を使用して、同じ CICS 領域内の適切なプロバイダー・パイプラインに要求を送信します。

cics://PIPELINE/*pipeline***?targetServiceUri=***targetServiceUri*
: *pipeline* 値は、リクエスター・パイプラインの名前を定義し、トランスポート・ハンドラーは指定されたリクエスター・パイプラインを開始する前に、*targetServiceUri* の値を DFHWS-URI に入力します。この URI タイプは、多数のリクエスター・パイプラインをまとめてチェーニングし、要求をさまざまなメッセージ・ハンドラーのセットで処理できるようにします。

一方、パイプラインのメッセージ・ハンドラーは、アプリケーションで提供された URI を指定変更し、パイプラインの処理を制御することができます。

これらの URI で指定できるパラメーターの詳細については、「*CICS Web サービス・ガイド*」を参照してください。

SCA の問題のトラブルシューティング

CICS での SCA 複合アプリケーションの実装時に起こる可能性がある問題は、CICS がサービス要求を処理している間の配置プロセス中または実行時に発生することがあります。

このタスクについて

エラーが発生すると CICS は、接頭部が DFHPI または DFHRL であるメッセージを CICS ログに発行します。通常、それらのメッセージには、問題解決のために実行できる追加のアクションを示すユーザー応答が含まれています。

- BUNDLE リソースが使用不可の状態でインストールされ、そのリソースを使用可能にできない。ほとんどの場合、この問題は、動的に作成されるリソースのいずれかが、UNUSABLE 状態でインストールされたことが原因で発生します。
  1. IBM CICS Explorer を使用し、CICS がバンドルに作成したリソースを表示します。
  2. UNUSABLE 状態のリソースを特定し、問題を修正します。ログに追加の DFHPI メッセージまたは DFHRL メッセージがあれば、この状態でリソースがインストールされた理由が分かる場合があります。
  3. 問題を修正したら、BUNDLE リソースを破棄し、再インストールします。
- BUNDLE リソースが使用不可の状態でインストールされ、エラー・メッセージ DFHPI2005 を受け取る。 このメッセージを受け取った場合、複合アプリケーションでバインディングの互換性の問題が発生しています。
  1. Rational Developer for System z でコンポジットを編集し、コンポジット・サービスまたは参照のバインディングを変更します。サービスのバインディングが、どちらも CICS バインディングまたは Web サービス・バインディングのどちらかであることを確認してください。
  2. バンドルを CICS に再デプロイします。
  3. 使用不可の BUNDLE リソースを破棄し、再インストールします。
- アプリケーションがサービスを呼び出せなくなり、CICS がエラー・メッセージ DFHPI を発行する。 コンポジット・サービスの URI が正しいことを確認してください。

第 30 章 ユーザー・リソース・タイプをバンドルで作成する

バンドルでサポートされているリソース・タイプは拡張が可能なため、必要に応じて独自のリソース・タイプを提供できます。新しいリソース・タイプを CICS でサポートするには、バンドルが CICS にデプロイされる際に、そのリソース・タイプの作成とライフ・サイクルを処理するプログラムを作成する必要があります。

始める前に

CICS にバンドルをデプロイする前に、リソース・タイプを定義するバンドル・マニフェストがバンドル・マニフェスト・スキーマに準拠していることを確認します。

このタスクについて

CICS はチャネル・ベースの 2 つのインターフェースを提供します。最初のインターフェースは、プログラムをユーザー・リソースのマネージャーとして登録します。2 つ目のインターフェースは、CICS ドメイン間のコールバックに似ています。CICS はコールバック・インターフェースを使用し、 BUNDLE リソースのインストール時に作成する必要のあるリソースの詳細を登録済みプログラムに渡します。

1. コールバック・プログラムをリソース・タイプのマネージャーとして CICS 領域に登録するプログラムを作成します。 CICS は、領域が始動してバンドルのリソース・タイプをリカバリーするときに、このプログラムを使用します。
2. リソース・タイプの管理を行うコールバック・プログラムを作成します。 CICS は、リソース・タイプを収容するバンドルがデプロイされるときに、コールバック・インターフェースを使用してこのプログラムを呼び出します。
3. 2 つのプログラムの PROGRAM リソース定義を作成およびインストールします。 登録インターフェースを使用するプログラムは、PLTPI 処理中に使用可能である必要があります。

次のタスク

リソース・タイプを収容するバンドルが CICS に正常にデプロイできるかどうか、有効 BUNDLE リソースが CICS の再始動時にリカバリーできるかどうかをテストします。

バンドル登録インターフェース

アプリケーション・バンドルにデプロイするリソース・タイプの所有者としてプログラムを登録するには、バンドル登録インターフェースを使用します。このプログラムは、CICS 領域にバンドル実装の一部としてインストールされている特定タイプのすべてのリソースのライフ・サイクルを管理します。

バンドル とは、アプリケーションの実装単位のことです。バンドルは、BUNDLE リソースを使用して定義され、CICS 領域にインストールされます。バンドルには、アプリケーションが要求するすべてのリソースを Universal Resource Identifier (URI)

© Copyright IBM Corp. 1989, 2009

として記述するマニフェストが含まれます。これらのリソースは、標準的な CICS RDO リソース、zFS に存在する成果物、またはユーザー・リソースである場合があります。

バンドルでユーザー・リソースを定義するには、バンドルが CICS 領域にデプロイされるときにコールバック・プログラムを作成し、ユーザー・リソースを管理します。CICS は、ユーザー・リソース・タイプの所有者としてコールバック・プログラムを登録するための登録インターフェースを提供します。登録インターフェースはチャネル・ベースで、登録プログラムによってデータが取り込まれるコンテナーを備えています。

コールバック・プログラムで処理することが必要なリソース・タイプがバンドルに含まれている場合、CICS はチャネル・ベースのコールバック・インターフェースを使用してプログラムを呼び出し、リソースを作成します。

バンドルに関連付けられたリソースはすべて一括して管理されます。例えば、BUNDLE リソースを無効にすると、関連付けられたリソースもすべて無効になります。コールバック・プログラムでは、ユーザー・リソースに対して適切な場合、このライフ・サイクル・サポートを選択できます。

登録プログラムの書き込み

CICS は、バンドルでのデプロイ時に、コールバック・プログラムを登録してリソース・タイプを処理できるように、インターフェースを提供します。

このタスクについて

登録プログラムは、サポートされているプログラミング言語のいずれかで書き込むことができます。使用する必要のあるコンテナーは DFHRLUC*x* コピーブックに記述されています。*x* はプログラミング言語を表します。これらのコピーブックは CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

1. プログラム DFHRLVC にリンクし、チャネル DFHRLVC-V1 および以下の必須コンテナーを渡します。
   - DFHRL-ACTION (コールバック・プログラムの登録または登録取り消しを行うアクションが含まれる)
   - DFHRL-PROGRAM (コールバック・プログラムの名前が含まれる)
   - DFHRL-TYPE (コールバック・プログラムによって管理されるリソース・タイプの Universal Resource Name (URN) が含まれる)
2. CICS は、コールバック・プログラムの登録または登録取り消しを実行できない場合、DFHRL-ERROR コンテナーを追加します。 エラーが発生したら、プログラムはこれらの応答に対応し、適切な処理を実行する必要があります。
3. バンドル内のリソース・タイプが適切にリカバリーされるようにするには、登録プログラムを Post-initialization Program List Table (PLTPI) に追加します。 ウォーム・リスタートまたは緊急リスタートの場合、CICS は登録プログラムによって提供される情報を使用し、インストール済みの BUNDLE リソースに定義されたリソース・タイプの再作成を要求します。

タスクの結果

リソース・タイプが含まれる BUNDLE リソースをインストールする場合、CICS は登録プログラムによって指定されたコールバック・プログラムを使用し、リソース・タイプを処理します。リソース・タイプが CICS に登録されていない場合、またはコールバック・プログラムが利用できない場合、BUNDLE リソースは使用不可な状態でインストールされ、リソース・タイプ自体は使用不能な状態でインストールされます。

登録インターフェースで使用されるコンテナー

登録インターフェースは、プログラムが CICS への特定のリソース・タイプの登録に使用できる必須コンテナーおよびオプションのコンテナーを提供します。

DFHRL-ACTION コンテナー

DFHRL-ACTION は DATATYPE(CHAR) のコンテナーです。コールバック・プログラムを CICS に登録するか、または CICS から登録取り消しを行うかを指定する文字ストリングが含まれます。

このコンテナーには、以下のいずれかの値を含めることができます。

REGISTER
: コールバック・プログラムを CICS に登録し、指定のリソース・タイプを処理します。

DEREGISTER
: CICS からのコールバック・プログラムの登録取り消しを行います。このプログラムで管理されるリソース・タイプが含まれる CICS 領域にデプロイされた新規バンドルは、別のプログラムを登録しない限り、障害が起こります。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル登録インターフェースのチャネル DFHRLVC-V1 でのみ使用されます。

DFHRL-ERROR コンテナー

DFHRL-ERROR は DATATYPE(BIT) のコンテナーです。CICS がコールバック・プログラムを登録できない場合に表示されるエラー・コードを含む出力コンテナーです。

このコンテナーの構造は次のとおりです。

フィールド	タイプ	説明
rl_error_major_version	bin(31)	エラー・コンテナーのメジャー・バージョン番号。この値は 1 です。

フィールド	タイプ	説明
rl_error_minor_version	bin(31)	エラー・コンテナーのマイナー・バージョン番号。この値は 0 です。
rl_error_code	bin(8)	以下のエラー・コードが示される可能性があります。 **1** コールバック・プログラムは既に登録されています。 **2** コールバック・プログラムは既に登録が取り消されています。 **3** 関数が無効です。 **4** チャネルにコンテナーがありません。 **5** コンテナー・タイプが無効です。コンテナーには、DATATYPE(BIT) ではなく DATATYPE(CHAR) が含まれているか、またはその逆です。 **6** コンテナーに無効な内容が含まれています。 **7** チャネル名が無効です。 **8** CICS にチャネルが渡されていません。 **9** 内部エラーがあります。
rl_error_container	char(16)	コンテナーの名前が誤りです。このフィールドには、エラー・コードが 4、5、または 6 の場合に値が入力されます。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル登録インターフェースのチャネル DFHRLVC-V1 でのみ使用されます。

DFHRL-PROGRAM コンテナー

DFHRL-PROGRAM は DATATYPE(CHAR) のコンテナーです。リソース・タイプの管理とライフ・サイクルを処理するプログラムの名前が含まれます。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル登録インターフェースのチャネル DFHRLVC-V1 でのみ使用されます。

DFHRL-TYPE コンテナー

DFHRL-TYPE は DATATYPE(CHAR) のコンテナーです。コールバック・プログラムによるバンドルでの管理が可能なリソース・タイプを記述する URI が含まれます。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル・コールバック・インターフェースのチャネル DFHRLVI-V1 またはバンドル登録インターフェースのチャネル DFHRLVC-V1 でのみ使用されます。

コールバック・プログラムの書き込み

コールバック・プログラムは、デプロイされたバンドルに含まれるユーザー・リソースのライフ・サイクルを作成および管理します。デプロイされたバンドルは、BUNDLE リソースを使用して CICS 領域に定義されます。

このタスクについて

BUNDLE リソースが定義、有効化、無効化、または廃棄されると、CICS はチャネル・ベースのインターフェースを使用してコールバック・プログラムにリンクし、チャネル DFHRLVI-V1 上のコンテナーを渡します。コールバック・プログラムは、CICS に制御を返す前に、このチャネルのコンテナーを使用してユーザー・リソースを作成および管理する必要があります。

1. CICS はご使用のコールバック・プログラムにリンクします。CICS は、チャネルに以下のコンテナーを受け渡し、ユーザー・リソースを作成します。
   - DFHRL-CONTROL (リソースのトークンと、ユーザー・リソースの作成または管理を求める要求が含まれる)
   - DFHRL-DATA (リソースに関するバンドルからのメタデータが含まれる)
   - DFHRL-NAME (リソース名が含まれる)
   - DFHRL-ROOT (ファイル・システム内でバンドルがデプロイされる場所が含まれる)
   - DFHRL-TYPE (リソース・タイプの URI が含まれる)

   ユーザー・リソースの照会、ユーザー・リソースの状況の変更、またはユーザー・リソースの破棄を行う場合、CICS は DFHRL-CONTROL コンテナーだけを

そのチャネル上に配置します。 コンテナーの定数は DFHRLUC*x* コピーブックに記述されています。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

2. ユーザー・リソースを作成する、使用可能にする、使用不可にする、または廃棄する場合は、プログラムでこれらのコンテナーを使用する必要があります。 DFHRL-CONTROL コンテナーには、システム・プログラマーによって実行されている操作および BUNDLE リソースの状態を示す 2 つのフィールドがあります。

rl_function フィールド	**rl_state フィールド**	**プログラムの動作**
バンドルの作成	使用可能	プログラムは、チャネルのコンテナーに含まれる情報を使用して、ユーザー・リソースを作成して使用可能にする必要があります。
	使用不可	プログラムは、チャネルのコンテナーに含まれる情報を使用して、ユーザー・リソースを使用不可状態で作成する必要があります。
バンドルの設定	使用可能	プログラムは、rl_client_token 値を使用するか、または CICS によって DFHRL-CONTROL コンテナーでも指定されているバンドルおよびリソース・トークンを使用して、ユーザー・リソースを使用可能にする必要があります。
	使用不可	プログラムは、rl_client_token 値を使用するか、または CICS によって DFHRL-CONTROL コンテナーでも指定されているバンドルおよびリソース・トークンを使用して、ユーザー・リソースを使用不可にする必要があります。
バンドルの問い合わせ	使用可能	プログラムは、DFHRL-CONTROL コンテナーのユーザー・リソースの状態を返す必要があります。
	使用不可	
バンドルの廃棄	廃棄	プログラムは、ユーザー・インターフェースを廃棄する必要があります。

3. プログラムは処理の完了後、DFHRL-CONTROL コンテナーの rl_state フィールドを更新し、ユーザー・リソースの作成、有効化、無効化、または廃棄のいずれかの処理が成功したかどうかを示す必要があります。

- プログラムがユーザー・リソースを正常に作成できた場合、rl_state フィールドを使用可能の状態で更新し、rl_client_token フィールドに固有値を入力します。このトークンは、プログラムに対するその後のすべての呼び出しで、リソースを管理するために CICS によって使用されます。
- プログラムがユーザー・リソースを正常に使用不可に設定できた場合、rl_state フィールドを使用不可の状態で更新します。
- プログラムがユーザー・リソースを正常に廃棄できた場合、rl_state フィールドを廃棄された状態で更新します。
- プログラムがユーザー・リソースの状態を返している場合、状況に応じて rl_state フィールドを使用可能または使用不可の状態で更新します。
- プログラムがユーザー・リソースの作成、使用不可への設定、または廃棄に失敗した場合、rl_state フィールドを失敗した状態で更新します。

タスクの結果

CICS は更新された DFHRL-CONTROL コンテナーを受信すると、その情報を使用して BUNDLE リソースの状態を更新します。

コールバック・インターフェースで使用されるコンテナー

コールバック・インターフェースは、必須コンテナーやオプションのコンテナーを登録済みプログラムへの入力データとして提供し、特定のタイプのリソースを作成または管理します。

CICS は、バンドルのデプロイ時にコンテナーを初期化し、DFHRLVI-V1 というチャネルを使用して登録済みプログラムにリンクします。

DFHRL-CONTROL コンテナー

DFHRL-CONTROL は DATATYPE(CHAR) のコンテナーです。バンドルのトークン、リソースの作成、有効化、無効化、または廃棄を求める要求が含まれます。

これらのトークンは、以下のとおりです。

フィールド	タイプ	説明
rl_bundle_token	char(8)	バンドルを表す 8 バイト・トークン
rl_resource_token	char(8)	リソースを表す 8 バイト・トークン
rl_client_token	char(8)	コールバック・プログラムによって保留されるリソースのビューを表す 8 バイト・トークン

フィールド	タイプ	説明
rl_state	bin(8)	BUNDLE リソースの状態。このフィールドには、以下のいずれかの値を含めることができます。 **0** 使用可能化 **1** 使用可能 **2** 使用不可に設定 **3** 使用不可 **4** 廃棄 **5** 失敗
rl_function	bin(8)	システム・プログラマーが実行するアクションです。このフィールドには、以下のいずれかの値を含めることができます。 **0** バンドルの作成 **1** バンドルの設定 **2** バンドルの問い合わせ **3** バンドルの廃棄

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル・コールバック・インターフェースのチャネル DFHRLVI-V1 でのみ使用されます。

DFHRL-DATA コンテナー

DFHRL-DATA は、ユーザー・リソース用のメタデータを収容する DATATYPE(BIT) のコンテナーです。メタデータは、バンドル内のマニフェストが提供する情報から取得されます。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル・コールバック・インターフェースのチャネル DFHRLVI-V1 でのみ使用されます。

DFHRL-NAME コンテナー

DFHRL-NAME は DATATYPE(CHAR) のコンテナーです。デプロイされたバンドルでのリソース名を表す文字ストリングが含まれます。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル・コールバック・インターフェースのチャネル DFHRLVI-V1 でのみ使用されます。

DFHRL-ROOT コンテナー

DFHRL-ROOT は DATATYPE(CHAR) のコンテナーです。バンドル・ディレクトリーのファイル・システムの完全修飾パスが含まれます。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル・コールバック・インターフェースのチャネル DFHRLVI-V1 でのみ使用されます。

DFHRL-TYPE コンテナー

DFHRL-TYPE は DATATYPE(CHAR) のコンテナーです。コールバック・プログラムによるバンドルでの管理が可能なリソース・タイプを記述する URI が含まれます。

このコンテナーは、 DFHRLUC*x* コピーブックで定義されます。*x* はプログラミング言語を表します。これらのコピーブックは、 CICSTS41.CICS ライブラリーの SDFHSAMP メンバーです。

このコンテナーは、バンドル・コールバック・インターフェースのチャネル DFHRLVI-V1 またはバンドル登録インターフェースのチャネル DFHRLVC-V1 でのみ使用されます。

バンドル・マニフェスト・スキーマ

マニフェストを作成してバンドルでリソースを記述する場合、ファイルは XML スキーマに準拠している必要があります。

マニフェストは、BUNDLE リソースを正常にインストールするために CICS に必要なすべてのリソースとメタデータを記述します。マニフェストは、デプロイ時にバンドルの META-INF ディレクトリーに存在している必要があります。マニフェストが無効な XML であるか、またはスキーマに準拠していない場合、CICS では BUNDLE リソースをインストールできません。マニフェスト・スキーマは bundle.xsd と呼ばれ、z/OS UNIX の /usr/lpp/cicsts/cicsts41/schemas/bundle/ ディレクトリーに格納されています。

マニフェスト・スキーマの注釈付きバージョンは以下のとおりです。

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.ibm.com/xmlns/prod/cics/bundle"
            xmlns="http://www.ibm.com/xmlns/prod/cics/bundle" 1
            elementFormDefault="qualified"
            attributeFormDefault="unqualified">
```

```
    <xsd:element name="manifest"> 2
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="meta_directives" minOccurs="0" maxOccurs="1"> 3
                   <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="timestamp" maxOccurs="1" > 4
                                <xsd:simpleType>
                                    <xsd:restriction base="xsd:dateTime"/>
                                </xsd:simpleType>
                            </xsd:element>
                            <xsd:any minOccurs="0" maxOccurs="unbounded"> 5
                            </xsd:any>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="import" minOccurs="0" maxOccurs="unbounded"> 6
                    <xsd:complexType>
                        <xsd:attribute name="name"     type="xsd:string" use="required" />
                        <xsd:attribute name="type"     type="xsd:string" use="required" />
                        <xsd:attribute name="optional" type="xsd:boolean" use="optional" />
                        <xsd:attribute name="warn"     type="xsd:boolean" use="optional" />
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="export" minOccurs="0" maxOccurs="unbounded"> 7
                    <xsd:complexType>
                        <xsd:attribute name="name" type="xsd:string" use="required" />
                        <xsd:attribute name="type" type="xsd:string" use="required" />
                    </xsd:complexType>
                </xsd:element>
                <xsd:element name="define" minOccurs="0" maxOccurs="unbounded"> 8
                    <xsd:complexType>
                        <xsd:attribute name="name"     type="xsd:string" use="required" />
                        <xsd:attribute name="type"     type="xsd:string" use="required" />
                        <xsd:attribute name="path"     type="xsd:string" use="required" />
                        <xsd:attribute name="optional" type="xsd:boolean" use="optional" />
                        <xsd:attribute name="warn"     type="xsd:boolean" use="optional" />
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="bundleVersion" use="required"> 9
                <xsd:simpleType>
                    <xsd:restriction base="xsd:int">
                        <xsd:enumeration value="1" />
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="bundleRelease" use="optional"> 10
                <xsd:simpleType>
                    <xsd:restriction base="xsd:int"/>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="build" use="optional"> 11
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:maxLength value="20" />
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

1. CICS にデプロイされたアプリケーション・バンドルに属するものとしてマニフェストを定義する名前空間。

2. バンドル・マニフェストの定義。
3. バンドルのメタディレクティブ。
4. マニフェストが作成されたときのタイム・スタンプ。
5. このオプションの xsd:any フィールドによって、将来の拡張性が許可されます。
6. バンドルの「インポート」定義。
7. バンドルの「エクスポート」定義。
8. バンドルの「定義」の定義。
9. バンドル・マニフェストのバージョン・レベルを定義する必須属性。
10. マニフェストの作成に使用されるコードのリリースを含めることが可能なオプションの属性。
11. マニフェストの作成に使用されるコードのビルド・レベルを含めることが可能なオプションの属性。

SCA 複合スキーマ

Service Component Architecture 仕様に準拠する複合アプリケーションを、ツールを使用せずに作成する場合、CICS で使用される XML スキーマに、複合アプリケーションの定義を準拠させる必要があります。

バンドル・マニフェストは、バンドルに含まれるすべてのリソースを記述します。CICS は、SCACOMPOSITE のリソース・タイプの定義を複合アプリケーションとして処理します。このリソース・タイプを使用して、マニフェストでの複合アプリケーションを定義する必要があります。SCDL での複合定義が無効な XML である場合、またはスキーマに準拠していない場合、CICS は BUNDLE リソースをインストールできません。

複合アプリケーションを記述するスキーマ・ファイルには、sca-core.xsd、sca-binding-cics.xsd、および sca-implementation-cics.xsd の 3 つのファイルがあります。sca-core.xsd スキーマは sca-binding-cics.xsd スキーマをインポートし、sca-binding.xsd スキーマは sca-implementation-cics.xsd スキーマをインポートします。

コンポジット・スキーマは、z/OS UNIX の /usr/lpp/cicsts/cicsts41/schemas/sca/ ディレクトリーに格納されています。

第 7 部 ファイル制御

CICS ファイル制御装置を理解して操作します。

© Copyright IBM Corp. 1989, 2009

第 31 章 ファイル制御についての理解

CICS データ管理サービスは、従来は ***CICS ファイル制御*** と呼ばれていました。CICS ファイル制御によって、仮想記憶アクセス方式 (VSAM) または基本直接アクセス方式 (BDAM) のいずれかで管理されるにデータ・セットにアクセスできます。

CICS ファイル制御によって、VSAM および BDAM データ・セット内のデータの読み取り、更新、追加、およびブラウズを行い、VSAM データ・セットからデータを削除します。CICS ファイル制御を使用して、CICS 共用データ・テーブルおよびカップリング・ファシリティー・データ・テーブルにアクセスすることもできます。

CICS アプリケーション・プログラムはそのデータの読み取りおよび書き込みを個別レコードの形式で行います。各読み取り要求または書き込み要求は CICS コマンドによって行います。

レコードにアクセスするには、アプリケーション・プログラムがレコードとそれを保持するデータ・セットの両方を識別する必要があります。また、アプリケーション・プログラムは、レコードの読み取り先またはレコードの書き込み元となるストレージ域も指定する必要があります。

VSAM データ・セット: KSDS、ESDS、RRDS

CICS は、以下のタイプのデータ・セットへのアクセスをサポートしています。

- キー順データ・セット (KSDS)
- 入力順データ・セット (ESDS)
- 相対レコード・データ・セット (RRDS) (固定および可変レコード長の両方)

VSAM データ・セットは、直接アクセス記憶装置 (DASD) 補助記憶装置に保持されます。VSAM では、そのデータ・セットが制御域 (CA) に分割され、それがさらに制御インターバル (CI) に分割されています。制御インターバルは仮想記憶域と補助記憶装置の間のデータ伝送の単位です。一般に、それぞれの制御インターバルは固定サイズで、何件かのレコードが入ります。KSDS または ESDS では、複数の制御インターバルにまたがるレコードを持つことができます。これらはスパン・レコードと呼ばれます。

キー順データ・セット (KSDS)

キー順データ・セットでは、各レコードがキーによって識別されます (各レコードの**キー**は、レコード内で事前定義された位置にあるフィールドです)。各キーは、そのデータ・セット内で固有にする必要があります。

データ・セットに最初にデータをロードするとき、あるいは新規レコードを追加するときに、そのレコードの論理順序がキー・フィールドの照合シーケンスによって決まります。また、これにより、データ・セットをブラウズする場合のレコードの取り出し順序が決まります。

© Copyright IBM Corp. 1989, 2009

KSDS 内のレコードの物理的位置を見つけるために、VSAM は**索引**を作成し保守します。これは、各レコードのキーをデータ・セット内のレコードの相対位置に関連付けます。レコードの追加または削除を行う場合は、それに応じてこの索引は更新されます。

DFSMS/MVS™ 1.4 およびそれ以降のリリースでは、データ・セットがストレージ・クラスの中で、拡張形式および拡張アドレッシング機能として定義されている場合には、データ・セットのサイズは 4GB 以上が可能です。CICS は、RL および非 RLS モードで、これらの拡張属性で定義された KSDS データ・セットをサポートします。

入力順データ・セット (ESDS)

入力順データ・セットでは、各レコードがその**相対バイト・アドレス** (RBA) によって識別されます。

レコードは、最初にデータ・セットにロードされた順序で ESDS 内に保持されます。ESDS に追加される新規レコードの位置は、常に、データ・セット内の最終レコードの後になります。レコードを削除したり、レコードの長さを変更することはできません。レコードを ESDS に保管した後は、その RBA は一定のままです。ブラウズ時には、レコードはデータ・セットに追加された順序で取り出されます。

標準 RBA は、符号なしの 32 ビット数値です。32 ビットの RBA を使用するということは、標準 ESDS に 4 ギガバイト (GB) を超えるデータを収容できないことを意味します。ただし、64 ビットの**拡張相対バイト・アドレス** (**XRBA**) をサポートする別の種類の ESDS があり、これは 4GB の制限を受けません。このタイプの ESDS は、「拡張フォーマット、拡張アドレス方式 ESDS データ・セット」と呼ばれます。これを略して、「拡張アドレス方式 ESDS」、または「**拡張 ESDS**」と呼びます。CICS TS for z/OS バージョン 3.2 以降では、CICS は、64 ビット XRBA および拡張 ESDS データ・セットをサポートしています。

相対レコード・データ・セット (RRDS)

相対レコード・データ・セット では、相対レコード番号 (RRN) によってレコードが識別されます。データ・セットの 1 番目のレコードは RRN 1、2 番目のレコードは RRN 2 で以下同様です。

RRDS のレコードは、固定長または可変長レコードです。VSAM によるデータの処理方法は、そのデータ・セットが固定 RRDS であるか可変 RRDS であるかによって決まります。固定 RRDS には VSAM に事前定義された固定長スロットがあります。レコードはここに保管されます。固定 RRDS のレコード長は、常にスロットのサイズに等しくなります。VSAM は、スロット・サイズに RRN (ファイル制御要求に応じて与えます) を乗じて、データ・セットの頭からのバイト・オフセットを計算することにより、固定 RRDS のレコードの位置決めを行います。

一方、可変 RRDS は、そのデータ・セットの最大長までの長さレコードを受け入れることができます。可変 RRDS では、VSAM は索引を使ってレコードを見つけます。

概して、パフォーマンスの点では固定 RRDS のほうが上です。機能面では可変 RRDS のほうが優れています。

DFSMS/MVS 1.5 およびそれ以降のリリースでは、データ・セットがストレージ・クラスの中で、拡張形式および拡張アドレッシング機能として定義されている場合は、4 GB より大きいサイズのデータ・セットを使用できます。4 バイトの RRN フィールドで指定できる RRN を使用して、4 GB の境界を超えて存在するレコードにアクセスする場合、CICS は拡張 RRDS または VRRDS データ・セットへのアクセスをサポートします。

空のデータ・セット

空のデータ・セットとは、まだレコードがまったく書き込まれていないデータ・セットです。VSAM では、非 RLS アクセス・モードでオープンされる空のデータ・セットに対して、いくつか制約事項が適用されます。ただし、CICS ではこれらの制約事項をユーザーからは見えないようにしているため、アクセス・モードの種類にかかわらず、データが収容されたデータ・セットの場合と同じ方法で空のデータ・セットを使用できます。

VSAM 代替索引

レコードの同一セットを異なった方法でアクセスしたくなることがあります。例えば、人事のデータ・セットにレコードがあり、キーとして従業員番号が設定されているとします。Smith という名前の人が何人いたとしても、それぞれの人が固有の従業員番号をもっています。これを 1 次キーとします。

そのデータ・セットから電話帳を作成する場合には、従業員番号順ではなく名前順のリストが必要になることもあります。データ・セット内のレコードは、上記の 1 次キーではなく 2 次キー (代替) を使用して識別できます。それで、1 次キーを従業員番号とし、従業員氏名を**代替キー**とします。KSDS における 代替キーは 1 次キーと同様であり、レコード内の固定長および固定位置のフィールドです。代替キーは 1 次キーすなわち基本キーとは違い、基本ファイルに対していくつでも持つことができます。代替キーは固有である必要はありません。

人事データ・セットの例を続けると、従業員の部門コードを追加の代替キーとして定義することもできます。

VSAM では、KSDS および ESDS (RRDS または 拡張 ESDS は除く) の各データ・セットで代替キーを使用できます。データ・セットの作成時に、レコード内の各代替キーごとに 2 次索引すなわち**代替索引**が 1 つ作成され、1 次キーすなわち基本キーと関連付けられます。代替キーを使用してレコードにアクセスするためには、もう一つの VSAM オブジェクト、**代替索引パス**を定義しなければなりません。これにより、このパスは、代替キーを使用してレコードにアクセスする場合に使用する KSDS と同じように振る舞います。

パスによってレコードを更新する場合には、対応する代替索引が変更内容を反映するように更新されます。ただし、基本の方法により、または異なるパスによってレコードを直接更新する場合、代替索引が更新されるのは、それが基本データ・セットの**更新セット**に属するように VSAM に (作成時に) 定義されている場合のみです。ほとんどのアプリケーションで、代替索引を更新セットに入れたくなる可能性があります。

CICS アプリケーション・プログラムは、アクセスするファイルがパスであるか基本データ・セットであるかは問題にしません。実行中の CICS システムで、単一の基

本データ・セットへのアクセスは、基本の方法により行うことも、また、各アクセス経路がファイル制御テーブル (FCT) で定義されている場合には、それに定義されているいずれかのパスによって行うこともできます。

CICS アプリケーション・プログラムは、パスではなく代替索引として直接定義されているファイルにアクセスすることもできます。これにより、ファイル・データではなく、索引データがアプリケーション・プログラムに返されることになります。レコード・レベル共用 (RLS) モードでオープンされるファイルの場合、この操作はサポートされていません。

RLS モードでのファイルのアクセス

レコード・レベル共用 (RLS) は、DFSMS バージョン 1 リリース 3 以上で提供された VSAM 機能です。この機能により、多くの CICS 領域で実行中の多くのアプリケーション間で完全な更新能力を備えた VSAM データを共用することができます。

RLS では、VSAM データ・セットを共有する CICS 領域は、MVS 並列シスプレックス内の 1 つ以上の MVS イメージに常駐可能です。CICS 領域とバッチ・ジョブの間でデータ・セットが共有されている場合にも、RLS は有利です。

RLS モードでファイルをオープンする場合、制御インターバル (CI) レベルではなく、レコード・レベルでロックが起こり、デッドロックのリスクが減ります。

CICS は、以下のタイプの VSAM データ・セットへのレコード・レベル共用 (RLS) アクセスをサポートします。

- キー順データ・セット (KSDS)。KSDS を使用している場合には、相対バイト・アドレス (RBA) を使用して、ファイルにアクセスできないことに注意してください。
- 入力順データ・セット (ESDS)。RLS アクセス・モードを入力順データ・セット (ESDS) で使用することはできますが、レコードの追加時にデータ・セットのパフォーマンスおよび可用性を低下させる可能性があるため、お勧めできないことに注意してください。(「*CICS パフォーマンス・ガイド*」のVSAM レコード・レベル共用のパフォーマンス様相を参照してください。)
- 相対レコード・データ・セット (RRDS) (固定長レコードと可変長レコードの両方)。

注: ファイルに対して RLS モードを指定する SET FILE EMPTY コマンドを出すと、その要求は受け入れられますが、ファイルが RLS モードでオープンしているときは、常に無視されます。ファイルをクローズして非 RLS モードに切り替えると、データ・セットはリセットされて空になります (ただし、IDCAMS 定義で再利用可能と定義されている場合)。

ほとんどのタイプのデータ・セットを VSAM レコード・レベル共用として扱うことができ、ほとんどの CICS アプリケーションが、このアクセス・モードを有効に活用できます。しかし、一部のアプリケーションに影響を及ぼすことのある制限がいくつかあります。以下のタイプのファイル、データ・セット、またはアクセス方法は、RLS モードではサポート**されません**。

- KSDS に対する RBA アクセス

- キー範囲データ・セット
- 一時データ・セット
- IMBED 属性を持つ VSAM クラスター
- 代替索引の直接的なオープン
- クラスターの個々のコンポーネントのオープン
- カタログまたは VVDS データ・セットに対するアクセス
- CICS 保守データ・テーブル
- Hiperbatch™ (ハイパー・パッチ)

VSAM レコードの識別

データ・セット内のレコードは、キー、*相対バイト・アドレス (RBA)* または*拡張相対バイト・アドレス (XRBA)*、または *相対レコード番号 (RRN)* により識別することができます。

CICS ファイル制御コマンドの RIDFLD (レコード識別フィールド) オプションは、アクセス方式およびアクセス対象ファイルのタイプに対応したレコード ID が収容されたフィールドを識別します。レコードに対して実行可能なほとんどの操作 (読み取り、追加、削除、またはブラウズの開始) においては、RIDFLD オプションを指定してレコードを識別します (更新するために最初に読み取る場合を除く)。しかし、ENDBR、REWRITE、および UNLOCK コマンドに対し RIDFLD はありません。

- 『キー』
- 546 ページの『相対レコード番号 (RRN)、相対バイト・アドレス (RBA) または拡張相対バイト・アドレス (XRBA)』

キー

一般に、キーを使用する場合には、完全キーまたは総称 (部分) キーのいずれか一方を指定することができます。この規則の例外は、KSDS にレコードを書き込む場合、あるいは代替索引パスによってレコードを書き込む場合です。いずれの場合でも、コマンドの RIDFLD オプションに完全キーを指定し***なければなりません***。

総称キーを使用する場合には、KEYLENGTH オプションにその長さを指定し、コマンドに GENERIC オプションを指定しなければなりません。総称キーのキー長をキー全体の長さと等しくすることはできません。これは完全キーより短く定義しなければなりません。

また、一定のコマンドで、完全キーおよび総称キーの両方に GTEQ オプションを指定することもできます。そうすると、一致するキーが見つからない場合には、コマンドは 1 つ上のキーを持つレコードに位置決め、あるいは適用します。データ・セットを代替索引パスによってアクセスする場合には、一致するレコードが見つからないときに識別されるレコードは 1 つ上の代替キーを持つレコードです。

総称キーを使用する場合でも、完全キーの長さと等しい長さのレコード識別フィールド用のストレージ域を必ず使用します。ブラウズ操作時に、レコードの検索後、CICS は検索されたレコードの実際の ID をレコード識別域にコピーします。CICS は、コマンドに総称キーを指定した場合でも、完全キーをアプリケーションに返し

ます。例えば、各 READNEXT コマンドおよび READPREV コマンドでの KSDS に対する総称ブラウズは、完全キーをアプリケーションに返します。

相対レコード番号 (RRN)、相対バイト・アドレス (RBA) または拡張相対バイト・アドレス (XRBA)

データ・セットにアクセスするほとんどのコマンドに、RRN、RBA、および XRBA オプションを使用することができます。実際には、これらのオプションはレコード識別フィールド (RIDFLD) の形式を定義します。RRN、RBA、または XRBA のいずれも指定しない限り、RIDFLD オプションには、KSDS (または代替索引によって KSDS または ESDS) にアクセスするのに使用されるキーを保持する必要があります。

RRN

RRN は、レコード識別フィールドにはアクセスするレコードの相対レコード番号を入れることを指定します。データ・セット内の最初のレコードは番号 1 です。RRDS を参照し、RIDFLD オプションを指定するすべてのファイル制御コマンドには、RRN オプションも指定しなければなりません。

RBA および XRBA

RBA は、レコード識別フィールドにはアクセスするレコードの相対バイト・アドレスを入れることを指定します。相対バイト・アドレスは ESDS にアクセスするために使用され、RLS アクセス・モードではオープンされない KSDS にアクセスするために使用することもできます。ESDS 基本データ・セットを参照し、RIDFLD オプションを指定するすべてのファイル制御コマンドには RBA オプションも指定しなければなりません。

注: この方法で KSDS にアクセスする場合には、トランザクションの実行中に、同一データ・セットにレコードを追加するか、あるいはそこからレコードを削除する別のトランザクションの結果としてレコードの RBA が変更されることがあります。

RBA は、符号なしの 32 ビット数値です。32 ビットの RBA を使用するということは、標準 ESDS に 4 ギガバイト (GB) を超えるデータを収容できないことを意味します。ただし、64 ビットの**拡張相対バイト・アドレス** (**XRBA**) をサポートする別の種類の ESDS があり、これは 4GB の制限を受けません。このタイプの ESDS は、**拡張 ESDS** と呼ばれます。CICS TS for z/OS バージョン 3.2 以降では、CICS は、64 ビット XRBA および拡張 ESDS データ・セットをサポートしています。

通常、拡張 ESDS にアクセスする場合、プログラムでは 64 ビットの XRBA を提供します。32 ビットを使用する既存のプログラムを 64 ビットを使用するように変換することができます。これを行うには、関連コマンドの RBA キーワードを XRBA キーワードに置き換え、キーに使用する領域の長さを変更します。また、一部の環境では、拡張 ESDS へのアクセスを変更せずに、32 ビットの RBA を受け渡すが使用しない古いプログラムを再利用することができます。 547 ページの『ESDS の拡張アドレッシングへのアップグレード』 では、標準 ESDS を拡張アドレス方式の ESDS にアップグレードする方法と、新しいフォーマットで既存のプ

ログラムをアップグレードまたは再利用する方法を説明しています。

関連情報

『ESDS の拡張アドレッシングへのアップグレード』
拡張入力順データ・セット (ESDS) を使用するには、データ・セットをアップグレードして 32 ビットの相対バイト・アドレッシング (RBA) を使用する既存の CICS アプリケーション・プログラムを、64 ビットの拡張相対バイト・アドレッシング (XRBA) に変換する必要があります。

ESDS の拡張アドレッシングへのアップグレード

拡張入力順データ・セット (ESDS) を使用するには、データ・セットをアップグレードして 32 ビットの相対バイト・アドレッシング (RBA) を使用する既存の CICS アプリケーション・プログラムを、64 ビットの拡張相対バイト・アドレッシング (XRBA) に変換する必要があります。

標準 ESDS の拡張アドレス方式 ESDS へのアップグレード

注: 標準拡張アドレス方式を使用するように ESDS をアップグレードする前に、順方向リカバリーの使用がデータ・セットに定義されている場合は、順方向リカバリー製品を拡張アドレス方式 ESDS 用に書き込まれた新しいログ・レコードを読み取ることができる製品にアップグレードする必要があります。CICS VR を使用する場合は、CICS VSAM Recovery for z/OS V4.2 以上を使用する必要があります。

既存の標準 ESDS を拡張アドレス方式 ESDS に変換するには、データ・セットを次のように再作成する必要があります。

1. 既存データ・セットの内容を続けて使用する場合は、その内容をコピーします。これは、AMS REPRO 機能を使用して行います。
2. 既存データ・セットを削除します。
3. 新規データ・セットを作成します。新規データ・セットの AMS 定義には、前のデータ・セットの定義を基本として使用できます。必須の変更は、新規データ・セット定義の **DATACLAS** パラメーターに、拡張フォーマットと拡張アドレス方式の両方を指定する SMS データ・クラスを指名することだけです。

   SMS データ・クラスの定義方法は、「*DFSMS ストレージ管理リファレンス*」マニュアルに説明されています。
4. 必要に応じて、前にコピーしたデータ・セットの内容を復元します。

プログラムの 32 ビット RBA から 64 ビット XRBA へのアップグレード

既存プログラムを 32 ビット RBA から 64 ビット 拡張相対バイト・アドレッシング (XRBA) を使用するように変換するには、次を実行する必要があります。

1. 次のすべてのコマンドで、RBA キーワードを XRBA キーワードに置き換えます。
   - EXEC CICS READ
   - EXEC CICS READNEXT
   - EXEC CICS READPREV
   - EXEC CICS RESETBR

- EXEC CICS STARTBR
- EXEC CICS WRITE

2. キーに対して使用されているすべての 4 バイト領域を 8 バイト領域で置き換えます。「RBA」を「XRBA」に変更しても、キー領域の長さを変更しないと、次のようになります。
   a. STARTBR と READ コマンドについては、CICS は 4 バイト RBA を 8 バイトの XRBA の上部 4 バイトとして扱います。ほとんどの場合、この手順により大量の XRBA 番号が作成されます。このエラーは、プログラムが即時に「RBA にレコードがありません」という応答を受信するため、迅速に追跡することができます。
   b. WRITE コマンドは、より小さいエラーを発生させる場合があります。このコマンドは、キー領域のすぐ後に続く 4 バイトを上書きする 8 バイト XRBA を返します。

RBA に依存しないプログラムを使用した拡張 ESDS データ・セットへのアクセス

64 ビットの拡張 ESDS へのアクセスには、RBA を使用しない既存の 32 ビット RBAプログラムを再使用することができます。

例えば、最初にレコードを順次に書き込み、後で最初からそのレコードを順次にブラウズする、共通タイプのアプリケーションがあります。各 RBA は CICS とプログラムの間で受け渡されますが、プログラムはそれらの RBA を使用しません。プログラムは、次のレコードの読み取り、または書き込みをするだけです。これらのプログラムは、「RBA に依存しない」プログラムと呼ばれます。名前付き RBA でレコードを直接読み取ったり更新したりするその他のプログラムは、「RBA に依存する」プログラムと呼ばれます。

既存の 32 ビットの RBA に依存しないプログラムは、変更することなく、64 ビットの拡張 ESDS にアクセスできます。RLS モードおよび非 RLS モードの両方がサポートされています。

32 ビットの RBA に依存するプログラムは、データ・セットに収容されているデータが 4 ギガバイトより少ない場合でも、64 ビットの拡張 ESDS にはアクセスできません。

バックレベル AOR の CICS TS for z/OS バージョン 4.1 FOR への接続

このシナリオでは、旧スタイルの 32 ビット RBA プログラムが CICS TS for z/OS バージョン 4.1ファイル専有領域 (FOR) のファイルにアクセスしようとします。このアクセスは、次のいずれかのケースで機能します。

- FOR のターゲット・ファイルが従来型 ESDS から拡張アドレス方式 ESDS に変換されていない場合。
- ターゲット・ファイルは拡張アドレス方式 ESDS に変換されているが、プログラムが RBA 依存である場合。

ターゲット・ファイルが拡張アドレス方式 ESDS に変換されている場合、AOR で実行している 32 ビットの RBA 依存プログラムはそれにアクセスできません。プ

ログラムは、ILLOGIC 応答を受信します。

バックレベル FOR へのCICS TS for z/OS バージョン 4.1 AOR の接続

このシナリオでは、新スタイルの 64 ビット XRBA プログラムがバックレベルのファイル専有領域のファイルにアクセスしようとします。

ターゲット領域は 32 ビット RBA のみをサポートするため、64 ビットの XRBA を理解することはできません。プログラムは、ILLOGIC 応答を受信します。

リカバリー可能ファイルにおける VSAM レコードのロック

リカバリー可能ファイル内のレコードを変更するたびにレコード・ロックが獲得されるという点でのトランザクション・デッドロックの防止については、563 ページの『トランザクションのデッドロック』 に説明されています このようなロックは、ファイルがレコード・レベル共用 (RLS) モードでアクセスされている場合は VSAM によって、そうでない場合は CICS によって獲得されます。ロックは変更を行っているトランザクションのために、そのトランザクションが同期点要求を発行するか、あるいは終了する (この時点で同期点が自動的に実行されます) まで保留されます。VSAM リカバリー可能ファイルの処理については、以下の点に注意してください。

- VSAM レコードが変更または削除のために取得される場合は常に、CICS ファイル制御 (RLS の場合は VSAM) は 1 次レコード ID をエンキュー引数として使用し、ENQUEUE 要求によってレコードをロックします。

  パスによってレコードを変更する場合には、エンキューは基本キーまたは基本 RBA を引数として使用します。したがって、CICS は一度に 1 つのトランザクションがその要求を実行することしか許さず、他のトランザクションは最初のトランザクションが同期点に達するまで待機する必要があります。
- READ UPDATE および REWRITE 関連のコマンドの場合は、READ UPDATE コマンドが発行されるとすぐに、レコード・ロックが獲得されます。

  DELETE コマンドの前に READ UPDATE コマンドがない場合、または WRITE コマンドの場合、VSAM コマンドの実行時にレコード・ロックが獲得されます。

  (一連の WRITE コマンドから構成される) WRITE MASSINSERT コマンドの場合は、個々の WRITE コマンドが実行されるたびに、別々にレコード・ロックが獲得されます。同様に、DELETE GENERIC コマンドの場合は、削除される各レコードは、要求を発行したトランザクションのために別々のロックを獲得します。

更新ロックおよび削除ロック (非 RLS モードのみ)

既述のレコード・ロックはレコードを更新 (変更) する場合は常に獲得されるので、更新ロックと呼ばれます。リカバリー可能な KSDS、または KSDS を使用したリカバリー可能パスに対し DELETE、WRITE、または WRITE MASSINSERT コマンドを実行中の場合は、ファイル制御によってそれ以上のタイプのロック (削除ロック) が獲得されることもあります。したがって、削除操作では、削除されるレコードに関して 2 つの別々のロックが獲得されることがあります。

別々の削除ロックはファイル制御がその書き込み操作を実行する方法のために必要です。 KSDS または RRDS に対し WRITE MASSINSERT コマンドを実行する前に、ファイル制御は、1 つ以上の新規レコードを入れる空の範囲を検出してロックします。空の範囲は、データ・セット内の次の既存レコードを識別し、その削除ロックを獲得することによってロックされます。

空の範囲は、レコードをそこに同時に追加する他の要求を停止するためにロックされます。さらに、空の範囲の終わりを定義しているレコードは追加操作時に除去することはできません。別のトランザクションが要求を発行し、空の範囲にレコードを追加、またはその範囲の終わりのレコードを削除する場合には、WRITE コマンドまたは WRITE MASSINSERT コマンドが完了するまで、削除ロックによりトランザクションが待機されます。しかし、削除ロックによって保留されたレコードが別の CI に入っている場合は、書き込み操作中に別のトランザクションによって***更新される*** 可能性があります。

更新ロックと異なり、削除ロックは、削除操作または書き込み操作の期間、または UNLOCK コマンドによって終了される一連の WRITE MASSINSERT コマンドの期間にのみ保留されます。複数の空の範囲のファイルにレコードを追加する WRITE MASSINSERT コマンドは、新しい空の範囲に移動するときに、それまでの削除ロックを保留解除します。

CICS エンキュー・メカニズムは更新および削除専用で、読み取り要求が次の同期点の前に満たされるのを妨げることはしません。これらの状況下では、READ コマンドの保全性は保証されていません。

RLS レコード・レベルのロック

RLS モードでオープンされたファイルは、複数の CICS 領域から同時にアクセスすることができます。この場合、個々の CICS 領域でレコード・ロックを制御しようとするのは実際的でなく、したがって、VSAM は MVS カップリング・ファシリティーのロック補助機構を使って単一の中央ロック構造を保守します。この中央ロック構造は、レコード・レベルでシスプレックス全体のロックを提供します (制御インターバル (CI) ロックは使用しません)。これは、有効範囲が単一の CICS 領域に限定され、CI ロックか CICS ENQ のいずれかである非 RLS モードのファイルのロックとは対照的です。

RLS 内のレコード・ロックは、名前付き CICS 領域内の名前付き UOW によって所有されます。ロック所有者名は、CICS 領域の APPLID に UOW ID を付けたものです。例えば、CICS は、ロックを作成する可能性がある要求を出す場合、VSAM に UOW ID を渡します。これにより、VSAM は UOW ID、レコード・キー、および CICS 領域名を使用してロック名を作成できます。

CICS は VSAM インターフェースを使用し、UOW の完了時にすべてのロックを解除します。

複数の要求で同じリソースに対して排他ロックを要求すると、リソースが解放され、ロックを認可できるまで、VSAM は 2 番目以降の要求をキューに入れます。しかし、VSAM は、保存ロックによってロックされたリソースに対する要求はキューに入れません (552 ページの『ロックのアクティブ状態および保存状態』を参照してください)。

注: RLS アクセス・モードでオープンされたファイルに対する MASSINSERT 操作の場合、CICS は、個々の WRITE コマンドがそれぞれ発行される時点で、個別の更新ロックを獲得します。非 RLS モード操作 (549 ページの『リカバリー可能ファイルにおける VSAM レコードのロック』で説明しています) と違って、CICS は更新ロックに加え別個の削除ロックを獲得することはありません。非 RLS モードでオープンされたファイルの場合、MASSINSERT 機能のための範囲に対するロックに相当するものはありません。

排他ロックおよび共用ロック

RLS モードでアクセスされるファイルの場合、VSAM は以下の 2 タイプのロックをサポートします。2 つのタイプとは、*排他*ロックと*共用*ロックです。

排他ロックにはアクティブと保存状態があります。また、共用ロックにはアクティブしかあり得ません (552 ページの『ロックのアクティブ状態および保存状態』を参照してください)。RLS モードには削除ロックがないことに注意してください。

排他ロック

排他ロックは、リカバリー可能およびリカバリー不能の両方のファイル・リソースに対する更新を保護します。1 つのトランザクションだけが、これらを同時に所有することができます。排他ロックを要求するトランザクションは、要求するリソースに対し別のタスクが現在排他ロックまたは共用ロックを所有している場合、待機しなければなりません。

共用ロック

共用ロックは読み取りの保全性をサポートします。共用ロックはレコードが読み取り専用要求の間に更新中にならないようにします。共用ロックを使用して、レコードが読み取られた時点と次の同期点の間にレコードが更新されないようにすることもできます。

リソースに対する共用ロックを同時に複数のタスクで所有することができます。しかし、複数のタスクは共用ロックを所有できますが、タスクにロックを待たせるような状況がいくつかあります。

- 共用ロックに対する要求は、別のタスクが現在このリソースに対する排他ロックを所有している場合、待機しなければなりません。
- 排他ロックに対する要求は、他の複数のタスクが現在このリソースに対する共用ロックを所有している場合、待機しなければなりません。
- 共用ロックに対する新しい要求は、別のタスクが、既に共用ロックを得ているリソースに対し排他ロックのために待機中の場合、待機しなければなりません。

ロック期間

反復可能読み取り要求、リカバリー可能なデータ・セット、リカバリー不能なデータ・セットに対する共用ロックは、次の同期点まで保留されます。

リカバリー不能なデータ・セットのレコードに対する排他ロックは、この要求の期間のみ保留されたままです。つまり、要求の開始時に獲得され、要求の完了時に解除されます。例えば、WRITE 要求により獲得されたロックは、WRITE 要求が完了すると解除され、READ UPDATE 要求により獲得されたロックは、次の REWRITE

または DELETE 要求が完了するとすぐに解除されます。例外的に、順次要求により獲得されたロックは、即時操作の完了の後も残ることがあります。順次要求は、MASSINSERT オプションを指定し、更新要求のためにブラウズする WRITE コマンドです。MASSINSERT オプションを指定した WRITE コマンドによって獲得されたロックは、常に対応する UNLOCK コマンドの完了時に解除されますが、WRITE MASSINSERT シーケンスの最初のほうの要求によって解除されることもあります。アンロックを起こす、シーケンス内の正確な要求を予測することはできません。同様に、READNEXT UPDATE コマンドによって獲得されるロックは、後続の DELETE または REWRITE コマンドが完了してもなお存在していることがあります。このロックの解除が保証されるのは、後続の ENDBR コマンドが完了した時ですが、ブラウズ・シーケンスの中間要求によって解除されることもあります。

リカバリー可能なデータ・セットを更新する要求が出された場合、関連する排他ロックは次の同期点まで保留されたままでなければなりません。これにより、要求をコミットするかバックアウトするかが決定されるまで、リソースは確実に保護されています。CICS に障害が起きると、VSAM は CICS が再始動されるまでロックを保留します。

```
    Task 1                                      Task 2
CICS: READ(filea) UPDATE KEY(99)
VSAM: grants exclusive lock - key 99
                                  CICS: READ(filea) KEY(99)
                                        with integrity
                                  VSAM: Queues request for shared lock
CICS: REWRITE(filea) KEY(99)
VSAM: holds exclusive lock until syncpoint

CICS: task completes and takes syncpoint
VSAM: frees exclusive lock
                                  VSAM grants shared lock to task 2
```

図 116. リカバリー可能なデータ・セットに対する CICS タスク間のロック競合の例

ロックのアクティブ状態および保存状態

VSAM RLS はロックの**アクティブ**状態および**保存**状態をサポートします。この 2 タイプのロックの違いは、アクティブ・ロックを持つリソースに対して要求を出すと、そのリソースが使用可能になるまでキューで待機させられるのに対し、保存ロックを持つリソースに対して要求を出すと即時に失敗することです。

アクティブ状態は、排他ロックにも共用ロックにも適用できます。ただし、リカバリー可能リソースに対しては、排他ロックだけが状態をアクティブから保存に変更することができます。これらの状態の重要な特性は、タスクがロックを待たなければならないかどうかをこれらの状態で判別することです。

- 以下の 2 つの場合を除き、要求されたリソースに対しアクティブ・ロックが既に存在する場合には、ロックに対する要求は*待機* させられます。
  1. 現在のアクティブ・ロックが共用ロックでもあり、しかも待機している排他ロック要求がない場合には、共用ロックに対する要求は待機する必要はありません。

2. アクティブ・ロックが既に存在していれば、NOSUSPEND を指定する更新要求はロックに対して待機しません。この場合、CICS は「レコード使用中」を示す例外条件を返します。

- 要求されたリソースに対し保存ロックが存在する場合には、ロックに対する要求は LOCKED 条件で即時に拒否されます。

最初に獲得されるロックはアクティブ・ロックです。このロックはいずれにせよ解放され、そのロック期間はロックのタイプによって異なります。または、UOW が一時的に失敗し、その結果ロックが異常に長く保留されることになるイベントが発生した場合、そのロックは保存ロックに変換されます。ロックを保存ロックにすることのできるイベントのタイプは以下のとおりです。

- CICS システム、VSAM サーバーまたは MVS システム全体の障害
- バックアウトが失敗した状態になった作業単位
- リモート・システムの障害またはリモート・システムへのリンクの障害によって未確定となった分散作業単位

UOW が失敗すると、VSAM は、リカバリー可能なデータ・セットに対して失敗した UOW が所有していた排他レコード・ロックを引き続き保留します。失敗した UOW が所有するロックが原因で新規要求が待機することを防止するため、VSAM は失敗した UOW が所有するアクティブ・ロックを保持ロックに変換します。ロックを保存することにより、ロックされたレコードに対するデータ保全性は、UOW が完了するまで確実に保持されます。

CICS に障害が発生した場合も、リカバリー可能な排他ロックは保存ロックに変換されます。これにより、CICS が再始動されるか、あるいはリカバリー処理を実行するまで、データ保全性は確実に保持されます。

VSAM データ共用サーバー (SMSVSAM) が失敗した場合にも、リカバリー可能な排他ロックは保存ロックに変換されます (この変換はシスプレックスの他のサーバーか、すべてのサーバーに障害が発生した場合は最初に再始動したサーバーにより実行されます)。つまり、UOW は RLS の保存ロックを保留する目的で自ら失敗しなくてもいいわけです。

失敗した CICS 領域が所有する共用ロックはすべて廃棄されるので、アクティブ共用ロックが保存されることは絶対にありません。同様に、リカバリー不能なアクティブ排他ロックは廃棄されます。排他的で、しかもリカバリー可能なリソースに適用されるロックだけが、保存ロックになることができます。

BDAM データ・セット

CICS では、キー付き、およびキーなしの BDAM データ・セットへのアクセスをサポートしています。BDAM サポートは DASD 装置上のレコードの物理的性質を使用します。BDAM データ・セットは、次の形式の非ブロック化レコードで構成されています。

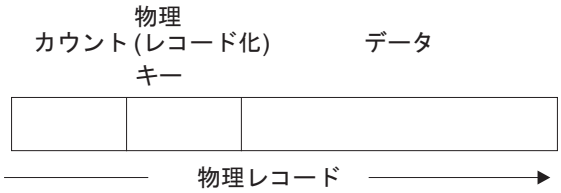


図 117. BDAM データ・セットの非ブロック化レコードの形式

キー付き BDAM ファイルは、BDAM レコードを識別する物理キーをもっています。カウント域には、物理キー長、物理データ長、およびレコードのデータ位置が入っています。

CICS では、BDAM データ・セットに加えて、ブロック化データ・セットの概念を導入した次のような構造を定義できます。

図 118. ブロック化データ・セット

物理レコードのデータ部分は論理レコードが入っているブロックのように見えます。CICS は物理レコードのデータ部分からの、論理レコードの検索をサポートします。また CICS は非ブロック化レコード (ここでは、構造は物理レコード当たり 1 つの論理レコードという BDAM 本来の概念に戻ります) をサポートします。

CICS のもとで BDAM ファイルの物理レコードからデータを検索するために、物理レコードの検索方法を指定するために、レコード識別フィールド (RIDFLD) を定義する必要があります。これは、物理キーを使用するか、相対アドレスによるか、あるいは絶対アドレスによって行うことができます。

データ・セットをブロック化されたものとして CICS に定義した場合、ブロック内の個々のレコードは、2 つのアドレッシング・モード (キーによるモードまたは相対レコードによるモード) で取得する (非ブロック化する) ことができます。

キーによる非ブロック化は、ブロックから必要なレコードを識別する論理レコードのキー (すなわち、論理レコードに入っているキー) を使用します。相対レコードによる非ブロック化は、検索するレコードのブロック内のレコード番号 (ゼロから始まる相対番号) を使用します。

CICS BDAM ファイルのアクセス時に使用する RIDFLD オプションのサブフィールドに、非ブロック化のために使用するキーまたは相対レコード番号を指定します。CICS BDAM ファイルのアドレッシング・モードは FCT で RELTYPE キーワードを使用して設定します。

レコード識別および BDAM レコード・アクセスの詳細については、 541 ページの『第 31 章 ファイル制御についての理解』を参照してください。

BDAM レコードの識別

BDAM データ・セットのレコードは、**ブロック参照**、**物理キー** (キー付きデータ・セット)、または**非ブロック化引数** (ブロック化データ・セット) によって識別します。

CICS ファイル制御コマンドの RIDFLD (レコード識別フィールド) オプションは、アクセス方式およびアクセス対象ファイルのタイプに対応したレコード ID が収容されたフィールドを識別します。レコードに対して実行可能なほとんどの操作 (読み取り、追加、削除、またはブラウズの開始) においては、RIDFLD オプションを指定してレコードを識別します (更新するために最初に読み取る場合を除く)。(しかし、ENDBR、REWRITE、および UNLOCK コマンドに対し RIDFLD はありません。)

BDAM レコードの場合、RIDFLD オプション内のレコード ID には、ブロック参照、物理キー、および非ブロック化引数のサブフィールドがあります。これらのサブフィールドを使用する場合は、上記の順序にする必要があります。

注: EDF の使用時には、上記の 3 フィールドの最初のフィールド (ブロック参照サブフィールド) しか表示されません。

ブロック参照サブフィールド

ブロック参照サブフィールドは以下のいずれかです。

- 相対ブロック・アドレス: 相対ブロック 0 で始まる 3 バイトの 2 進数 (RELTYPE=BLK)
- 相対トラックおよびレコード (16 進形式): 2 バイトの TT、1 バイトの R (RELTYPE=HEX)

  2 バイトの TT は相対トラック 0 から始まります。1 バイトの R は相対レコード 1 から始まります
- 相対トラックおよびレコード (ゾーン 10 進形式): 6 バイトの TTTTTT、2 バイトの RR (RELTYPE=DEC)
- 実 (絶対) アドレス: 8 バイトの MBBCCHHR (RELTYPE オペランドは省略します)

システム・プログラマーは、データ・セットを定義する DFHFCT TYPE=FILE システム・マクロの RELTYPE オペランドで使用するブロック参照のタイプを指定しなければなりません。

物理キー・サブフィールド

これが必要なのは、データ・セットにレコード化キーを入れるように定義した場合だけです。使用する場合には、ブロック参照のすぐ後に続けなければなりません。物理キーの長さは、データ・セットを定義する DFHFCT TYPE=FILE システム・マクロの BLKKEYL オペランドに指定されている長さと一致しなければなりません。

非ブロック化引数サブフィールド

これが必要なのは、ブロックから特定のレコードを検索したい場合だけです。使用する場合には、物理キー (存在する場合) またはブロック参照のすぐ後に続けなければなりません。非ブロック化引数を省略した場合には、ブロック全体が検索されます。

非ブロック化引数は、キーまたは相対レコード番号とすることができます。非ブロック化引数がキーである場合には、READ コマンドまたは STARTBR コマンドに DEBKEY オプションを指定し、その長さが DFHFCT TYPE=FILE システム・マクロの KEYLEN オペランドでの指定と必ず一致するようにします。非ブロック化引数が相対レコード番号である場合には、READ コマンドまたは STARTBR コマンドに DEBREC オプションを指定します。これは 1 バイトの 2 進数 (最初のレコード = ゼロ) です。

557 ページの図 119 は、以下の可能な BDAM レコード ID の形式の例を示しています。以下の例では、4 バイトの物理キーおよび 3 バイトの非ブロック化キーを想定しています。

- 相対ブロック番号とそれに続く相対ブロックによる検索のための相対レコード番号、および相対レコードによる非ブロック化
- 相対ブロック番号とそれに続く相対ブロックによる検索のためのキー、およびキーによる非ブロック化
- TTR とそれに続く相対トラックとレコードとキーによる検索のための物理キーと非ブロック化キー、およびキーによる非ブロック化
- MBBCCHHR とそれに続く実アドレスによる検索のための相対レコード番号、および相対レコードによる非ブロック化
- TTTTTRR とそれに続くゾーン 10 進相対トラックとレコードとキーによる検索のための物理キーと非ブロック化キー、およびキーによる非ブロック化
- TR とそれに続く相対トラックとレコードによる検索のための物理キー、およびキーによる非ブロック化

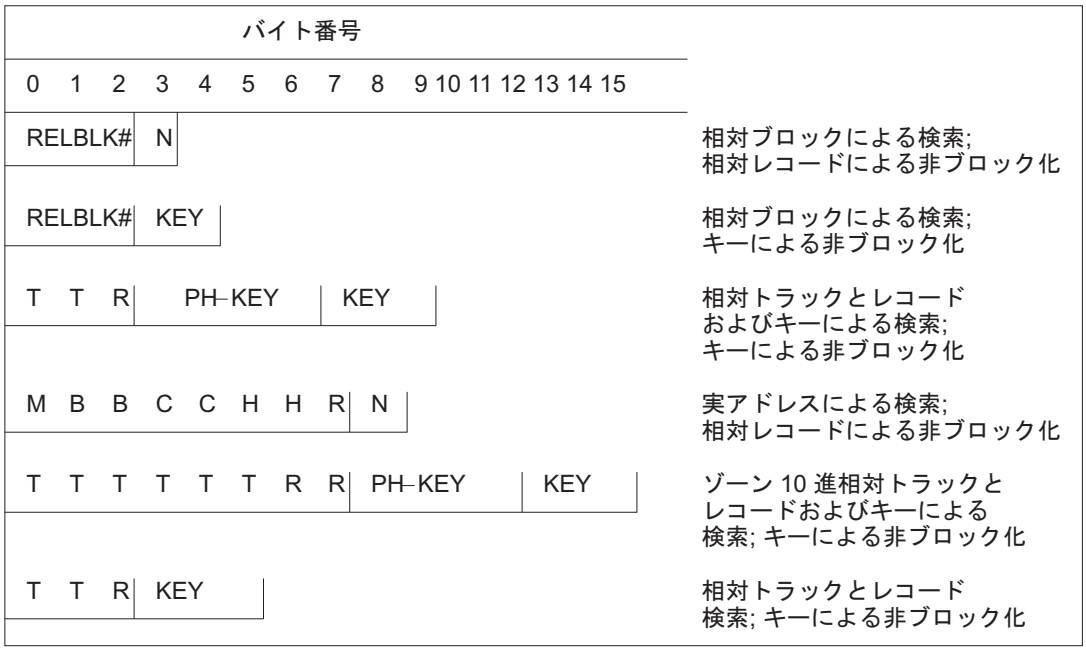


図 119. BDAM レコード識別の例

CICS 共用データ・テーブル

ファイル制御コマンドは共用データ・テーブルにアクセスすることができます。共有データ・テーブルは、16 MB 境界より上の仮想記憶域に保持されているテーブルに入っているデータ・レコードへの高速アクセスの構成、保守および取得のための方式を提供します。各共用データ・テーブルは、そのソース・データ・セットである VSAM KSDS に関連付けされます。

共用データ・テーブルの詳細については、「*CICS Shared Data Tables Guide*」を参照してください。

テーブルは、CEDA DEFINE FILE パネルを使用して定義されます。テーブルがオープンされると、CICS はそのテーブルに対応するソース・データ・セットからデータを抜き出し、それを 16 MB 境界より上の仮想ストレージにロードすることによってテーブルを構築します。

CICS は、以下の 2 タイプの共用データ・テーブルをサポートしています。

CICS 保守テーブル (CMT)
このタイプのデータ・テーブルは、CICS によって、そのソース・データ・セットとの同期が保たれます。データ・テーブルに対する変更内容は、すべてソース・データ・セットに反映されます。同様に、ソース・データ・セットに対する変更内容は、すべてデータ・テーブルに反映されます。

CICS 保守データ・テーブルのソースは、RLS アクセス・モードでオープンされたファイルではあり得ないことに注意してください。

ユーザー保守テーブル (UMT)
このタイプのデータ・テーブルは、ロードされた後、そのソース・データ・

セットから完全に切り離されます。テーブルに対する変更内容は、自動的にはソース・データ・セットに反映**されません**。

VSAM KSDS データ・セットに適した完全なファイル制御 API が、CICS 保守のデータ・テーブル用にサポートされます。データ・テーブルの参照によって満たすことができない要求があると、ソース・データ・セットにアクセスする VSAM への呼び出しが発生します。リカバリー可能として定義されているテーブルには完全な保全性がサポートされます。

ファイル制御 API のサブセットが、ユーザー保守のテーブル用にサポートされます。各コマンドに関するプログラミング情報については、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。ファイル制御コマンド名別に個々の説明がリストされ、次に UMT が続きます。例えば、レコードをユーザー保守テーブルに書き込む際の情報は WRITE(UMT) の項にあります。リカバリー可能と定義されているテーブルは動的トランザクション・バックアウトの対象となりますが、再始動または XRF 引き継ぎ時にはリカバリーされません。

カップリング・ファシリティー・データ・テーブル

CICS ファイル制御コマンドは、カップリング・ファシリティー・データ・テーブル (CFD) にアクセスすることができます。カップリング・ファシリティー・データ・テーブルは、ファイル所有領域や VSAM RLS サポートがなくても、ファイル・データ共用の方式を提供します。CICS カップリング・ファシリティー・データ・テーブル・サポートは、更新の保全性により、シスプレックス全体で作業データを高速で共用できるように設計されています。データは、カップリング・ファシリティー、すなわち多くの点で共用ユーザー保守データ・テーブルと類似するテーブル内に保持されます。カップリング・ファシリティー・データ・テーブルは、VSAM ソース・データ・セットからの初期読み込みがオプションであるという重要な点で UMT とは異なります。ユーザー・アプリケーション・プログラムからデータを直接書き込むことによって、LOAD(NO) を指定し、テーブルをロードすることができます。データを格納し検索するために使用する API は、ユーザー保守データ・テーブルに使用するファイル制御 API に基づいています。CFDT への読み取りアクセスおよび書き込みアクセスには同様な性能があり、これによって、このテーブルの形式が特に非公式な共用データに役立つ形式になります。非公式な共用データには、以下のような特徴があります。

- データは相対的に短期的性質を持つ (アプリケーションが実行中として作成されるか、あるいは初期に外部ソースからロードされる)
- 通常、データ・ボリュームはあまり多くない
- データには高速でアクセスする必要がある
- データが失われても、ユーザー・アプリケーションがそれを黙認できる
- 一般に、データの更新には保全性が必要である

典型的な使用法として、シスプレックス全体の CICS 領域間の共用スクラッチパッド・データ、または変更を永続的に保存する必要のないファイルの共用が含まれることがあります。アプリケーションが非公式共用データを使用するさまざまな方法がありますが、大部分は、カップリング・ファシリティー・データ・テーブルを使用して実装することができます。カップリング・ファシリティー・データ・テーブルは、データを異なるテーブルにグループ化する場合に特に有用です。この場合、

項目はそのキーで識別、検索することができます。例えば、カップリング・ファシリティー・データ・テーブル内のレコードを使用して、順序処理アプリケーションで、次に使用する空きオーダー番号を保守することができます。他には、以下のような例があります。

- 電話番号または盗難にあったクレジット・カードのルックアップ・テーブル
- 顧客リストからの顧客サブセットなどの、いくつかの項目から成る作業データ
- アプリケーションのユーザー固有の情報、またはアプリケーションを実行している端末に関連する情報。
- さらに処理を行うために、より大きいファイルまたはデータベースから抽出されたデータ

カップリング・ファシリティー・データ・テーブルによって、ユーザーの非公式データへのさまざまなタイプのアクセスが可能になります。これらのタイプには、読み取り専用、単一更新プログラム、複数更新プログラム、順次アクセス、ランダム・アクセス、ランダム追加および削除があります。

有効範囲がグローバルであるため、カップリング・ファシリティー・データ・テーブルには、さまざまな目的に合わせ CICS 共通作業域 (CWA) など、リソースについての大きな利点があります。

アプリケーション・プログラムに対して、CFDT は、シスプレックス全体のユーザー保守データ・テーブルに非常に類似しています。なぜなら、CFDT へは UMT と同じ API のサブセット (つまり、MASSINSERT オプションと RBA オプションを除く全ファイル制御 API) を使用してアクセスするためです。しかし、CFDT は、最大キー長が 16 バイトに制限されます。

以下のような、他のユーザー保守データ・テーブルとの比較に注意してください。

- UMT への更新のように、CFDT への更新は、基本的な VSAM データ・セットには反映されない (テーブルが最初にデータ・セットからロードされた場合)。更新されるのは CFDT だけです。
- CFDT は、最初にカップリング・ファシリティー・データ・テーブル内にテーブルが作成されたときに、一度だけロードされ、CFDT を参照する最後のファイルがクローズされても、カップリング・ファシリティー内にそのまま存在し続ける (これに対して UMT は所有領域が終了するたびに削除される)。元のソース・データ・セットから CFDT を強制的に再ロードできるのは、CFDT サーバーの DELETE TABLE コマンドを使用して、最初に CFDT プールからテーブルを削除した場合だけです。削除操作の後で最初に CFDT に対してファイルをオープンすることによって、サーバーがテーブルを再ロードします。

  注: カップリング・ファシリティー・データ・テーブル・プールは、カップリング・ファシリティー・リスト構造として定義され、複数のデータ・テーブルを保持できます (カップリング・ファシリティー・データ・テーブルのリスト構造の作成については、「*CICS System Definition Guide*」を参照)。
- ロード過程にある UMT に対するアクセス規則によって、テーブル (レコードが既にロードされている場合)、またはソース・データ・セットのいずれかからの、直接読み取り要求があれば、それが満たされます。しかし、不正確な読み取りまたはブラウズ要求があった場合は、ロード条件によって更新要求が拒否されま

す。CFDT の場合、ロード中はあらゆる要求が許可されますが、正常に実行されるのは、既にロードされたキー範囲内にあるレコードに対する要求だけです。

カップリング・ファシリティー・データ・テーブルのモデル

カップリング・ファシリティー・データ・テーブルには、以下の 2 つのモデルがあります。

競合モデル

このモデルはパフォーマンス面では最適ですが、プログラムが更新用の読み取り要求を出してからデータが変更された状態を処理するために作成されたプログラムが必要になります。新規 CHANGED 応答は、REWRITE コマンドまたは DELETE コマンドで起こります。既存の NOTFND 応答にも新規使用法があります。この応答は、プログラムが更新用の読み取り要求を出してからレコードが削除されたアプリケーション・プログラムを示すために、返されることがあります。

注: 競合モデルでは、既存のプログラムが REWRITE または DELETE で CHANGED または NOTFND 例外を受け取らないことが確実な場合、それらのプログラムを使用することもできます。この例は、アプリケーション・プログラムが、プログラムのユーザーに関係があるレコード上でのみ稼働するため、同一のレコードを更新しているユーザーが他にいない場合です。

ロック・モデル

このモデルは、ファイル制御 API の UMT サブセットに準拠する既存のプログラムとの API 互換性があります。ロック・モデルは以下のような性質があります。

リカバリー不能

リカバリー不能 CFDT に対して更新を行った場合、ロックは同期点まで継続せず (ファイル制御要求の完了時に解除される)、作業単位が失敗すると更新はバックアウトされません。

リカバリー可能

CFDT は、作業単位、CICS 領域、CFDT サーバー、および MVS で障害が発生した場合にはリカバリー可能です (障害の発生時に実行中であった作業単位ごとの更新はバックアウトされます)。

リカバリー可能なロック・モデルは、未確定障害およびバックアウト障害をサポートします。作業単位が、更新を CFDT にバックアウト中に失敗したり、同期点処理中に未確定で失敗したりすると、ロックは保存済みロックに変換され、その作業単位はわきへ押しのけられます。

CFDT は順方向にはリカバリーできません。CFDT は、それが常駐する CF 構造が失われると存在しなくなります。

ファイル・リソース定義の各テーブルに必要な更新モデルを指定します。これによって、テーブルごとに別のモデルを使用することができます。

データの共用手法

このトピックでは、表形式で、異なる状態で使用できる CICS の各種手法を比較することによって、カップリング・ファシリティー・データ・テーブルの使用を考慮すべき場合を示します。

表 24. スクラッチパッド・データの共用手法

制約および要因	単一領域	単一 MVS	シスプレックス
推奨されない手法 (制約過多)	TWA	—	—
各トランザクションの単一区域に推奨される方式	COMMAREA またはチャネル	COMMAREA またはチャネル	COMMAREA またはチャネル
一時記憶域 (TS) キューを使用する既存のアプリケーション・プログラム	ローカル TS キュー	リモート TS キュー	共用 TS キュー
既存プログラムは UMT を使用 要ランダム挿入および削除 複数タイプのデータを保管	UMT	リモート UMT	CFDT (競合モデル)

表 24 では、トランザクションのフェーズ間でスクラッチパッド・データを渡す場合に、さまざまな手法が考えられます。この場合、一度にデータにアクセスするタスクは 1 つだけですが、1 つの領域のタスクから別の領域のタスクにデータが渡されることがあります。「リモート UMT」は、共用ユーザー保守データ・テーブルを意味することに注意してください。このテーブルは、必要に応じた機能シップ (つまり、更新アクセス用)、または 非更新アクセス用の SDT メモリー間共用のいずれかによって、AOR からアクセスされます。表は、Parallel Sysplex®内では、データのランダム挿入と削除、および複数タイプのデータを格納する必要がある場合、カップリング・ファシリティー・データ・テーブルが最良の解決法であることを示しています。これらの制約がなければ、アプリケーション・プログラムが既に一時記憶域を使用している場合には、共用 TS キューを選択するほうが妥当です。

表 25. データ・キュー共用の手法

制約および要因	単一領域	単一 MVS	シスプレックス
先頭で読み取り専用、 末尾で書き込み専用 要トリガー	ローカル一時データ (TD)	リモート TD	リモート TD
項目の処理バッチ	TS キューまたは UMT	リモート TS またはリモート UMT	共用 TS CFDT
処理後の各項目の削除。要ランダム挿入および削除。	UMT	リモート UMT	CFDT

表 25 には、データ・キューを共用するための各種手法が示されています。この場合、情報は、同一シーケンスの別のアプリケーション・プログラムまたはタスクで処理されるように、特定のシーケンスに保管されます。多くの場合、CICS 一時データおよび一時記憶キュー機能の使用をお勧めします。この機能には、データ・テー

ブルによって、シーケンス化したデータをより適切に処理できる解決法を提供するインスタンスがいくつかあります。

表 26. 制御レコードの共用手法

制約および要因	単一領域	単一 MVS	シスプレックス
推奨されない手法	CWA	MVS CSA	—
単一更新領域、単一レコード	TS キューまたは UMT	リモート TS キューまたは UMT	共用 TS キューまたは CFDT (競合モデル)
複数更新領域または複数レコード	UMT	リモート UMT	CFDT

表 26 では、制御レコードのさまざまな管理手法が示されています。これは、すべてのトランザクションに対して情報を使用可能にするために、中央制御レコードが使用される場合を説明しています。例えば、この表には、プログラムがキー付きファイルまたはデータベース内に、新規レコードをより容易に作成できるように、次の未使用オーダー番号、または顧客番号が含まれています。(このタイプのアプリケーションの場合、名前付きカウンター機能を考慮することも必要です。この機能もシスプレックス全体の機能です。詳細については、717 ページの『第 53 章 概要: 名前付きカウンター・サーバー』を参照してください。)

この表は、更新をすべて単一レコードにするような単一領域がある場合、MVS イメージでは、オーバーヘッドを伝送する機能がなくても UMT を使用できることを示しています。

制御レコードを更新中の複数領域がある場合、あるいは更新すべき複数の制御レコードがある場合は、カップリング・ファシリティー・データ・テーブルが並列シスプレックス環境における唯一の解決法です。また、その環境は、単一 MVS における UMT の更新を伝送する機能より効果的である可能性もあります。

表 27. キー付きデータの共用手法

制約および要因	単一領域	単一 MVS	シスプレックス
読み取り専用またはまれに更新	UMT	UMT	複製された UMT
単一更新領域	UMT	UMT	複製された UMT または CFDT
リカバリー可能な複数の更新領域 (バックアウト専用)	UMT	リモート UMT または CFDT	CFDT

表 27 には、キー付きデータを共用するための各種手法が示されています。これは、従来のキー付きファイルと同様の構造のデータを使用するアプリケーションを対象としていますが、ファイルに情報を永続的に保管する必要がないことが条件です。関連するデータを保管するために主記憶装置またはカップリング・ファシリティー・リソースを使用する必要がありますが、パフォーマンスが十分に向上します。

この種のデータは、ファイル制御 API を使用して最も適切にアクセスされます。このことは、並列シスプレックス内では、解決法として以下を使用することを意味します。

- 複製されたユーザー保守データ・テーブル。最高のパフォーマンスが必要であり、アクセスが読み取り専用であるか、あるいはまれに更新されます。また、単一領域を構成するようにこれらの更新を配置し、他の領域内の複製された UMT を最新表示することができます。
- カップリング・ファシリティー・データ・テーブル

UMT 用のリカバリー・サポートは、障害発生後のトランザクションのバックアウトに限定されるので注意してください。カップリング・ファシリティー・データ・テーブルの場合、CICS および CFDT サーバー障害、さらに未確定の障害でもリカバリーされます。

トランザクションのデッドロック

アプリケーションは、トランザクション・デッドロックを回避するように設計してください。デッドロックが発生するのは、2 つのトランザクション (例えば、A および B) のそれぞれが、他方が既に保持しているリソース (例えば、データ・セット内の特定レコード) を排他的に使用する必要がある場合です。トランザクション A はそのリソースが利用可能になるまで待機します。しかし、トランザクション B も同様に A によって保持されているリソースに対して待機しているためにリソースを解放できる状態でない場合、両方ともデッドロック状態となります。デッドロックを解除するには、トランザクションの一方を取り消して、そのトランザクションのリソースを解放するしか方法がありません。

ファイル制御コマンドを実行中、トランザクションはリソースの排他制御を必要とします。VSAM および BDAM の両方のデータ・セットの場合、変更されるすべてのレコードは、変更の開始時 (例えば、レコードの制御を取得するために READ UPDATE コマンドが発行された時) から変更の終了時 (例えば、REWRITE コマンドがレコードの変更を終了した時) まで、CICS による排他制御下に置かれます。

RLS モードでアクセスされる VSAM ファイルの場合、この処理の間、個々のレコードのみがロックされます。非 RLS モードでアクセスされる VSAM ファイルの場合、VSAM が、レコードの制御を要求するコマンドを受け取ると、そのレコードが含まれる全制御インターバルをロックします。すると、CICS は、そのレコード上の必要なエンキューを得て、制御インターバルを解放します。この時、そのレコードのみがロック状態のままにされます。制御インターバル・ロックは、各コマンドが完了するごとに解放され、変更プロセスの開始から終了までの間、個々のレコードのみがロックされます (制御インターバル・ロックの解放方法の詳細については、「*CICS Recovery and Restart Guide*」のロックを参照してください)。

CICS Transaction Server for z/OS バージョン 2 リリース 2 より前のリリースのアクセス方式では、コマンド間 (変更の開始時から変更の終了時まで) においても全制御インターバルのロックが保持されます。今回のリリースから、これは行われません。

変更プロセスの間に CICS がレコードの排他制御を保持する他、トランザクションが**リカバリー可能**ファイル内のレコードを変更する際に追加のロック期間が存在します。この状況では、CICS (または RLS モードでアクセスされるファイルの場合は VSAM) は、変更を実行する要求が完了した後であっても、そのレコードをこのトランザクションに対しロックします。トランザクションは同一レコードのアクセスおよび変更を続行することができます。しかし、このトランザクションが終了す

るか、あるいは同期点要求を発行してロックを解放するまで、他のトランザクションは待機しなければなりません。詳しくは、 380 ページの『同期点処理』を参照してください。

デッドロックが起こるかどうかは、別の並行トランザクションでリソースを入手したり、解放したりする相対的なタイミングによって決まります。アプリケーション・プログラムは、デッドロックの原因となる状況が起きるまで、しばらくの間、続けて使用されることがあります。アプリケーション・プログラムの設計段階の初期にデッドロックの可能性を認識し、考慮しておくことが重要です。

リカバリー可能なデータ・セットで発生するさまざまなタイプのデッドロックの例を以下に示します。

- 並行して実行される 2 つのトランザクションが、以下のように、同一の FCT 項目によって、1 つのリカバリー可能ファイルのレコードを変更している場合。

```
    Transaction 1
              READ UPDATE record 1
              UNLOCK      record 1

              WRITE       record 2

Transaction 2
             DELETE      record 2

             READ UPDATE record 1
             REWRITE     record 1
```

(UNLOCK コマンドで READ UPDATE コマンドが完了していたにもかかわらず)、トランザクション 1 がレコード 1 に対しレコード・ロックを獲得しました。同様に、トランザクション 2 がレコード 2 に対するレコード・ロックを獲得しました。その後で、一方が他方によって保留される CICS ロックを必要とするために、トランザクションはデッドロックになります。CICS ロックは、同期点まで解放されません。

- 並行して実行される 2 つのトランザクションが、以下のように、2 つのリカバリー可能ファイルを変更している場合。

```
    Transaction 1                    Transaction 2
READ UPDATE  file 1, record 1      READ UPDATE file 2, record 2
REWRITE      file 1, record 1      REWRITE     file 2, record 2

READ UPDATE  file 2, record 2      READ UPDATE file 1, record 1
REWRITE      file 2, record 2      REWRITE     file 1, record 1
```

この例では、レコード・ロックが 1 つのファイルの中だけでなく、異なるファイルの間でも獲得されています。それでも、最初の例と同様のデッドロックが起こります。

RLS モードでアクセスされる VSAM ファイルの場合、CICS はデッドロックを引き起こす可能性のある状況を検出し、デッドロックに入ろうとしているトランザクションは、異常終了させられます。これが別のトランザクションによりデッドロックされた場合、異常終了コードは AFCF で、また、自らデッドロックした場合、異常終了コードは AFCG です。

VSAM が検出するデッドロック (RLS のみ)

RLS モードでアクセスされるファイルの場合、デッドロックは、おそらく異なる MVS イメージ下で実行中の 2 つの異なる CICS 領域の間で起こる可能性があります。このような場合、CICS ではデッドロックの検出および解決はできません。そのため VSAM が対応することになります。

VSAM は、RLS デッドロック条件を検出した場合、CICS にデッドロック例外条件を返し、CICS ファイル制御により、そのトランザクションを異常終了コード AFCW で異常終了させます。CICS はまた、デッドロック・チェーンのメンバーを識別するメッセージおよびトレース項目も作成します。

しかし、VSAM ではリソース間デッドロック (例えば、RLS および DB2 リソースの使用により起きたデッドロック) は検出できません。これには、別のリソース・マネージャーが関係しています。リソース間デッドロックは、タイムアウト期間が終わり、待ち要求がタイムアウトになった時点で VSAM により解決されます。この状況で、VSAM は、タイムアウトが、リソース間デッドロックにより引き起こされたのか、RLS ロックを獲得して、それを解放しない別のトランザクションにより引き起こされたのか判別できません。タイムアウトが起きた場合は、タイムアウト・トランザクションが待機しているロックの保持者を識別できるように、CICS はトレース項目およびメッセージを書き込みます。

RLS モードで発行されたすべてのファイル制御要求は、関連するタイムアウト値をもっています。このタイムアウト値は、DTIMEOUT がトランザクションに対してアクティブにあれば、DTIMEOUT により定義され、また、DTIMEOUT がアクティブでなければ、システム初期設定テーブルの FTIMEOUT により定義されるものです。

デッドロックを回避するための規則

しかし、以下の規則に従ってデッドロックを回避することができます。

- 複数のリソースを更新 (変更) するすべてのアプリケーションは、同じ順序でそれを行う必要があります。例えば、トランザクションがデータ・セット内の複数のレコードを更新する場合には、キーの昇順でこれを実行できます。複数のファイルにアクセスしているトランザクションは、常に事前定義したファイルのシーケンスでこれを行う必要があります。

  データ・セットに代替索引がある場合は、基本キーでいくつかの更新を実行するトランザクションと、代替キーでいくつかの更新を実行するトランザクションとの混合に注意してください。更新を実行するトランザクションは、常にキーの昇順にレコードにアクセスするものとします。その場合、基本キーによりすべての更新を実行するトランザクションは、基本キーによりすべての更新を実行するトランザクションとの兼ね合いでデッドロックすることはありません。同様に、代替キーですべての更新を実行するトランザクションは、代替キーですべての更新を実行する他のトランザクションとの兼ね合いでデッドロックすることはありません。しかし、基本キーですべての更新を実行するトランザクションは、代替キーですべての更新を実行するトランザクションとの兼ね合いでデッドロックすることがあります。これは、ロックされるキーが常に基本キーであるためです。したがって、代替キーで更新を実行するトランザクションは、基本キーで更新を実行するトランザクションとは異なる順序でロックを獲得します。

- READ UPDATE コマンドを発行するアプリケーション・プログラムは、その後に REWRITE、DELETE (RIDFLD を指定しない)、または UNLOCK コマンドを続けてファイルにその他の処理を実行する前に位置を解放するか、または各更新要求の両方の部分に TOKEN オプションを組み込む必要があります。
- 一連の WRITE MASSINSERT コマンドは、UNLOCK コマンドで終了して、位置を解放しなければなりません。そのファイルに対する他の操作は、UNLOCK コマンドを発行する前に実行してはいけません。
- アプリケーション・プログラムは、ENDBR コマンドを使ってファイルのブラウズをすべて終了させておかないと (したがって VSAM ロックを解放しない)、そのファイルに READ UPDATE、WRITE、または DELETE (RIDFLD を指定した) コマンドを発行することはできません。

第 32 章 ファイル制御操作

CICS ファイル制御によって、VSAM および BDAM データ・セット内のデータの読み取り、更新、追加、およびブラウズを行い、VSAM データ・セットからデータを削除します。CICS ファイル制御を使用して、CICS 共用データ・テーブルおよびカップリング・ファシリティー・データ・テーブルにアクセスすることもできます。

CICS コマンドを使用したレコードの読み取り

このタスクについて

このセクションでは、データ・セットにアクセスするために、アプリケーション・プログラムで使用可能な機能について説明します。VSAM データ・セットについて説明しますが、ほとんどの機能が BDAM にも同様に適用されます。以下のことについて説明します。

- 『直接読み取り (READ コマンドの使用)』
- 571 ページの『順次読み取り (ブラウズ)』
- 576 ページの『スキップ順次処理』

ファイルは、固定長レコードまたは可変長レコードのいずれかが入るものとしてファイル定義に定義することができます。固定長レコードを定義する必要があるのは、以下の場合だけです。

- (アクセス方式サービスを使用する) VSAM データ・セットの定義で、最大レコード・サイズに等しい平均レコード・サイズを指定している場合
- **および**、データ・セット内のすべてのレコードがその長さになっている場合。

直接の読み取りおよびブラウズの場合は、ファイルに含まれているレコードが固定長で、しかもアプリケーション・プログラムが用意した区域にレコードを読み込むのであれば、その区域は定義済みの長さでなければなりません。ファイルに可変長レコードが入っている場合は、コマンドにレコードを保持するために用意する区域の長さ (通常、ファイルの最大レコード長にする必要があります) も指定しなければなりません。

固定長レコードの場合、および (SET オプションを使用する際) CICS に用意されているストレージ内に取り出されるレコードの場合には、LENGTH 引数を指定する必要はありません。ただし、LENGTH 引数はオプションですが、INTO オプションを使用している場合はこの引数を指定するようにしてください。この指定があれば、CICS は、使用できるデータ区域に対して、読み取られたレコードが長すぎないか検査するためです。LENGTH を指定すれば、CICS は LENGTH フィールドを使って、取り出したレコードの実際の長さを返します。

直接読み取り (READ コマンドの使用)

ファイル内のレコードは、READ コマンドを使用して読み取ることができます。

© Copyright IBM Corp. 1989, 2009

このタスクについて

READ コマンドを使用する場合は、以下のようにします。

- RIDFLD (レコード識別フィールド) オプションを使用して対象のレコードを指定し、そのレコード識別フィールドの内容を記述する各オプションを追加します。レコードを識別する実際の方式は、以下のようにデータ・セットのタイプによって決まります。
  1. KSDS の場合は、全キーを指定して必要なレコードを固有に識別するか、または総称キーを指定してその要件に一致するキーを持つ最初の (最下位のキーの) レコードを取り出すことができます。
     - GENERIC オプションは、キーの一部のみの一致が必要であること指示します。GENERIC オプションを指定した場合は KEYLENGTH オプションも指定して、左から始まり、一致する必要のあるキーの桁数を示す必要があります。READ コマンドの場合、CICS は最初の KEYLENGTH オプション文字のみを使用します。
     - GTEQ (より大か等しい) オプションは、指定したキーより「大きいか等しい」キーを持つ最初のレコードを取り出すことを指示します。GTEQ は全キーまたは総称キーのいずれか一方を指定して使用することができます。
     - GTEQ オプションと反対の EQUAL オプション (デフォルト) があります。これを使用すると、指定したキーの割当て (フルまたは総称) と正確に一致するキーを持つレコードだけを必要としているということになります。

     KSDS に代替索引および代替索引パスがある (さらに FCT に該当する項目がある) 場合には、代替索引で設定した代替キーを指定することにより、ファイル内のレコードを取り出すことができます (ステップ 3 を参照)。
  2. 標準 (非拡張) ESDS の場合には、その相対バイト・アドレス (RBA) を指定することにより、レコードを識別できます。

     RBA オプションを追加し、RBA が使用中であることを CICS に通知します。ESDS 内のレコードの RBA は変更することができないので、ユーザー・アプリケーション・プログラムはアクセスの必要があるレコードと対応する RBA の値を記録します。RBA は常にレコードの先頭を指している必要があります。KSDS で使用可能な GENERIC または GTEQ の各オプションに相当するオプションはありません。
  3. 拡張 ESDS の場合には、その拡張相対バイト・アドレス (XRBA) を指定することにより、レコードを識別できます。

     XRBA オプションを追加して、XRBA を使用していることを CICS に通知します。ESDS 内のレコードの XRBA は変更できないため、ユーザー・アプリケーション・プログラムではアクセスの必要があるレコードに対応する各 XRBA の値を追跡できます。XRBA は常にレコードの先頭を指している必要があります。KSDS で使用可能な GENERIC または GTEQ の各オプションに相当するオプションはありません。
  4. 代替索引がある KSDS または 標準 ESDS (拡張 ESDS では代替索引を使用できません) の場合には、代替索引で設定した代替キーを指定することにより、ファイル内のレコードを取り出すことができます。 代替キーを使用する場合は、以下のようになります。

- GENERIC オプションおよび GTEQ オプションは、KSDS および ESDS の両方のレコードに対し、1 次キーを使用した KSDS からの読み取りと同じ方法で使用できます。
- 代替キーが固有ではない場合は、ファイルからそのキーの最初のレコードが読み取られ、DUPKEY 状態が発生します。同一の代替キーを持つ他のレコードを検索するためには、この地点でブラウズ操作を開始する必要があります。
- 一致するレコードが検出されない場合は、1 つ上の代替キーを持つレコードが識別されます。

5. RRDS の場合は、相対レコード番号 (RRN) を指定することにより、レコードを識別します。RRN オプションを追加し、RRN が使用中であることを CICS に通知します。 アプリケーション・プログラムには、取り出したいレコードの RRN 値がわかっている必要があります。KSDS で使用可能な GENERIC または GTEQ の各オプションに相当するオプションはないため、代替キーは使用できません。

- 必要に応じて KEYLENGTH オプションを指定し、RIDFLD オプションで指定したキーの長さを指定します。
  1. レコード識別フィールドに RBA または RRN を指定し、さらに RBA または RRN オプションを指定した場合、KEYLENGTH オプションは**必要ありません**。
  2. GENERIC オプションを指定した場合は、KEYLENGTH オプションが**必要です**。総称キーの長さを指定します。この長さは、VSAM 定義で指定したキーの長さより小さくする必要があります。
  3. リモート・ファイルを読み取り、SYSID オプションを指定した場合、キーの長さは以下のいずれかの方法で識別できます。
     - ファイル定義での指定。
     - KEYLENGTH オプションを使用したアプリケーション・プログラムでの指定。
     - デフォルトの 4 (キーが 4 文字より長く、キーの長さがファイル定義またはアプリケーション・プログラムで指定されていない場合)。

     これは、RBA または RRN を使用した場合を除き、リモート・ファイルでは KEYLENGTH オプションが必須である WRITE コマンドの場合と異なります。
  4. その他の場合、KEYLENGTH オプションは指定も省略も可能です。 データ・セット用に定義された長さと異なる KEYLENGTH を指定し、その操作が汎用ではない場合は、INVREQ 状態が発生します。
- INTO または SET オプションを使用して、そのレコードが、アプリケーション・プログラムが提供する主記憶の領域に読み取られるか (READ INTO)、またはファイル制御によって獲得された CICS SET ストレージの主記憶の領域に読み取られるか (READ SET) を指定します。 後者の場合には、CICS SET ストレージ内のデータのアドレスがユーザー・プログラムに返されます。 CICS SET ストレージが有効なまま保持されるのは、通常、次の同期点、タスクの終了、または同一ファイルに対する次の READ のいずれかが最初に起きた時点までです。
- 以下のように、必要に応じて LENGTH パラメーターが設定されていることを確認します。

1. SET オプションを使用した場合は、LENGTH オプションを指定する必要はありません。ただし、レコード長が VSAM での元の定義に合致するかどうか確認する場合は、指定できます。 指定されたデータ域は、レコードの取得後に、実際のレコード長に設定されます。
2. 固定長レコードの場合は、LENGTH オプションを指定する必要はありませんが、指定することをお勧めします。 このオプションを指定すると、使用可能なデータ域に対してレコードが長すぎないか CICS により検査されます。指定する長さは、そのファイルの作成時に指定されたレコード長と正確に一致させる必要があります。
3. 可変長レコードの読み取りに INTO オプションを使用した場合は、長さのパラメーターを指定する必要があります。 ただし、以下のように、コマンドで LENGTH オプションを明示的に指定する必要は必ずしもありません。
   - リモート・システム上のファイルの場合、LENGTH パラメーターはここで設定する必要はありませんが、ファイル・リソース定義で設定する必要があります。
   - アセンブラー言語または PL/I を使用している場合、LENGTH を明示的に指定する代わりに、アセンブラー言語における長さ属性参照、または PL/I における STG および CSTG を使用することにより、デフォルト設定を使用できます。C の場合は LENGTH を明示的に指定する必要があります。

   可変長ファイルの読み取り時に LENGTH オプションを明示的に指定する場合は、アプリケーション・プログラムで受け入れ可能な最長のレコードを指定し、その値がデータ・セットの作成時に定義された最大レコード・サイズの値に対応している必要があります。 レコードが指定された長さを超過した場合は、LENGERR 状態が発生し、そのレコードは指定した長さに切り捨てられます。 レコードが取得された後、(切り捨てが行われる前に) そこで指定されたデータ域が実際のレコード長に設定されます。

- ファイルが RLS モードでオープンされた場合は、UNCOMMITTED、CONSISTENT、REPEATABLE および NOSUSPEND の各オプションを使用し、読み取りの保全性を制御します。
  1. UNCOMMITTED オプションを指定した場合、読み取りの保全性はなく、共用ロックは読み取り要求のためには使用されません。 550 ページの『RLS レコード・レベルのロック』 には、共用および排他ロックについて説明されています。 これはデフォルトで、非 RLS モードでオープンされるファイルに対してファイル制御を行う方法です。
  2. CONSISTENT オプションを指定した場合、読み取り要求の対象であるレコードが別のタスクによって更新中であると、そのレコードの読み取り要求はキューに入れられます。 その読み取りが完了するには、更新が完了し、その更新の作業単位 (UOW) が排他ロックを解放したときです。 380 ページの『同期点処理』 には、UOW および同期点について説明されています。
  3. REPEATABLE オプションを指定した場合、その読み取り要求の処理は、読み取り側が同期点まで共用ロックを保持する場合を除き、整合性のある読み取り要求の場合と同じです。 これは、リカバリー可能ファイルおよびリカバリー不能ファイルの両方に適用されます。これにより、ある UOW がさらに読み取り要求を出している間は、その UOW におけるレコード読み取りは変更で

きなくなります。特に、関連のある一連の読み取り要求を発行して、最後のレコードが読み取られるまでレコードの変更が行われないようにしたい場合には便利です。

4. CONSISTENT または REPEATABLE のどちらかを指定する場合は、NOSUSPEND オプションも指定すると、レコードが VSAM のアクティブ・ロックによりロックされてもその要求が待機しないようにできます。 552 ページの『ロックのアクティブ状態および保存状態』 には、アクティブ・ロックについて説明されています。

上記のオプションをいずれも指定しない場合は、ファイル・リソース定義からの値が使用されます。

注: アプリケーションが「不整合」のデータを許容できない場合にのみ、読み取りの保全性を指定してください。これは、読み取りの保全性をサポートするために RLS がロックを使用するので、その結果アプリケーションで、読み取り整合性をサポートしていない CICS のリリースでは発生しないデッドロックが起こることがあるためです。ファイル・リソース定義で読み取りの保全性を定義する場合、このことは特に重要です。これらのファイルを参照するアプリケーション・プログラムは、読み取りの保全性をサポートしない CICS リリースに合わせて書かれていることがあり、その場合、読み取り専用ファイルにアクセスする際のデッドロック条件に対処できるように設計されていません。

順次読み取り (ブラウズ)

ファイル内のレコードをブラウズするには、
STARTBR、READNEXT、READPREV、および RESETBR の各コマンドを使用します。ブラウズ時には、前方または後方へのブラウズ、およびブラウズの位置または特性の変更が可能です。ブラウズを終了するには、ENDBR コマンドを使用します。

このタスクについて

レコードをブラウズする場合、一般的に READ コマンドを使用した直接読み取り (567 ページの『直接読み取り (READ コマンドの使用)』 を参照) と同じ方法でレコードを識別し、読み取ります。レコード識別フィールド (RIDFLD オプション) を指定し、読み取るレコードの宛先を選択します。ファイルが RLS モードでオープンされている場合は、UNCOMMITTED、CONSISTENT、REPEATABLE および NOSUSPEND の各オプションを使用し、読み取りの保全性を制御できます。このトピックでは、ブラウズに関する特別なケースがいくつか示されています。

以下のように、ブラウズでは直接読み取りと同じタイプのキーが使用されます。

- KSDS の場合は、全キー、総称キー、または代替キーを使用できます。
- 標準 (非拡張) ESDS の場合は、RBA または代替キーが使用可能です。
- 拡張 ESDS の場合は、拡張 RBA (XRBA) を使用できます。

  注: 環境によっては、32 ビット RBA を使用して拡張 (64 ビット) ESDS にアクセスできます。 546 ページの『相対レコード番号 (RRN)、相対バイト・アドレス (RBA) または拡張相対バイト・アドレス (XRBA)』を参照してください。
- RRDS の場合は、RRN が使用されます。

ブラウズに代替キーを使用する場合、レコードは代替キーの順序で取得されます。代替キーが固有でない場合には、重複キーの最終オカレンスを除き、検索操作のたびに DUPKEY 条件が起こります。例えば、同一代替キーを持つレコードが 3 レコードある場合には、最初の 2 レコードに対しては DUPKEY が起こりますが、3 レコード目には起こりません。重複した代替キーがあるレコードが返される順序は、レコードがデータ・セットに書き込まれた順序です。READNEXT コマンドを使用していても READPREV コマンドを使用していても、同様です。このために、同一の代替キーを持つレコードを逆順に検索することはできません。

CICS によって、トランザクションは、同一ファイルに対する複数のブラウズを同時に実行することができます。各ブラウズ・コマンドに REQID オプションを組み込んで、ブラウズ操作を区別します。

このトピックでは、ブラウズに関する一般的な原則、およびさまざまなタイプの VSAM データ・セットに関する固有の情報について説明します。BDAM データ・セットのブラウズに関する固有の情報については、575 ページの『BDAM データ・セットからのレコードのブラウズ』を参照してください。

- ブラウズを開始するには、STARTBR コマンドを使用します。 STARTBR コマンドはブラウズの開始位置を特定するのみであり、レコードを検索するわけではありません。 直接読み取りと同じ方法で特定のレコードを識別し、RIDFLD オプションを使用してレコード ID を指定します。567 ページの『直接読み取り (READ コマンドの使用)』 を参照し、また、以下の考慮事項についても注意してください。
  1. ファイルの開始点にブラウズ位置を指定するには、KSDS または ESDS の場合は 16 進数でゼロの RIDFLD を指定します。 標準 ESDS の場合は RBA オプションも指定します。拡張 ESDS の場合は、XRBA オプションを指定します。
  2. ファイルの開始点にブラウズ位置を指定するには、RRDS の場合は RIDFLD オプションを使用して 1 の RRN を指定し、RRN オプションも指定します。
  3. KSDS の場合のみ、ファイルの開始点にブラウズ位置を指定する代わりの方法として、GENERIC、GTEQ、および KEYLENGTH(0) を指定できます。 その値が使用されない場合でも RIDFLD キーワードが必要であり、そのコマンドの完了後に、CICS はその総称キーの長さを使用します。 ブラウズのコマンドに KEYLENGTH(0) オプションを指定した場合は常に、KEYLENGTH(1) および 2 進ゼロの 1 バイトの部分キーが指定されているように扱われます。
  4. ファイルの最終レコードにブラウズ位置を指定し、逆方向のブラウズを使用可能にするには、RIDFLD に X'FF' の文字を指定します。 これは、KSDS、ESDS、または RRDS のブラウズの開始に適用されます。標準 ESDS の場合は RBA オプションを指定します。拡張 ESDS の場合は、XRBA オプションを指定します。RRDS の場合は、RRN オプションを指定します。
  5. KSDS のブラウズの開始では、RIDFLD オプションを使用して総称キーを指定できます。 ただし、総称キーを使用する場合は、ファイル全体で順方向のブラウズのみが可能であり、逆方向のブラウズはできません。
  6. KSDS のブラウズを開始する場合は、「EQUAL (キーが等しい)」および「GTEQ (キーが大きいか等しい)」の各オプションが使用可能であり、READ コマンドの場合と意味は同じです。 ただし、GTEQ オプションが STARTBR コマンドのデフォルト値であるのに対し、EQUAL オプションは READ コマ

ンドのデフォルト値です。STARTBR コマンドでは、指定したキーの位置に位置決めするか、または、そのキーがない場合はそれより大きい最初のキーの位置に位置決めします。

7. RRDS のブラウズを開始する場合は、「GTEQ (キーが大きいか等しい)」オプションが STARTBR コマンドのデフォルトです。 このオプションが有効である場合、指定された RRN が存在しないと、それより大きいキーの最初のレコードにポインターが設定されます。 GTEQ オプションは、RRDS に対する直接 READ コマンドにおいては有効ではないことに注意してください。直接 READ コマンドで存在しない RRN が指定されると、NOTFND 状態が返されます。
8.

- 複数のレコードをブラウズするには、READNEXT コマンドを使用します。 READNEXT コマンドは、STARTBR コマンドで指定された開始位置からレコードを順次に読み取ります。これは、直接読み取りと同じように作動します。 567 ページの『直接読み取り (READ コマンドの使用)』 を参照し、また、以下の考慮事項についても注意してください。
  1. RIDFLD オプションを指定し、CICS に取得した各レコードの ID を返す方法を指示しますが、ブラウズを位置変更する場合を除いて、そのフィールドは設定しないでください。 それぞれの READNEXT コマンドの完了時には、CICS はそのコマンドが取得したレコードの全キーを返します。より短い総称キーを指定して STARTBR コマンドを使用する場合でも、必ず、全キーと同じ長さのフィールドを指定してください。
  2. RIDFLD オプションを使用してキーを指定し、ブラウズの位置を変更する場合は、KEYLENGTH オプションのみが必要です。 その他の場合は、STARTBR コマンドでの設定時、およびブラウズの位置の最後の変更時に、現行キーの長さが使用されます。
  3. 直接読み取りの場合と同様に、レコードは、(INTO オプションを使用している場合には) アプリケーション・プログラムが提供する区域に、または (SET オプションを使用している場合には) CICS 提供のストレージに、読み取ることができます。 後者の場合には、SET ポインターによってアドレッシングされた CICS ストレージは、ブラウズの次の操作、あるいはブラウズの終了、同期点、またはタスクの終了のいずれかが最初に発生するまで有効です。
- ファイルを逆方向にブラウズするには、READNEXT コマンドではなく、READPREV コマンドを使用します。 READPREV コマンドは、レコードが現在位置から**逆方向**に順次に読み取られる点を除き、READNEXT コマンドと類似しています。ある方向から逆の方向に、いつでも切り替えできます。ある方向から別の方向に切り替えるときは、位置指定し直さない限り、同一レコードが二度取り出されます。

  注: KSDS の場合、STARTBR コマンドで総称キーを使用すると、READPREV コマンドが使用できないことに注意してください。この場合に READPREV コマンドを使用すると、INVREQ 状態が返されます。
- ブラウズの開始後に現行のブラウズ位置を変更するには (ブラウズの位置変更)、RESETBR コマンド、READNEXT コマンド、または READPREV コマンドを使用します。

1. VSAM の場合、次回の READNEXT コマンドまたは READPREV コマンドの発行時に、RIDFLD に値を設定することにより、ブラウズの位置を変更できます。
   - ブラウズの位置変更で RIDFLD を設定する場合、レコード ID は直前の STARTBR または RESETBR コマンドと同じ形式にする必要があります (キー、RBA、XRBA、または RRN)。キーまたは 16 進数のゼロを使用し、ファイルの開始点を示すか、または KSDS の場合は、GENERIC、GTEQ、および KEYLENGTH(0) の各オプションを指定してファイルの開始点を示すことができます。KEYLENGTH(0) オプションを使用する場合、RIDFLD キーワードは値が使用されていなくても必要であることに注意してください。コマンドの完了後に、CICS はその総称キーの長さを使用します。(X'FF' 文字は、READNEXT または READPREV コマンドによる位置変更には使用できません。)
   - READPREV コマンドの KEYLENGTH に、現行の総称キーとは違った長さで、かつ全体の長さとも等しくない値を指定することにより、総称キーの長さを変更できます。この方法で総称キーの長さを変更した場合には、RIDFLD オプションによって指定された総称キーに位置変更されます。

   ブラウズの特性も変更する場合は、代わりに RESETBR コマンドを使用します。
2. RESETBR コマンドでは、STARTBR コマンドと同じ方法で新規のブラウズ位置を指定できます。 以下のことを行うことができます。
   - 特定のレコードを識別する。
   - 16 進ゼロのキーを使用してファイルの開始点を示す。
   - X'FF' 文字のキーを使用してファイルの終了点を示す。
   - KSDS の場合は、GENERIC、GTEQ、および KEYLENGTH(0) を各オプションを使用して、ファイルの開始点を示す。KEYLENGTH(0) オプションを使用する場合、RIDFLD キーワードは値が使用されていなくても必要であることに注意してください。コマンドの完了後に、CICS はその総称キーの長さを使用します。

- ブラウズの特性を変更するには (例えば、完全一致突き合わせではなく、総称キーで検索するなど)、RESETBR コマンドを使用します。 このコマンドでは以下を実行できます。
  - キーの形式をキーから RBA に変更する。
  - 総称キーと全キーの切り替え、または検索の「等しい」と「大きいか等しい」の切り替え。

  一定の条件で、576 ページの『スキップ順次処理』 に説明されているように、キーを変更すると、CICS は VSAM スキップ順次処理を使用します。
- ブラウズを終了するには、ENDBR コマンドを使用します。このコマンドには、RIDFLD はありません。 ファイル内の最後のレコードを通り越してブラウズしようとすると ENDFILE 条件が起こります。ENDBR コマンドは、同一ファイルに対する更新操作 (READ UPDATE コマンド、RIDFLD オプションを指定した DELETE コマンド、または WRITE コマンド) を実行する前に発行する必要があります。更新操作の前に発行しなかった場合には、ユーザー自身のトランザクション内でのデッドロックも含めて、予測不能な結果になります。

BDAM データ・セットからのレコードのブラウズ

レコード識別フィールドには、そのデータ・セットに定義されたアドレッシング方式に適合したブロック参照 (例えば、TTR または MBBCCHHR など) が含まれていなければなりません。処理は、指定されたブロックから始まり、ブラウズが終了するまで、各後続のブロックで続行されます。

データ・セットにブロック化レコードが入っている場合は、処理は最初のブロックの最初のレコードから始まり、レコード識別フィールドの内容に関係なく、後続レコードをそれぞれ処理していきます。

すなわち、CICS は RIDFLD の TTR サブフィールドまたは MBBCCHHR サブフィールドに保持されている情報のみを使用して、レコードを識別します。CICS は、物理キーおよび相対レコード、あるいは論理キーなどの他のすべての情報を無視します。

READNEXT コマンドの後で、CICS は RIDFLD フィールドを、取得したレコードを完全に識別して更新します。例えば、ブラウズがブロック化されたキー付きデータ・セットの最初のレコードから開始され、論理キーによる非ブロック化が実行されるものとします。 STARTBR コマンドを実行する前に、最初のブロックの TTR (これはアドレッシング方式とします) をレコード識別フィールドに入れます。最初の READNEXT コマンドの後、レコード識別フィールドには X'0000010504' が収容されています。この場合の X'000001' は TTR 値を、X'05' は (長さが 1 の) ブロック・キーを、X'04' は論理レコード・キーをそれぞれ表します。

ここで、ブロック化されたキーなしデータ・セットが相対レコード非ブロック化を使用してブラウズされており、3 番目の相対トラック上の 2 番目の物理ブロックからの 2 番目のレコードがコマンドにより読み取られるとすると想定し、最初のブロックの TTR (これはアドレッシング方式とします) をレコード識別フィールドに入れます。最初の READNEXT コマンドの後、アプリケーション・プログラムに戻るときには、レコード識別フィールドには X'00020201' が収容されています。この場合の X'0002' はトラックを、X'02' はブロックを、X'01' はそのブロック内のレコードのゼロからの相対的な数をそれぞれ表します。

注: ブロック化データ・セットをブラウズする場合には、STARTBR コマンドに DEBREC オプションおよび DEBKEY オプションを指定してください。これにより、CICS は RIDFLD に正しい値を返すことができます。 STARTBR コマンドに DEBREC を指定すると、相対レコード番号が返されます。STARTBR コマンドに DEBKEY を指定すると、論理レコード・キーが返されます。

ブロック化ファイルのブラウズ時には、DEBREC または DEBKEY を省略しないでください。省略した場合には、論理レコードはブロックから検索され、長さパラメーターは論理レコード長と等しく設定されます。***しかし***、RIDFLD はレコードの完全な識別によって更新されません。***この方法は使用しないでください。***

このことと、ブロック化 BDAM データ・セットからの読み取り時に DEBREC または DEBKEY オプションを省略した場合に起こる事態を比較してください。この場合には、ブロック***全体*** が検索され、長さパラメーターはそのブロックの長さと等しく設定されます。

リモート BDAM ファイルで、DEBKEY オプションまたは DEBREC オプションが指定されている場合は、KEYLENGTH (明示指定する場合) はキーの全長 (すなわち、指定されたすべてのサブフィールドの長さ) にする必要があります。

スキップ順次処理

可能な場合には、CICS は VSAM「スキップ順次」処理を使用してブラウズを高速化します。スキップ順次処理がファイルの正方向ブラウズに適用されるのは、キー形式に RIDFLD を指定した場合だけです。 CICS がこれを使用するのは、READNEXT コマンドの RIDFLD のキー値を大きくして、KEYLENGTH などの他のキー関連の設定を指定しない場合です。この状態では、VSAM は、スクラッチから位置指定し直すのではなく、索引を通じて正方向読み取りを行うことによって、次に必要とするレコードを見つけます。この方式が高速なのは、検索しているレコードが相互に比較的接近しているが、必ずしも隣接していない場合です。この方式が逆の効果をもつのは、レコードが大容量ファイルの中で非常に離れている場合です。位置変更するキーがファイル内でははるかに高い場合や、長い索引スキャンになる可能性があることがわかっている場合は、強制的に最初から位置変更する RESETBR コマンドの使用を考慮した方が良い場合があります。

CICS コマンドを使用したレコードの更新

このタスクについて

レコードを更新するためには、最初に UPDATE オプションを指定したファイル制御読み取りコマンドを使ってそのレコードを検索しなければなりません。レコードは直接読み取りとまったく同じ方法で識別されます。KSDS または ESDS では、レコードは (直接読み取りを使用する場合と同様に)、基本データ・セットまたはそこに定義されたパスを指すファイル定義によってアクセスすることができます。RLS モードでファイルをオープンした場合、EXEC CICS コマンドで UPDATE オプションの他に NOSUSPEND オプションを指定して、レコードを VSAM のアクティブ・ロックによりロックしておけば、要求が待機することは確実になくなります。

アプリケーション・プログラムにより変更された後、レコードは、REWRITE コマンドを使ってデータ・セットに再び書き込まれます。このコマンドは再書き込みされたレコードを識別しません。ある作業単位内において、各 REWRITE コマンドは、共通キーワード (TOKEN) によって直前の READ UPDATE と関連付ける必要があります。同時に未解決の TOKEN がない、単一の READ UPDATE も使用できます。TOKEN を使用しない同じデータ・セットに対し、作業単位内で複数の更新要求を並行して実行しようとすると、CICS により阻止されます。レコードを再書き込みまたは削除しないで、READ UPDATE によって保持されているストリングを解放する場合には、UNLOCK コマンドを使用します。UNLOCK コマンドは、READ コマンド用に獲得されたすべての CICS ストレージ、および READ コマンドが保持している VSAM リソースを解放します。TOKEN が UNLOCK コマンドで指定されている場合、CICS は、これを同じ値の TOKEN を持つ未解決の READ UPDATE と一致させようとします。(TOKEN オプションの詳細な説明は、577 ページの『TOKEN オプション』を参照してください)。

更新コマンドおよび非更新コマンドのどちらの場合も、RIDFLD オプションに指定されたレコード識別フィールドで、検索するレコードを識別する必要があります。READ UPDATE コマンドが完了するとただちに、アプリケーション・プログラムで

RIDFLD データ域を再利用することができます。REWRITE コマンドおよび UNLOCK コマンドでは、RIDFLD オプションは使用しません。

ブラウズ操作の途中で検索されたレコードをブラウズ中に更新することができるのは、ファイルが RLS モードでオープンされている場合だけです (580 ページの『ブラウズ中のレコードの更新および削除 (VSAM RLS のみ)』を参照してください)。非 RLS モードでオープンされたファイルの場合、アプリケーション・プログラムはブラウズを終了し、READ UPDATE コマンドを使用して必要なレコードを読み取り、更新を実行します。READ UPDATE コマンドを発行する前のブラウズの終了で失敗すると、デッドロックを引き起こすことがあります。

更新するレコードは (直接読み取りの場合と同様に) アプリケーション・プログラムが用意しているストレージ、または CICS により設定されているストレージに読み取ることができます。 READ UPDATE コマンドの場合の CICS SET ストレージは、次に REWRITE、UNLOCK、RIDFLD を指定していない DELETE、または SYNCPOINT コマンドのいずれかが最初に発行されるまで、確保されます。

KSDS の場合には、レコードの変更時に、レコードの基本キーを変更してはいけません。同様に、パスによる変更の場合には、他の代替キーを変更することがあっても、レコードを識別するために使用している代替キーを変更してはいけません。ファイル定義に可変長レコードを使用できる場合には、レコードの長さが変更されることがあります。

FROM オプションを使用し、書き込むレコードを指定します。FROM オプションは、書き込むレコードが入っている主記憶装置の区域を指定します。一般に、この区域はユーザー・アプリケーション・プログラムが所有するストレージの一部です。 REWRITE コマンドの FROM 区域は、通常、READ UPDATE コマンド上の対応する INTO 区域と同じです (しかし、必ずではありません)。

ESDS、固定長 RRDS、または固定長 KSDS のレコードの長さを更新時に変更してはいけません。ただし、レコードの長さは、可変長レコードの KSDS への再書き込み時に変更可能です。

固定長レコードが入るものとして CICS に定義されたファイルの場合には、再書き込みするレコードの長さは***元の長さと等しくなければなりません***。ただし、固定長レコードでファイルに書き込む場合は、LENGTH オプションを指定する必要はありません。LENGTH オプションを指定した場合には、その値は定義済みの値と比較検査され、値が一致しない場合には、LENGERR 条件が起こります。

可変長レコードの場合には、READ UPDATE コマンドおよび REWRITE コマンドの両方に LENGTH オプションを指定しなければなりません。長さは VSAM に定義されている最大長より大きくしてはいけません。

TOKEN オプション

TOKEN オプションは、更新のための読み取りコマンドにおいて CICS が提供するタスク内の固有の値であり、関連する REWRITE、DELETE、または UNLOCK コマンドで CICS にこれを返します。各ファイルがタスクにより更新されている場合には、TOKEN オプションを指定しないで、UPDATE オプションを指定した未解決の読み取り要求を同時に 1 つだけ持つことができます。

更新のための読み取りコマンドに TOKEN オプションを組み込み、関連する REWRITE、DELETE、または UNLOCK コマンドでトークンを指定することにより、同一タスクを使用する同一データ・セットに対して複数の更新を並行して実行できます。非 RLS モードでアクセスされるファイルの場合、要求と関連付けられた TOKEN に関係なく、複数のレコードが同一 CI で更新されていると、一組の並列更新は失敗します。また、ブラウズ上の指定の REQID に対し有効のまま残るトークンは 1 つのみであり、そのトークンが最終の READNEXT または READPREV コマンドで返されます (580 ページの『ブラウズ中のレコードの更新および削除 (VSAM RLS のみ)』を参照してください)。

TOKEN オプションを含む更新のための読み取り要求を機能伝送することができます。しかし、このキーワードを認識していない CICS ファミリー・プロダクトのメンバーに TOKEN を指定して要求を機能伝送しても、要求は失敗します。

条件付き VSAM ファイル更新要求

RLS モードでオープンされたファイルに対するファイル制御更新要求の場合、即時にロックを与えて要求を条件付きにすることにより、ロックのために待機しないで済みます。これを行うためには、要求に NOSUSPEND オプションを指定します。別のタスクが既にアクティブ・ロックを保留している場合には、CICS は要求をキューイングしないで、RECORDBUSY 条件を返します。

NOSUSPEND は、READ、READNEXT、READPREV、WRITE、REWRITE、および DELETE の各コマンドに指定できます。

LOCKED 応答と RECORDBUSY 応答を区別することは重要です。

- LOCKED 応答が発生するのは、要求が*保存* ロックによりロックされているレコードにアクセスしようとする場合です。
- RECORDBUSY 応答が発生するのは、要求がアクティブ・ロックによりロックされているレコードにアクセスしようとする場合です。これは、DEADLOCK が原因で発生する可能性があり、その場合は再試行が無効になる場合があるので注意してください。状態を解決するために、ロールバックを伴う、あるいはロールバックを伴わない SYNCPOINT を出す必要があるかもしれません。

  注: NOSUSPEND を指定する要求は、CICS が RECORDBUSY 応答を返すまで少なくとも 1 秒は待機します。

要求に NOSUSPEND が指定されていない場合に、レコードが既にアクティブ・ロックによりロックされていれば、CICS は要求をロックのために待機させます。NOSUSPEND を指定している場合に、レコードがアクティブ・ロックによりロックされると、要求は RECORDBUSY 応答を受け取ります。

保存ロックによりロックされたレコードに対し、要求 (NOSUSPEND オプションを指定している場合も指定していない場合もある) を発行すると、CICS は LOCKED 応答を返します。

NOSUSPEND の動作は、ファイル制御コマンドにおける場合と他の CICS コマンドにおける場合とで比較すると多少違います。 HANDLE CONDITION (RECORDBUSY) を発行した場合、後続のファイル制御要求で NOSUSPEND とみな

されることはありません。一方、HANDLE CONDITION(QBUSY) を指定した場合は、NOSUSPEND が明示的に指定されていなくても、後続の一時データ・コマンドでは NOSUSPEND とみなされます。

BDAM データ・セットからのレコードの更新

非ブロック化を指定する REWRITE コマンドで、可変ブロック化または非ブロック化の BDAM レコードのレコード長を変更することはできません。REWRITE コマンドで不定形式 BDAM レコードのレコード長を変更することもできません。

注: ブロック化 BDAM レコードを更新用に読み取る場合、CICS は包含ブロックの排他制御を維持します。ブロックからの最初のレコードを (REWRITE コマンドによって) 更新する前、あるいは排他制御を (UNLOCK コマンドによって) 解放する前に、2 番目のレコードを読み取ろうとするとデッドロックになります。

CICS コマンドを使用したレコードの削除

ファイルからレコードまたはレコードのグループを削除するには、DELETE コマンドを使用します。更新用に最初にレコードを取り出す場合があります。RLS モードでオープンされたファイルの場合は、ブラウズ時にレコードを削除することもあります。

このタスクについて

DELETE コマンドで絶対キーを指定した場合には、そのキーの単一レコードが削除されます。したがって、非固有代替キーが使用可能な代替索引パスによってデータ・セットにアクセスしている場合、そのキーを持つ最初のレコードのみが削除されます。削除後、同一の代替キーを持つレコードがまだ存在する場合には、DUPKEY 条件になるので、そのようなレコードが存在するかどうかわかります。

567 ページの『直接読み取り (READ コマンドの使用)』 に説明されている NOSUSPEND オプションは、ファイルの更新に使用する場合の CICS ブラウズ・コマンドに適用されます。

- ESDS からレコードを削除することは絶対にできません。
- KSDS または RRDS にある単一のレコードを削除するには、以下の 3 つの方法のいずれかを使用してください。
  1. READ UPDATE コマンドを使用して更新用にレコードを取り出し、次に RIDFLD オプションを指定せずに DELETE コマンドを発行します。
  2. RIDFLD オプションを指定した DELETE コマンドを発行します。
  3. RLS モードでオープンしたファイルの場合には、UPDATE オプションを指定した READNEXT または READPREV コマンドを使用してレコードを取り出し、次に DELETE コマンドを発行します。この方法については、580 ページの『ブラウズ中のレコードの更新および削除 (VSAM RLS のみ)』で説明します。
- KSDS または RRDS にあるレコードのグループを削除するには、総称キーを指定して DELETE コマンドを使用します。 単一のレコードが削除されるのではなく、そのファイル内の総称キーと一致するすべてのレコードが単一コマンドによって削除されます。代替索引パスによりアクセスする場合に、削除されるレコードはすべて代替キーが総称キーと一致するレコードです。ただし、KEYLENGTH

の値がキー全体の長さと等しい場合は、(重複キーが許可されていても)、この方法を使用できません。このコマンドに NUMREC オプションが組み込まれている場合には、削除されたレコード数がアプリケーション・プログラムに返されます。

ブラウズ中のレコードの更新および削除 (VSAM RLS のみ)

このタスクについて

RLS モードでアクセスされるファイルの場合、READNEXT または READPREV コマンドに UPDATE オプションを指定して、次に、DELETE または REWRITE コマンドを発行することにより、レコードを更新または削除することができます。ブラウズ・コマンドが TOKEN を返す場合、その TOKEN は次のブラウズ要求があるまで有効です。TOKEN は TOKEN に対し同じ値を指定する REWRITE、DELETE、または UNLOCK コマンド、あるいは、同じ REQID を指定する READNEXT、READPREV、または ENDBR コマンドの発行により無効となります。UPDATE および TOKEN オプションを指定した 複数の READNEXT コマンドを発行すると、TOKEN は互いに無効にし合うので、最後の TOKEN のみが使用可能になります。(TOKEN オプションの詳細な説明は、577 ページの『TOKEN オプション』を参照してください)。

ブラウズにおいて UPDATE オプションを使用する場合は、以下の規則に従ってください。

- ブラウズ内で UPDATE を指定できるのは、ファイルが RLS モードでアクセスされる場合だけです。非 RLS モードでアクセスされるファイルに対し UPDATE を指定すると、CICS は INVREQ 条件を返します。
- UPDATE が指定できるのは、READNEXT および READPREV コマンドの場合のみであり、STARTBR または RESETBR コマンドでは指定できません。
- CICS は、ブラウズ・シーケンスにおいて 1 つの TOKEN のみをサポートします。READNEXT または READPREV コマンドのそれぞれの TOKEN の値により、前の値が上書きされます。
- 同一ブラウズ内では、更新要求と非更新要求を混在させることができます。
- READNEXT、DELETE または UNLOCK コマンドにおいて、対応する READNEXT または READPREV コマンドによって返される TOKEN を指定する必要があります。

UPDATE のロック

READNEXT または READPREV コマンドに UPDATE を指定することによって、排他ロックが獲得されます。ブラウズ内でのこれらの排他ロックの期間は、アプリケーション・プログラムがとる処置およびファイルがリカバリー可能かどうかによって決まります。

- ファイルがリカバリー可能であり、ブラウズでの更新読み取りにより獲得した最後のレコードを (関連付けられたトークンを使用して) DELETE または REWRITE する場合、VSAM 排他ロックは、UOW が完了するまで活動状態のままです。つまり、同期点またはロールバックが正常に完了するまでということです。
- ファイルがリカバリー不能であり、獲得した最終レコードを DELETE または REWRITE する場合、次に ENDBR コマンドを発行するか、続いて READNEXT

または READPREV コマンドを発行する場合に、ロックが解放されます。これについての詳細な説明は、 550 ページの『RLS レコード・レベルのロック』にあります。

- 読み取った最終レコードを更新しない 場合は、プログラムがブラウズにおいて別の READNEXT または READPREV コマンドを発行するか、ブラウズを終了する時点で、CICS は排他ロックを解放します。

注: UNLOCK コマンドは、ブラウズ操作中に獲得したレコードに対し VSAM が保持する RLS 排他ロックを解放しません。ブラウズ内で UNLOCK コマンドを発行すると、最終要求によって返された TOKEN を無効にします。ブラウズ内で別に READNEXT または READPREV コマンドを発行すると、直前の READNEXT または READPREV UPDATE コマンドによって読み込まれたレコードに対する TOKEN も無効にします。したがって、特別なレコードを更新しないようにしたアプリケーション・プログラムでは、UNLOCK を使用する必要はありません。

CICS コマンドを使用したレコードの追加

ファイルに新規レコードを追加するには、WRITE コマンドを使用します。

このタスクについて

新規レコードは、常にアプリケーション・プログラムによって用意された区域から書き込まれます。

このトピックでは、レコードの書き込みに関する一般的な原則、およびさまざまなタイプの VSAM データ・セットに関する固有の情報について説明します。BDAM データ・セットへのレコードの追加に関する固有の情報については、 583 ページの『BDAM データ・セットへのレコードの追加』を参照してください。

WRITE コマンドを使用する場合は、以下のようにします。

- KSDS にレコードを追加する場合は、RIDFLD (レコード識別フィールド) オプションを使用してそのレコードの基本キーを指定します。 レコードの基本キーにより、そのレコードが挿入されているデータ・セット内の位置が識別されます。そのキーはレコードの一部ですが、CICS ではアプリケーション・プログラムでもキーを別に指定する必要があります。
- ESDS 基本データ・セットにレコードを追加する場合は、そのレコードをファイルの最後に追加する必要があります。 ESDS では、既存のレコードの間にレコードを挿入することはできません。操作が完了した後で、レコードが入っているファイル内の相対バイト・アドレスがアプリケーション・プログラムに返されます。
- 代替索引パスの方法により KSDS または標準 ESDS にレコードを追加する場合は、RIDFLD オプションを使用して代替キーを指定します。o KSDS の場合、レコードはデータ・セット内の基本キーで決められた位置に挿入されます。ESDS の場合、レコードはデータ・セットの最後に配置されます。

  注: 拡張 ESDS データ・セットでは、代替索引はサポートされていません。
- RRDS にレコードを追加する場合は、RIDFLD オプションを使用して相対レコード番号を指定します。 レコードは、その RRN に対応するファイル内の位置に保管されます。

- 必要に応じて KEYLENGTH オプションを指定し、RIDFLD オプションで指定したキーの長さを指定します。
  1. レコード識別フィールドに RBA、XRBA、または RRN を指定し、RBA、XRBA、または RRN オプションを指定した場合、KEYLENGTH オプションは**必要ありません**。
  2. その他の場合は、KEYLENGTH オプションが**必要です**。 データ・セットに定義されている長さとは異なる長さの KEYLENGTH を指定した場合は、INVREQ 状態が発生します。
- FROM オプションを使用し、書き込むレコードが収容されている主記憶域を指定します。 一般に、この区域はユーザー・アプリケーション・プログラムが所有するストレージの一部です。
- 必要に応じて LENGTH オプションを指定します。
  1. 固定長レコードでファイルに書き込む場合、LENGTH オプションは必要ありません。 CICS は、書き込むレコードの長さとしてクラスター定義に指定されている長さを使用します。LENGTH オプションを指定した場合、その値は定義済みの値と比較検査され、値が一致しないときは LENGERR 状態が発生します。
  2. 可変長レコードでファイルを書き込む場合は、LENGTH オプションを常に指定する必要があります。 指定された値がクラスター定義で使用できる最大値を超えた場合には、コマンドを実行した時に LENGERR が起こります。可変長レコードを持つファイルにアクセスする場合、LENGTH オプションが省略されても、LENGERR が起こります。
- ファイルが RLS モードでオープンされ、レコードが既にロックされている可能性がある場合は、NOSUSPEND オプションを指定できます。 NOSUSPEND オプションを指定すると、アクティブ・ロックがある VSAM によりレコードがロックされている場合に要求が待機しないようになります。 552 ページの『ロックのアクティブ状態および保存状態』 には、アクティブ・ロックについて説明されています。
- 複数のレコードを KSDS、ESDS、またはパスに対して追加し、連続する要求におけるキーが昇順である場合は、MASSINSERT オプションを使用するとパフォーマンスが向上します。 (このパフォーマンスの向上は、キーが昇順である場合にのみ得られます。)
  1. シーケンス内の各 WRITE コマンドで MASSINSERT オプションを指定します。
  2. MASSINSERT の操作が完了し、すべてのレコードが書き込まれたら、UNLOCK コマンドを発行し、そのすべてのレコードがファイルに書き込まれてその位置が解放されるようにします。 UNLOCK コマンドは常に、同一データ・セットに対する更新操作 (READ UPDATE コマンド、RIDFLD オプションを指定した DELETE コマンド、または WRITE コマンド) を実行する前に発行する必要があります。更新操作の前にこのコマンドを発行しなかった場合には、デッドロックを含め、予測不能な結果になることがあります。
     UNLOCK コマンドを発行しない場合、MASSINSERT は、同期点が発行されたとき、またはタスク終了の時点で完了します。

     注: MASSINSERT 操作が完了していなかった場合、追加されたレコードは、READ コマンドで必ずしも検索されません。

MASSINSERT の詳細については、 584 ページの『VSAM データ・セット』を参照してください。

ESDS に書き込む場合の CICS ロック

RLS モード・アクセスの場合も非 RLS モード・アクセスの場合も、ESDS への CICS 書き込み操作は単一スレッド化されます。しかし、逐次化で保留されているロックは、非 RLS モードに比べ RLS モード・アクセスの場合、若干長く保留される可能性があります。ESDS ファイルに新しいレコードを追加するトランザクションのタスク優先順位を上げることにより、オーバーヘッドの見込まれる増加を補正することができます。ESDS を非 RLS モードから RLS モードに切り替える場合、新しいレコードを追加するこれらのトランザクションのタイムアウトが引き上げられている可能性があります。

また、ESDS で RLS モード・アクセスを使用すると、可用性の問題が発生することもあります。ESDS への書き込み中に CICS 領域に障害が発生すると、CICS 領域が再始動するまでデータ・セットがロックされる可能性があります。ESDS には RLS モード・アクセスを使用しないことをお勧めします。

BDAM データ・セットへのレコードの追加

このタスクについて

BDAM データ・セットにレコードを追加する場合は、下記の点に留意してください。

- 未定義または可変長 (キー付きまたはキーなし) のレコードを追加する場合には、各新規レコードを追加するトラックを指定しなければなりません。トラック上のスペースが使用可能な場合には、レコードは直前に書き込まれた最終レコードの後に書き込まれ、レコード番号がレコードのレコード識別フィールドの「R」部分に入れられます。トラック指定は、相対ブロックを除き、任意の形式にすることができます。ゾーン 10 進数相対形式を使用する場合には、レコード番号は、レコード識別フィールドの 7 〜 8 桁目に 2 バイトのゾーン 10 進数として返されます。

  指定のトラック上に使用可能スペースがない場合には、拡張検索オプションによって、レコードを別のトラックに追加することができます。レコードが追加される位置は、使用しているレコード識別フィールドに入れてアプリケーション・プログラムに返されます。

  未定義の長さのレコードを追加する場合には、LENGTH オプションを使用してレコードの長さを指定します。未定義のレコードを検索する場合には、アプリケーション・プログラムがその長さを判別しなければなりません。
- キー付き固定長レコードを追加する場合には、最初に、ダミー・レコードまたはレコードを追加できる「スロット」によってデータ・セットを形式設定する必要があります。ダミー・レコードは X'FF' のキーによって指定します。データの最初のバイトにはレコード番号を入れます。
- キーなし固定長レコードを追加する場合には、レコード識別フィールドにブロック参照を指定します。新規レコードは指定された位置に書き込まれ、その位置の直前の内容は破棄されます。

- キー付き固定長レコードを追加する場合には、トラック情報のみを使用してダミー・キーおよびレコードが検索され、見つかったレコードが、見つかった時点で新規キーおよびレコードによって置き換えられます。新規レコードの位置はレコード識別フィールドのブロック参照サブフィールドに入れてアプリケーション・プログラムに返されます。

  例えば、次の識別フィールドを持つレコードについて考えてみます。

  ```
  0 3 0  ALPHA
  T T R  KEY
  ```

  この場合は、検索は相対トラック 3 から開始されます。制御がアプリケーション・プログラムに戻った時点のレコード識別フィールドは、次のとおりです。

  ```
  0 4 6  ALPHA
  ```

  これは、レコードが相対トラック 4 にあるレコード 6 であることを示しています。
- 可変長ブロック化レコードを追加する場合には、各レコードに 4 バイトのレコード記述フィールド (RDF) を含めなければなりません。最初の 2 バイトには (4 バイトの RDF を含む) レコードの長さを指定し、残りの 2 バイトはゼロにします。

効率的なデータ・セットの操作

データベース操作およびデータ・セット操作の効率は、CICS システムのパフォーマンスにおいて重要な要素となります。VSAM では、効率に対する主な影響、したがって応答時間に対する主な影響は、順次使用リソース (レコード・キー、制御インターバル、およびストリング)、および使用ストレージとプロセッサー・オーバーヘッドに対する競合を引き起こします。これらの状態の常として、ある分野が改良できても、別の分野を犠牲にしてしまうことがあります。

VSAM データ・セット

VSAM データ・セットを使用する場合に、競合による遅れを最小に抑えるには、次のようにします。

- VSAM リソースが排他使用のために予約される時間を最小に抑えてください。**排他使用**エンキューは、CICS と VSAM が並行更新しないようにする方法です。

  544 ページの『RLS モードでのファイルのアクセス』 に説明されている VSAM レコード・レベル共用を使用する場合、VSAM は更新を要求されているレコードをロックし、他のトランザクションがそのレコードを同時に更新できないようにします。ファイルがリカバリー可能な場合は、VSAM は次の同期点でロックを解放します。ファイルがリカバリー可能でない場合は、VSAM は要求が完了するとロックを解放します。ファイルのリカバリー可能性は、統合カタログ機能 (ICF) のカタログで定義されます。

  VSAM レコード・レベル共用を使用しない場合は、CICS は、基本クラスター・レコード・キーによって更新要求を逐次化します。要求されたレコードに対して個別のコマンド (例えば、UPDATE オプション付きの READ コマンドなど) が実行されている間、そのレコードが含まれる全 VSAM 制御インターバル (CI) が

排他使用のために保持されます。それぞれのコマンドが完了するごとに制御インターバルは解放され、要求されたレコードのみがロック状態のままです。**リカバリー不能**データ・セットの場合、レコードの VSAM 排他使用および CICS 排他使用の両方は、VSAM において更新要求が完了すると (例えば、REWRITE コマンドが完了した時など) 終了します。しかし、**リカバリー可能データ・セット**の場合には、タスクが終了するか、あるいはタスクが SYNCPOINT コマンドを発行するまで、CICS 排他使用は終了しません。リカバリー可能性は、データ・セット・リソース定義で指定されます。FILE リソース定義について詳しくは、「*CICS Resource Definition Guide*」のFILE 定義属性を参照してください。

- VSAM データ・セット内での位置の保留は、できるだけ短い時間にしてください。表 28 は、位置を保留するコマンドおよび保留が解放される時点を示しています。

表 28. 位置を保留するコマンドおよび保留が解放される時点

コマンド	VSAM が解放する時点
READ.. UPDATE	REWRITE/DELETE/UNLOCK
WRITE.. MASSINSERT	UNLOCK
STARTBR	ENDBR

VSAM データ・セットに対して進行中の各要求には、少なくとも 1 つのストリングが必要です。位置を保留する要求は、保留位置を解放するためにコマンドが発行されるまで、ストリングを拘束します。位置を保留しない要求は、その要求が完了すると同時にストリングを解放します。

VSAM データ・セットを使用する場合のプロセッサー・オーバーヘッドを最小限にするには、以下のようにします。

- 多くのレコードを順々に追加している場合には、MASSINSERT オプションを使用します。これはプロセッサー・オーバーヘッドを最小化することによってパフォーマンスを改良し、したがって応答時間が改良されます。ESDS および KSDS の場合には、MASSINSERT を使用してレコードを追加すると、CICS は順次 VSAM 処理を使用します。これにより、分割が必要な時に、VSAM がレコードを制御インターバルに入れる方法が変更され、結果的に、関連する CI の中で分割が少なくなり、未使用スペースが減少します。
- キーが、相互に比較的接近しているが必ずしも隣接していない多くのレコードを順々に読み取っている場合には、**スキップ順次**処理を使用します (スキップ順次処理はブラウズの開始 (STARTBR コマンド) で始まります)。各レコードは READNEXT コマンドによって検索されますが、RIDFLD によって示されるキー・フィードバック域は、READNEXT コマンドを発行する前に、次に要求されたレコードのキーとともに提供されます。
- キーが共通の文字ストリングで始まるレコードのグループを削除する場合には、DELETE コマンドに GENERIC オプションを使用します。CICS は総称 DELETE を内部的に最適化します。

BDAM データ・セット

このタスクについて

CICS は、いくつかの単一スレッド処理を実行し、オペレーティング・システム待機を発行して、BDAM データ・セット要求を処理する必要があるので、BDAM データ・セットの効率は VSAM より低くなります。したがって、可能な場合には BDAM データ・セットの代わりに、相対レコード VSAM データ・セット、または相対バイト・アドレス (RBA) によってアドレッシングされる入力順データ・セットを使用する必要があります。

BDAM データ・セットを更新モードで使用している場合には、選択したデータ・セットの保全性によってパフォーマンスが著しく影響を受けることに注意してください。

BDAM データ・セットのファイル管理テーブルの SERVREQ オペランドに**排他制御**を指定した場合には、CICS は並行更新を避けるようオペレーティング・システムに要求します。しかし、これはオーバーヘッドに重大な影響を与えます。

効果的なブラウズ (非 RLS モード)

しばしば、データ・セット・ブラウズが、多数の出力画面を生成するトランザクションの出力のソースになり、これがシステム・リソースを独占することがあります。長いブラウズは、BMS、タスク制御、および端末に必要なオーバーヘッドに加えて、他のトランザクションをロックアウトし、全部の応答時間を増やすといった、システムに対する重大な負荷をもたらすことがあります。これが起きる理由は、CICS 制御の考え方が次のような想定に基づいているためです。すなわち、端末オペレーターは少数のデータ・レコードにアクセスするトランザクションを開始し、情報を処理し、結果を受け取るという想定です。このプロセスには、CICS がマルチタスキングを行うことができる多数の待機を伴います。しかし、CICS は割り込み駆動マルチタスキング・システムではないので、処理の割には I/O の量が少ないタスクが優先順位とは無関係にシステムを独占することがあります。制御インターバルの中に多くのレコードを持つデータ・セットのブラウズがこのようなトランザクションです。

この点は、他のタスクが制御権を獲得できるように、DELAY コマンドまたは SUSPEND コマンドを定期的に発行することによって防ぐことができます。ブラウズがページ出力を作成する場合には、トランザクションを、888 ページの『ページの作成およびルーティングの操作』で説明されている方法の 1 つで分割することを検討する必要があります。

第 8 部 端末管理

CICS アプリケーション・プログラミング・インターフェースには、以下の 2 組のコマンドが端末との通信用として含まれています。

1. 端末制御コマンド
2. 基本マッピング・サポート (BMS)

端末管理インターフェース

端末管理は 2 つの中でも基本的なものです。これは、柔軟性と機能を提供しますが、プログラミングの労力はより大きくなります。特に、端末管理レベルのコーディングを行う場合には、ユーザー・アプリケーションで装置データ・ストリームを構築する必要があります。

端末制御コマンドは、各種の装置に適用され、プログラムがサポートする端末、および端末を制御するアクセス方式に対する、プログラムの依存度を少なくします。コマンドそのものに加えて、CICS は、端末または論理装置との間の読み取りまたは書き込みに必要な、データ変換、入出力操作の同期化、およびセッション制御も提供します。これは、複雑で個々に異なる、個別のコミュニケーション・アクセス方式の API から解放されることを援助します。

BMS

BMS によって、はるかに高水準の言語レベルで端末と通信することができます。BMS はユーザー・データの形式設定を行うため、ユーザーはデータ・ストリームの詳細を知っている必要はありません。したがって、特に、ユーザー・アプリケーションが新しいタイプの端末をサポートする必要がある場合には、最初のコーディングが容易で、保守が容易です。しかし、BMS パス長は長く (BMS そのものが端末管理を使用する)、BMS は端末管理がサポートするすべての端末タイプをサポートするわけではありません。BMS についての説明は、775 ページの『第 60 章 基本マッピング・サポート』にあります。

BMS は、端末管理以上に、特定の装置の特性およびコミュニケーションのメカニズムからプログラムを切り離しますが、柔軟性と機能をいくらか犠牲にします。

このセクションでは、以下について説明します。

- 589 ページの『第 33 章 端末アクセス方式のサポート』
- 591 ページの『第 34 章 端末制御コマンド』
- 595 ページの『第 35 章 データ伝送コマンドの使用』
- 597 ページの『第 36 章 装置制御コマンド』
- 599 ページの『第 37 章 端末装置のサポート』
- 603 ページの『第 38 章 ご使用の端末情報の検出』
- 607 ページの『第 39 章 VTAM の使用』
- 611 ページの『第 40 章 順次端末サポートの使用』
- 615 ページの『第 41 章 TCAM の使用』
- 617 ページの『第 42 章 バッチ・データ交換の使用』

© Copyright IBM Corp. 1989, 2009

第 33 章 端末アクセス方式のサポート

CICS Transaction Server for z/OS バージョン 4 リリース 1 は、次のアクセス方式へのインターフェースを通じて直接端末をサポートします。

- **仮想記憶通信アクセス方式 (VTAM)**
- **基本グラフィックス・アクセス方式 (BGAM)** (GDDM® を使用しているグラフィックス端末の場合)
- **順次アクセス方式 (SAM)** (順次装置によってシミュレートされる端末の場合)

CICS はオペレーティング・システム・コンソールも端末としてサポートしますが、アクセス方式を通じてではなくシステム・サービスを通じてサポートします。コンソールへの端末管理インターフェースは (一定のコンソールには一定の制約事項がある場合があるにしても) 他の端末へのインターフェースと同じですが、BMS は利用不能です。CICS がサポートする端末の完全なリストは、「*CICS Resource Definition Guide*」のに記載されています。

CICS の以前のリリースでも、TCAM の「DCB」インターフェースを使用して各端末をサポートしていました。CICS では引き続き、TCAM/DCB を使用した端末からのトランザクションを実行できます。ただし、その端末自身は、リモートの CICS TS 3.1 以前の端末専有領域に接続する必要があります。TCAM/DCB を使用したローカル端末はサポートされていません。また、TCAM の ACB インターフェースはサポートされていません。

CICS のもとで実行するトランザクションは、リモート TCAM 端末用のローカル・サロゲートと通信し、2 つの CICS システムは、サロゲートと実際の端末との間の対応を管理します。トランザクションは、端末を所有している CICS がトランザクションを CICS 領域にルーティングしたとき、または CICS 領域内で自動トランザクション開始 (ATI) によって、呼び出されます。ATI とともに、この領域は、トランザクションのプリンシパル装置としての端末を、その端末を所有している CICS 領域を通して割り当てます。

© Copyright IBM Corp. 1989, 2009

第 34 章 端末制御コマンド

このセクションで説明されている各コマンドは、それを実行するタスクのプリンシパル装置に対してのみ適用されます。その機能は以下のいずれかです。

- SAM を使用して接続された装置
- VTAM を介して接続される LU タイプ 0、1、2、3、または 4。

注: このセクションでは、その代替装置またはプリンシパル装置のいずれに送信される場合についても、プログラム間通信に関しては説明されていません。この点については、別の解説書に説明があり、APPC コマンドに関しては「*CICS Distributed Transaction Programming Guide*」に説明されています。

端末制御コマンドは以下の 4 つのグループに分類されます。

- 基本データ伝送コマンド: RECEIVE、SEND、および CONVERSE。
- 装置制御、同期化伝送、セッション終了、または類似した制御機能の実行を行うコマンド。
- 端末について通知するコマンド: ASSIGN および INQUIRE。
- 特殊装置グループ・コマンド: バッチ・データ交換 (BDI) コマンド。

送信/受信モード

このセクションで説明される端末および論理装置は、すべて「半二重フリップフロップ」モードで作動します。これは、基本的に、任意のある時点で、会話における一方のパートナーが送信モード (データまたは制御コマンドを送信することができる) になっていて、他方が受信モード (受信に制限されている) になっているということです。このプロトコルは VTAM のもとで形式的に定義され、実施されます。CICS は、他のアクセス方式の下で接続された端末に関する同じ規則に従いますが、すべての操作が同じにならないようにするため、ハードウェアとアクセス方式はいずれも異なる作業を実行します。

端末がタスクのプリンシパル装置である場合には、その会話パートナーはそのタスクです。端末がタスクと関連していない場合には、その会話パートナーは CICS の端末管理コンポーネントです。VTAM のもとで、タスク間の会話は、いずれか一方のパートナーが先に送信することができる中立状態のままになっています。普通は、端末が先に、タスクを開始する非送信請求入力の送信を行います (244 ページの『タスクの開始方法』を参照してください)。

また、この伝送は送信 / 受信の役割を逆転します。その後は、端末は受信モードになり、接続されたタスクによって表される CICS は送信モードになります。タスクが開始され、たとえどんなに多くの SEND が実行されたとしても、明示的に会話の方向が変更されるまでは、そのタスクは送信モードのままになっています。タスクを受信モードにすることができる 1 つの方法は、SEND コマンドに INVITE オプションを指定することです。INVITE を指定した SEND の後で、タスクは受信モードになり、再び端末に送信する前に RECEIVE を実行しなければなりません。ま

© Copyright IBM Corp. 1989, 2009

た、INVITE を先行させないで RECEIVE を実行してタスクを受信モードにすることもできます。INVITE は伝送を最適化します。

呼び出された入力メッセージはずっと以前に伝送された (これがタスクを開始した) ために、非送信請求入力によって開始されたタスクの最初の RECEIVE コマンドは送信 / 受信モードの点で重要ではありません。この RECEIVE はメッセージをタスクに対してアクセス可能にするだけで、関連 EIB フィールドを設定します。

ATI タスク (CICS によって自動的に開始されたタスク) も、非送信請求入力で開始されたタスクと同様に、送信モードで開始します。

VTAM / ネットワーク・エラーの発生時にタスクが正常に実行中で非端末操作を行っている場合、タスクはエラーを認識せず、次の端末管理要求を出そうとするまで続けて処理をしているに注意してください。タスクが TERMERR を受信するのは、この時点です。タスクが次の端末管理要求を出さない場合は、TERMERR または ABEND を受信しません。

端末の競合

プリンシパル装置としての端末が使用可能になるとすぐに、CICS は自動タスク開始 (ATI) の要求を満たします。タスクが端末で終了し、CICS がその端末に対する ATI 要求を出した場合、CICS 間で競合が起こります。つまり、端末は ATI タスクの開始を要求し、端末ユーザーは非送信請求入力によって特定のタスクを開始しようとします。この状態では、CICS は常に自分自身を競合*敗者* としてセットアップします。すなわち、端末が直前のトランザクションが終了した後、十分に速やかに非送信請求入力を送信した場合には、CICS はそれを処理するタスクを作成し、ATI 要求の実行を遅らせます。これは意図的なもので、競合状態ではユーザーに優先権が与えられます。

RETURN IMMEDIATE

しかし、ユーザーの介入を許さずに、端末から一連の特定タスクを連続して実行することが必要になる場合があります。CICS は、タスクを終了する RETURN コマンドで IMMEDIATE を使用して、これを行う方法を提供します。RETURN IMMEDIATE によって、CICS はタスクを開始して、その端末でタスクに関する要求を待っている他のいずれかを指名する前に、端末からの入力を受け入れることをしないで、TRANSID オプションに名前が指定されているトランザクションを即時に実行します。旧タスクは、新規タスクにデータを渡すことができます。新規タスクは RECEIVE を使用して、ユーザーが非送信請求入力によってタスクを開始したかのように、このデータにアクセスしますが、入出力は起こりません。この RECEIVE は、非送信請求入力によって開始されたタスクの最初のコマンドと同様に、送信 / 受信状況には何の効果もありません。渡したデータを新規タスクに対して使用可能にするだけのことです。端末がブラケット・プロトコル (610 ページの『割り込みの防止 (ブラケット・プロトコル)』に説明があります) を使用している場合には、CICS は、通常そうするように、最初のタスクの終了時にブラケットを終了しませんが、その代わりに、ブラケットを続行して、次のタスクを組み込みます。したがって、タスク間でのブラケットの終了時点でキーボードの自動オープンは行われません。

順序のない会話

いつ「送話」(キー入力および伝送) し、いつ「受話」(出力待ち) すべきかについて、ユーザーは通常迷うことはありません。なぜならアプリケーションがこれを明確にしているためです。3270 ディスプレイおよびその他の多くの端末では、この規則を強化するために、ユーザーが伝送した後でキーボードがロックされます。キーボードは、タスクが RECEIVE の前の SEND でアンロックするか、あるいはタスクの最終 SEND でアンロックするまでロックされたままになっています。これは、プロトコルを中断するためには、ユーザーが特定の何かを行う (キーボードのリセット・キーを押す) 必要があるという意味です。

ユーザーがこれを行った場合に何が起こるでしょうか? VTAM のもとで接続された端末の場合には、先読みキューイングが強要されていない限り、このプロトコルに違反すると、タスクは異常終了 (コード ATCV) します。先読みキューイングによって、論理装置およびタスクはいつでも送受信することができます。CICS は入力メッセージをタスクが必要とするまで一時記憶域に保管します。タスク終了によって読み取られない入力は廃棄されます。先読みキューイングはトランザクション・レベルに適用されます (これはトランザクションの実行時の PROFILE の RAQ オプションに指定します)。先読みキューは LU タイプ 4 装置に対してのみ適用され (元は互換性のために提供されていました)、トランザクションにおいて同じ方法で BTAM 接続および VTAM 接続の両方の端末をサポートできるようにします。BTAM はサポート対象ではなくなっているため、先読みキューは使用しないでください。

順次端末は、送信/受信の規則において他の端末と異なっています。入力は事前準備されたファイルであるので、タスクが入力を要求した場合には常に CICS は入力メッセージを提供するだけで、プロトコルを中断することはできません。入力が誤って準備されたか、あるいはタスクがその処理のためにプログラミングされたものでない場合には、そのタスクが入力との同期を外すか、入力を早目に読み飛ばしてしまうか、あるいは入力の一部の読み取りを失敗させることがあります。

割り込み

VTAM は、受信モードになっている端末が送信することをそのパートナーに通知するメカニズムを提供しています。これは、VTAM における「シグナル」データ・フローであり、タスクからの次の SEND、RECEIVE、または ISSUE DISCONNECT コマンドで検出されます。シグナル・フローが起こった場合には、CICS は SIGNAL 条件を出し、EIB に EIBSIG を設定します。CICS の SIGNAL 条件に関するデフォルトのアクションはそれを無視することです。何らかの効果を持つシグナルの場合には、タスクが最初にそのシグナルを検出してから、会話の方向を変更してそれを取り扱う必要があります。

3270 ディスプレイ端末およびその他の一部では、ATTENTION キーは割り込みを生成するキーです。必ずしもすべての端末にこの機能が備わっているわけではありませんが、VTAM では、バインド・イメージはその機能のサポートを表す必要があります。そうしないと、VTAM は割り込みを無視します。

端末待ち

タスクが WAIT を指定しないで SEND コマンドを実行した場合には、CICS は出力の伝送を据え置いて、端末処理全体またはユーザー・タスクの伝送のいずれか一方を最適化することがあります。これを行う場合には、CICS は出力メッセージを保管し、ユーザー・タスクが実行を続行できるように、それをディスパッチ可能にします。出力の伝送も行う ISSUE COPY コマンドおよび ISSUE ERASE コマンドは WAIT なしで同様に動作します。

WAIT オプションを使用した場合には、CICS は出力操作が完了するまでユーザー・タスクに制御を返しません。この待機は、応答時間およびメモリー占有についての付随効果によって、タスクの経過時間を長引かせますが、続行前に SEND にエラーがあったかどうかが、ユーザー・タスクに確実にわかります。ユーザーは、この待機の一部を回避したり、SEND の後に実行すべき処理がある場合には、操作の完了を検査することができます。WAIT を指定しないで SEND を実行し、処理を続行してから、SEND の結果を知る必要がある地点で WAIT TERMINAL コマンドを実行します。

入力の伝送を必要とする RECEIVE コマンドを実行した場合には、伝送は RECEIVE コマンドが完了する前に行われる必要があるために、ユーザー・タスクは常に待機します。しかし、RECEIVE が端末の入出力と対応していない場合があります。非送信請求端末によって開始されるタスクの最初の RECEIVE が、これの最も頻繁にある例ですが、次のセクションで説明するように、この例は他にもあります。

また、端末に影響を与える何らかのコマンドを実行した場合にも、CICS は、新規コマンドを処理する前に、直前のコマンドが完了したかどうか (これには据え置かれたすべての伝送も含まれます) を確認します。

第 35 章 データ伝送コマンドの使用

タスクのプリンシパル装置である端末または論理装置との間でデータの伝送を行うコマンドには以下の 3 つがあります。

RECEIVE
端末からデータを読み取ります。

SEND 端末にデータを書き込みます。

CONVERSE
端末にデータを書き込み、入力を待機し、さらに入力を読み取ります。

CONVERSE は、基本的に、SEND と RECEIVE の組み合わせで、通常、SEND の後に RECEIVE が続いているものと同等です。ある場合には、SEND および RECEIVE の代わりに CONVERSE を使用しなければなりません。例えば、一定の 3270 装置への構造化フィールドの送信の場合などです。その他の場合には、CONVERSE は提供されていないので、SEND および RECEIVE を使用しなければなりません。これらは 601 ページの表 31 にあります。

SEND、RECEIVE、および CONVERSE の各コマンドの詳細ついては、 「*CICS アプリケーション・プログラミング・リファレンス*」に説明されています。装置およびアクセス方式が異なるとオプションもかなり異なるので、これらは装置グループによって分類されます。 599 ページの『第 37 章 端末装置のサポート』 には、特定の装置に対してどの装置グループを使用するかが示されています。

RECEIVE における受信内容

ここでは、「入力メッセージ」および「伝送」という用語を使用し、端末が送信した内容とアプリケーションが受信した内容の両方を示します。端末の最も一般的なタイプの場合には、これらは同等です。例えば、3270 ディスプレイはバッファー内で変更されたものはなんでも単一エンティティーとして送信し、通常、端末と関連したタスクが、単一の RECEIVE コマンドへの応答の中のメッセージ全体を入手します。

しかし、入力メッセージと物理伝送が常に同等なわけではなく、RECEIVE コマンドへのいずれか一方の 1 対 1 の対応関係が左右されることがある要因がいくつかあります。そのオプションは、以下のとおりです。

- VTAM チェーニング
- 論理レコード
- NOTRUNCATE オプション
- 「印刷」PA キー

入力のチェーニング

一部の SNA 装置は長い入力メッセージを複数の物理的伝送に分割し、このプロセスは「チェーニング」と呼ばれます。 CICS は、タスクに関連する端末の定義方法に応じて、コンポーネント伝送をアセンブルして 1 つの入力メッセージにするか、

© Copyright IBM Corp. 1989, 2009

個別に表示するかを決定します。これにより、チェーニングした入力メッセージを読み取るために必要な RECEIVE の数が左右されます。インバウンド・チェーニングの詳細については、 607 ページの『入力データのチェーニング』に説明があります。

論理メッセージ

一部の装置が長い入力を複数の伝送に分割するのとまったく同様に、他の装置は短い入力をブロック化して、単一の伝送でそれを送信します。ここで再び、CICS は、CICS または受信プログラムのどちらが非ブロック化するかについてのオプションを提供します。この選択項目は、単一の RECEIVE で入手するデータの量にも影響します (この件について詳しくは、 608 ページの『論理レコードの取り扱い』を参照してください)。

NOTRUNCATE オプション

さらに、RECEIVE 当たりに 1 個の入力メッセージという規則に対する別の例外が起こるのは、入力データの長さがプログラムが想定した長さより長い場合です。これが起こり、RECEIVE コマンドに NOTRUNCATE が指定されている場合には、CICS は超過データを保管し、端末への対応する読み取りのない、プログラムからの後続の RECEIVE コマンドを満足させるためにそのデータを使用します。NOTRUNCATE を使用している場合には、EIB の EIBCOMPL がオンに設定される (すなわち、X'FF' に設定される) まで RECEIVE を実行する必要があります。CICS が EIBCOMPL をオンにするのは、使用可能なメッセージがもうない場合です。

NOTRUNCATE を使用しないと、CICS は超過データを廃棄し、EIBCOMPL をオンにし、さらに LENGERR 条件を立ち上げます。CICS は LENGTH オプションに指定されているデータ域で (ユーザーが用意した場合)、切り捨てる前にデータの真の長さを報告します。

印刷キー

CICS システムに「印刷」として定義されている PA キーがある場合は、通常の送信/受信シーケンスに対する別の例外が発生することがあります。タスクが RECEIVE を発行し、ユーザーが「印刷」キーを押して応答した場合には、CICS はこの入力を代行受信し、要求の実現に必要な処理を実行し、端末を再び受信モードにします。ユーザーは別の入力を送信して、元の RECEIVE を満足させる必要があります (「印刷」キーについて詳しくは、 758 ページの『CICS 印刷キー』を参照してください)。

第 36 章 装置制御コマンド

データ伝送コマンドに加えて、端末用の CICS API には、データではなく指示または制御情報を、端末またはそれを制御するアクセス方式に送信する一連のコマンドが含まれます。これらのコマンドおよびその簡単な説明が以下の表にリストされています。これらのコマンドのすべてがすべての端末に適用されるわけではなく、一部については、異なる形式が異なる端末に適用されます。 599 ページの『第 37 章 端末装置のサポート』を参照してください。

次の表の端末は、他の方法で明示的に指定された場合を除き、常にコマンドを実行しているタスクのプリンシパル装置です。これは、通常は端末と見なされないタイプの論理装置とすることができます。

表 29. 端末および論理装置用の制御コマンド

コマンド	意味
FREE	現行のタスクが終了する前に別のタスクが端末を使用することができるように、その端末をタスクから解放します。
ISSUE COPY	TERMID オプションに名前が指定されている端末のバッファー内容を、タスクが所有している端末のバッファーにコピーします。端末は両方とも 3270 でなければなりません。
ISSUE DISCONNECT	CICS と端末の間のセッションの終了がタスクの終了時点になるようにスケジュールします。
ISSUE EODS	データ・セットの終わり機能管理ヘッダーを送信します (3650 インタープリター論理装置専用)。
ISSUE ERASEAUP	端末の無保護フィールドをすべて消去します (3270 装置専用)。
ISSUE LOAD	PROGRAM オプションに名前が指定されているプログラムをロードするように端末に指示します (3650 インタープリター論理装置専用)。
ISSUE PASS	CICS からの端末の切断および LUNAME オプションに指定された VTAM アプリケーションへの制御権移動が、発行タスクの終了時点になるようにスケジュールします。
ISSUE PRINT	端末バッファーを、印刷要求に最初に適格になったプリンターにコピーします (3270 ディスプレイ専用)。
WAIT SIGNAL	端末が SIGNAL データ・フロー・コマンドを送信するまで、タスクの発行を延期します。
WAIT TERMINAL	直前の端末操作が完了するまで、タスクの発行を延期します。

© Copyright IBM Corp. 1989, 2009

第 37 章 端末装置のサポート

ハードウェアとアクセス方式の重要度は、端末と通信するために BMS を使用する点と端末制御コマンドを使用する点の間の主な相違の 1 つです。端末管理はすべての機能を提供するのに対して、BMS は、機能の一部の損失と引き換えに、アプリケーションがハードウェアに依存することから保護します。

全機能が提供される結果は、すべての端末制御コマンドがすべての装置に適用されるわけではないということです。一部のコマンドでは、適用されるオプションと起こる可能性がある例外条件を判別するために、端末のタイプがわかっている必要があります。コマンドによっては、どのアクセス方式が使用されているかを知ることも必要になります。次の 2 つの表は、どのコマンドがどの端末とアクセス方式の組み合わせに適用されるかを示しています。複数タイプの端末をサポートする必要がある場合には、603 ページの『第 38 章 ご使用の端末情報の検出』に説明されているコマンドを使用して、ユーザー・タスクがそのプリンシパル装置として持っているタイプがどれかを判別することができます。

表を使用するためには、ユーザー・プログラムがサポートする必要がある端末タイプを表 30 の 1 列目から探し出してください。 2 列目の値を使用し、601 ページの表 31 の 1 列目で対応するコマンド群を見つけます。この表の 2 列目は、アクセス方式を示し、3 列目は、使用することができるコマンドを示しています。コマンド自体の完全な説明は、「*CICS アプリケーション・プログラミング・リファレンス*」のに記載されています。そのマニュアルに複数のバージョンのコマンドがある場合には、どれを使用するかがこの表に指示されています。この情報は、マニュアルそのものに示されるのとまったく同様に、コマンドの後の括弧の中に示されます。

表 30. CICS がサポートする装置

装置	コマンドの使用対象
2260、2265	2260
3101 (TWX 33/35 としてサポート)	3767
3230 (VTAM)	3767
3270 ディスプレイ、3270 プリンター (VTAM SNA)	LU タイプ 2/3
3270 ディスプレイ、3270 プリンター (VTAM 非 SNA)	3270 論理
3270 ディスプレイ、3270 プリンター (非 VTAM)	3270 ディスプレイ
SCS プリンター (VTAM)	SCS
3600 パイプライン・モード (VTAM)	3600 パイプライン
3601 (VTAM)	3600-3601
3614 (VTAM)	3600-3614
3600 として接続された 3630 (3631、3632、3633、3643、3604)	3600 入力を使用
3641、3644、3646、3647 (VTAM、3767 として接続)	3767
3643 (VTAM、LU タイプ 2 として接続)	LU タイプ 2/3
3642、3645 (VTAM、SCS プリンターとして接続)	SCS
3650 インタープリター LU	3650 インタープリター

© Copyright IBM Corp. 1989, 2009

表 30. CICS がサポートする装置 (続き)

装置	コマンドの使用対象
3650 ホスト会話型 LU (3270)	3650-3270
3650 ホスト会話型 LU (3653)	3650-3653
3650 ホスト・コマンド LU (3680、3684)	3650-3680
3650 インタープリター LU	3650 インタープリター
3650 ホスト会話型 LU (3270)	3650-3270
3650 ホスト会話型 LU (3653)	3650-3653
3650 ホスト・コマンド LU (3680、3684)	3650-3680
3730	3790 全機能または照会
3767 対話式 LU (VTAM)	3767
3770 対話式 LU (VTAM)	3767
3770 全機能 LU	3790 全機能または照会
3770 バッチ LU (3771、3773、3774) (VTAM)	3770
3790 全機能または照会	3790 全機能または照会
3790 3270 ディスプレイ LU	3790 3270 ディスプレイ
3790 SCS プリンター	3790 SCS
3790 3270 プリンター	3790 3270 プリンター
4700 (3600 としてサポート)	3600 入力を使用
5280 (3270 として接続)	3270 入力を使用
5520 VTAM (3790 全機能 LU としてサポート)	3790 全機能または照会
5550 (3270 としてサポート)	3270 入力を使用
5937 (3270 としてサポート)	3270 入力を使用
6670 VTAM	LU タイプ 4
DPCX のもとでの 8130、8140 (3790 としてサポート)	3790 全機能または照会
ホスト表示サービスまたはホスト・トランザクション機能を使用する 8100 DPPX/BASE (3790 として接続)	3790 全機能または照会
8775 接続を含む 8100 DPPX/DSC、DPCX/DSC (3270 としてサポート)	LU タイプ 2/3
8775	LU タイプ 2/3
8815	APPC
ディスプレイライター (3270 としてサポート)	3270 入力を使用
ディスプレイライター (APPC としてサポート)	APPC
INTLU (対話式 LU)	3767
PC、PS/2 (3270 として接続)	3270 入力を使用
Scanmaster	APPC
Series/1 (3650 パイプラインとしてサポート)	3600 パイプライン
Series/1 (3790 全機能 LU としてサポート)	3790 全機能または照会
System/32 (5320) VTAM (3770 としてサポート)	3770 入力を使用
System/34 (5340) VTAM (3770 としてサポート)	3770 入力を使用
System/34 (5340) 非 VTAM	システム/3
System/36™ (System/34 としてサポート)	システム/34 入力を使用

表 30. CICS がサポートする装置 (続き)

装置	コマンドの使用対象
System/38™ (5381) VTAM (3770 として接続)	3770 入力を使用
System/38 (5381) VTAM (APPC として接続)	APPC
TWX 33/35 VTAM NTO	3767
WTTY VTAM NTO	3767

表 31. 装置タイプ別の端末制御コマンド

装置グループ名	アクセス方式	適当なコマンド
2260	非 VTAM	RECEIVE (2260)、SEND (2260)、CONVERSE (2260)、ISSUE DISCONNECT (デフォルト)、ISSUE RESET
3270 ディスプレイ	非 VTAM	RECEIVE (3270 ディスプレイ)、SEND (3270 ディスプレイ)、CONVERSE (3270 ディスプレイ)、ISSUE COPY (3270 ディスプレイ)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PRINT、ISSUE RESET
LU タイプ 2/3 (3270 SNA)	VTAM	RECEIVE (LU タイプ 2/3)、SEND (LU タイプ 2/3)、CONVERSE (LU タイプ 2/3)、ISSUE COPY (3270 論理)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PASS、ISSUE PRINT
3270 論理 (3270 非 SNA)	VTAM	RECEIVE (3270 論理)、SEND (3270 論理)、CONVERSE (3270 論理)、ISSUE COPY (3270 論理)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PASS、ISSUE PRINT
SCS	VTAM	SEND (SCS)、CONVERSE (SCS)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS
3600 パイプライン	VTAM	RECEIVE (3600 パイプライン)、SEND (3600 パイプライン)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS
3600-3601	VTAM	RECEIVE (3600-3601)、SEND (3600-3601)、CONVERSE (3600-3601)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS、WAIT SIGNAL
3600-3614	VTAM	RECEIVE (3600-3614)、SEND (3600-3614)、CONVERSE (3600-3614)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS
3650 インタープリター	VTAM	RECEIVE (3650)、SEND (3650 インタープリター)、CONVERSE (3650 インタープリター)、ISSUE DISCONNECT (デフォルト)、ISSUE EODS、ISSUE LOAD、ISSUE PASS
3650-3270	VTAM	RECEIVE (3650)、SEND (3650-3270)、CONVERSE (3650-3270)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PASS、ISSUE PRINT
3650-3653	VTAM	RECEIVE (3650)、SEND (3650-3653)、CONVERSE (3650-3653)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS

表 31. 装置タイプ別の端末制御コマンド (続き)

装置グループ名	アクセス方式	適当なコマンド
3650-3680	VTAM	RECEIVE (3650)、RECEIVE (3790 全機能または照会)、SEND (3650-3680)、SEND (3790 全機能または照会)、CONVERSE(3650-3680)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS
3767	VTAM	RECEIVE (3767)、SEND (3767)、CONVERSE (3767)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS、WAIT SIGNAL
3770	VTAM	RECEIVE (3770)、SEND (3770)、CONVERSE (3770)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS、WAIT SIGNAL
3790 全機能または照会	VTAM	RECEIVE (3790 全機能または照会)、SEND (3790 全機能または照会)、CONVERSE (3790 全機能または照会)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS、WAIT SIGNAL
3790 3270 ディスプレイ	VTAM	RECEIVE (3790 3270 ディスプレイ)、SEND (3790 3270 ディスプレイ)、CONVERSE (3790 3270 ディスプレイ)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PASS、ISSUE PRINT
3790 3270 プリンター	VTAM	SEND (3790 3270 プリンター)、ISSUE DISCONNECT (デフォルト)、ISSUE ERASEAUP、ISSUE PASS
3790 SCS	VTAM	SEND (3790 SCS)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS
LU タイプ 4	VTAM	RECEIVE (LU タイプ 4)、SEND (LU タイプ 4)、CONVERSE (LU タイプ 4)、ISSUE DISCONNECT (デフォルト)、ISSUE PASS、WAIT SIGNAL
外部コントローラー (バッチ・データ交換)	VTAM	ISSUE ABORT、ISSUE ADD、ISSUE END、ISSUE ERASE、ISSUE NOTE、ISSUE QUERY、ISSUE RECEIVE、ISSUE REPLACE、ISSUE SEND、ISSUE WAIT
他のすべて	VTAM	RECEIVE (VTAM デフォルト)、SEND (VTAM デフォルト)、CONVERSE (VTAM デフォルト)、ISSUE PASS
他のすべて	非 VTAM	RECEIVE (非 VTAM デフォルト)、SEND (非 VTAM デフォルト)、CONVERSE (非 VTAM デフォルト)

第 38 章 ご使用の端末情報の検出

一部のアプリケーションは複数タイプの端末をサポートする必要があり、このタイプが別のコードを必要とすることで十分な違いがある場合があります。このようなプログラムを作成しており、現在通信している端末の種類が何かを判別する必要がある場合は、ASSIGN コマンドを使用してそれを判別することができます。

ASSIGN は、そのプリンシパル装置を説明する多数のフィールドを含め、実行中のタスクについてのさまざまな情報を返します。表 32 には、端末管理の操作に直接関連する情報がリストされています。ASSIGN オプションには、BMS およびタスクのその他の局面と関連する他のオプションがあります。すべての ASSIGN オプションの詳細については、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。表の 2 列目で引用されている「端末」は常にタスクのプリンシパル装置です。

表 32. 端末に関する ASSIGN コマンド・オプション

ASSIGN オプション	**返される情報**
ALTSCRNHT ALTSCRNWD	端末画面の (端末定義からの) 代替高さおよび幅。SCRNHT および SCRNWD も参照してください。
APLKYBD	端末が APL 型キーボードを持っているかどうか
APLTEXT	端末が APL テキスト機能を持っているかどうか
BTRANS	端末が背景透明機能を持っているかどうか
COLOR	端末が拡張カラー機能を持っているかどうか
DEFSCRNHT DEFSCRNWD	端末画面の (端末定義からの) デフォルト高さおよび幅。SCRNHT および SCRNWD も参照してください。
DELIMITER	端末のデータ・リンク制御文字 (3600 端末専用)
DESTID DESTIDLENGTH	外部宛先の ID およびその長さ (BDI 操作専用)
DSSCS	端末が SCS データ・ストリーム装置であるかどうか
DS3270	端末が 3270 データ・ストリーム装置であるかどうか
EXTDS	端末が「照会構造化フィールド」オーダーをサポートするかどうか
EWASUPP	端末が「消去書き込み代替」オーダーをサポートする (すなわち、代替画面サイズ機能を持っている) かどうか
FACILITY	端末の 4 文字の ID
FCI	タスクと関連したプリンシパル装置のタイプ (端末、キュー、など)
GCHARS GCODES	端末と関連した図形文字セット・グローバル ID およびコード・ページ・グローバル ID
HILIGHT	端末が拡張強調表示機能を持っているかどうか
KATAKANA	端末がカタカナをサポートするかどうか
LANGINUSE	3 文字の略号
MSRCONTROL	端末が磁気スロット読み取り装置をサポートするかどうか

© Copyright IBM Corp. 1989, 2009

表 32. 端末に関する ASSIGN コマンド・オプション (続き)

ASSIGN オプション	返される情報
NATLANGINUSE	現行タスクに使用中の各国語
NETNAME	VTAM ネットワーク内の端末の 8 文字の ID
NUMTAB	通帳の正確な位置に印刷エレメントを位置決めするために必要なタブの数 (2980 専用)
OPID OPCLASS	端末にサインオンしているユーザーのオペレーター ID およびオペレーター・クラス
OUTLINE	端末がフィールド枠取り機能を持っているかどうか
PARTNS	端末が画面分割をサポートするかどうか
PS	端末がプログラム式シンボル機能を持っているかどうか
SCRNHT SCRNWD	現行タスクの端末画面の高さと幅
SIGDATA	端末から受信した SIGNAL データ
SOSI	端末が混合 EBCDIC/2 バイト文字セット機能を持っているかどうか
STATIONID TELLERID	端末のステーションおよびテラー (2980 専用)
TERMCODE	端末のタイプおよび型式番号
TERMPRIORITY	端末優先順位値
TEXTKYBD	端末が TEXTKYBD 機能を持っているかどうか
TEXTPRINT	端末が TEXTPRINT 機能を持っているかどうか
UNATTEND	端末が未接続かどうか
USERID USERNAME USERPRIORITY	端末でサインオンしているユーザーの 8 バイトの ID、20 文字の名前、および優先順位
VALIDATION	端末が妥当性検査機能を持っているかどうか

また、INQUIRE TERMINAL コマンドを使用して、ユーザー所有の端末またはその他のすべての端末を判別することもできます。ASSIGN は現行タスクでのその端末の使用を説明するのに対して、INQUIRE TERMINAL は端末定義から情報を返します。しかし、多くのオプション、特にハードウェア特性の場合に、返される情報が同じです。INQUIRE TERMINAL については、「*CICS System Programming Reference*」のINQUIRE TERMINALに記載されています。

端末管理の操作における EIB フィードバック

CICS は、BMS によって生成された結果を含む、端末制御コマンドの処理の結果を、EIB に報告します。端末操作の複雑さのために、多くの EIB フィールドは端末コマンドに特有です。プリンシパル装置に適用される点については、605 ページの表 33 にリストされています (その他のフィールドは、LU タイプ 6.1、APPC およ

び MRO の各操作にのみ関連します。これらのプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」のを参照してください)。

EIB フィールドが記入されるのは、CICS が制御をユーザー・タスクに戻す時点で、それらは常に適用される最後のコマンドを説明しています。これは、代替装置を介してプログラム間通信を管理しており、プリンシパル装置を使用している場合には、一方からの結果を他方からの結果で上書きする前に、EIB を検査する必要があるという意味です。

また、タスクが端末からの非送信請求入力によって開始されたか、あるいは同一端末で直前のタスクの RETURN IMMEDIATE によって開始された場合には、入力を記述する EIB フィールドはタスクの開始時には設定されません。RECEIVE を実行して入力に対するアクセス権を取得して、EIB フィールドを記入する必要があります。

注: EIB 値にしか関心がなく、データそのものに関心がない場合には、RECEIVE から INTO および SET の両方のオプションを省略してください。

プリンシパル装置に適用されるフィールドは次のとおりです。

表 33. 端末制御コマンドに適用される EIB フィールド

フィールド	内容
EIBAID	最終入力操作からのアテンション ID (AID) (3270 専用。 636 ページの『AID』を参照してください。)
EIBATT	入力に付加ヘッダー・データ (付加 FMH) が含まれているかどうか
EIBCOMPL	実行したばかりの RECEIVE コマンドがすべての入力データを使用したかどうか、あるいはさらに RECEIVE が必要かどうか (607 ページの『出力データのチェーニング』を参照)
EIBCPOSN	最終入力操作時点のカーソル位置 (3270 専用)
EIBEOC	チェーン終了標識が最終 RECEIVE からの入力に現れたかどうか
EIBFMH	受信したか、あるいは検索したばかりのユーザー・データに FMH が含まれているかどうか
EIBFREE	使用したばかりの機能を解放したかどうか
EIBRCODE、EIBRESP、EIBRESP2	直前に実行したコマンドからの CICS 応答コード値 **注:** 伝送が据え置かれることがある出力コマンドの場合には、これらの値は、偶発的な伝送ではなく、コマンドの初期 CICS 処理のみを反映します (594 ページの『端末待ち』を参照してください)。
EIBSIG	端末が SIGNAL を送信したかどうか
EIBTRMID	端末の (CICS) ID

第 39 章 VTAM の使用

VTAM の下では、論理装置との通信は、論理装置のタイプに応じて異なる規則 (プロトコル) によって制御されます。このセクションでは、アプリケーションがこのプロトコルに適合するようにし、またこれらのプロトコルを最大限利用するために、CICS によって提供されるオプションについて説明します。

入力データのチェーニング

以前に注意したように、一部の SNA 装置は長い入力メッセージを伝送のためにセグメント化します。各個別セグメントは**要求単位** (RU) と呼ばれ、論理メッセージ全体は**チェーン**と呼ばれます。CICS は、端末定義で、誰がチェーンをアセンブルするかを制御するオプション BUILDCHAIN を提供します。端末の BUILDCHAIN 値が YES の場合には、CICS がチェーンをアセンブルし、単一の RECEIVE コマンドへの応答で、メッセージ全体をプログラムに提供します。この選択は、全チェーンが完全で、アプリケーションに提供する前に使用可能になっていることを確実にします。

BUILDCHAIN=NO の場合には、アプリケーションがチェーンをアセンブルします。CICS は各 RECEIVE ごとに 1 つの RU を提供します。チェーン中の最終 RU を受信した時点で CICS が EOC (チェーン終了) 条件を立ち上げるので、アプリケーションはそれによって最終 RU の受信を認識します。チェーン中の RU が 1 つしかない場合、CICS がチェーンをアセンブルする場合、あるいは入力が 3270 ディスプレイのようにインバウンド・チェーニングをサポートしない端末からの場合にも CICS はこの条件を起こします。EOC 条件はエラーとは見なされません。これが起こった場合の CICS デフォルトのアクションはこの条件を無視することです。

EOC は、EODS (データ・セットの終わり) 条件または INBFMH (インバウンド FMH) 条件、あるいはその両方と同時に起こる場合があります。その条件と EOC がともにアクティブにある CONDITION コマンドの対象になっている場合、制御がどちらに渡されるかを判別するときに、いずれかの条件が EOC に優先します。

出力データのチェーニング

VTAM は、インバウンド端末データの他にアウトバウンドのチェーニングもサポートします。出力メッセージの長さがアウトバウンド RU サイズを超え、端末がアウトバウンド・チェーニングをサポートする場合には、CICS はメッセージを RU サイズのセグメントに分割し、それを個別に伝送します。

ユーザー・アプリケーションは、チェーニングは複数の SEND コマンドを介して単一出力メッセージをビット単位で CICS に渡すことができるという事実を利用できます。これを実行するには、各 SEND (メッセージが完了しているものを除く) に CNOTCOMPL (「チェーンが完全でない」) を指定します (メッセージ・セグメントの長さは、特定されている必要はありません。CICS が必要なだけの数の RU をア

© Copyright IBM Corp. 1989, 2009

センブルし伝送するためです)。トランザクションが実行中である PROFILE 定義はユーザーがこれをするために CHAINCONTROL=YES を指定しなければなりません。

注: 完全な論理メッセージ (すなわち、全チェーン) に適用されるオプションは、チェーンに関する最初の SEND コマンドにだけ指定する必要があります。これらには、FMH、LAST、および 3601 の場合の LDC が含まれます。

論理レコードの取り扱い

以前に説明したように、一部の装置は入力メッセージをブロック化し、複数の入力を単一の伝送で送信します。CICS によって、CICS またはアプリケーションのどちらが入力を非ブロック化するかを指定することができます。この選択は、現行トランザクションが実行中の PROFILE の LOGREC オプションで指定します。

LOGREC (NO) によって、CICS は RECEIVE への応答に入力メッセージ全体を提供します (入力がチェーニングしていないか、あるいは BUILDCHAIN=YES になっていないものとします)。入力を非ブロック化するのはユーザーの責任です。BUILDCHAIN=NO の場合、RECEIVE はチェーンの RU を 1 回に 1 つ検索します。一般に、単一の RU に 1 つ以上の完全な論理レコードが入るように、論理レコードは複数の RU をスパンしません。例外は、論理レコードがある RU で始まり、別の RU に継続することがある LU タイプ 4 装置です。このために、BUILDCHAIN=YES をお勧めするのは、このような装置の非ブロック化をユーザー独自に行う場合です。

PROFILE に LOGREC (YES) が指定されている場合には、CICS は (CICS が入力チェーンをアセンブルするかどうかにかかわらず) 各 RECEIVE コマンドへの応答で 1 つの論理レコードを提供します。

RU に複数の論理レコードが含まれている場合には、改行 (NL) 文字 X'15'、レコード間分離文字 (IRS 文字) X'1E'、または透過 (TRN) 文字 X'35' によってレコードが分離されます。NL 文字を使用した場合には、データをプログラムに渡す場合にこの文字は除去されず、論理レコードの終わりに現れます。しかし、IRS 文字を使用した場合には、この文字は除去されます。区切り文字が透過文字の場合には、論理レコードに NL および IRS (これらの文字は透過モードでは通常のデータと見なされます) を含む任意の文字を入れることができます。しかし、終端の TRN は除去されます。CICS は TRN で分離される論理レコードを 256 文字に制限します。

応答プロトコル

VTAM のもとで、CICS では、アウトバウンド・データに関する確定応答プロトコルまたは例外応答プロトコルのいずれか一方を使用することができます。

例外応答のもとでは、端末が SEND に肯定応答するのは、エラーが起こった場合だけです。ユーザー・タスクが例外応答を使用している場合には、CICS は、ユーザー・タスクを終了する前に、完了するためにタスク中の最終 SEND (唯一の SEND である場合もあります) を待機しません。したがって、エラーが起こった場合には、タスクに報告することができない場合があります。これが起こった場合には、このために作成された CICS 提供のタスクに報告されます。

確定応答では、端末が SEND のたびごとに肯定応答する必要があり、CICS が最終 SEND コマンドで応答を入手するまで、CICS はユーザー・タスクを終了しません。確定応答プロトコルを使用すると、いくらかパフォーマンスが低下しますが、このプロトコルは一部のアプリケーションで必要になる場合があります。

タスクを実行している PROFILE の MSGINTEG オプションはどちらの応答モードを使用するかを判別します。しかし、MSGINTEG (NO) (例外応答) を選択した場合には、DEFRESP オプションを使用して、任意の特定 SEND で確定応答をなおも要求することができます。この方法では、必要な場合だけパフォーマンスを犠牲にして、確定応答を選択的に使用することができます。続行する前にデータの送達を検査する必要があるトランザクションでは、最終 SEND で DEFRESP オプションを使用する必要があります。

機能管理ヘッダーの使用

SNA アーキテクチャーは、あるメッセージをともなうヘッダー・フィールドの特定タイプを定義します。これは、機能管理ヘッダー (FMH) と呼ばれます。これは、メッセージとその取扱方法についての情報を伝えます。一部の論理装置の場合には、FMH の使用は必須で、その他の装置の場合には、オプションで、FMH をまったく使用することができない場合もあります。特に、FMH は、最も一般的な 3270 装置である LU タイプ 2 および LU タイプ 3 の端末には適用されません。

インバウンド FMH

FMH が入力メッセージに存在している場合には、CICS はトランザクションを実行している PROFILE 定義を参照して、それを除去するか、あるいは RECEIVE を実行したアプリケーション・プログラムに渡すかを決定します。PROFILE では、FMH を渡さないこと、データ・セットの終わりを指示する FMH のみを渡すこと、あるいはすべての FMH を渡すことを指定することができます。また、FMH をバッチ・データ交換プログラムに渡させるオプションもあります。

FMH が存在している場合には、入力メッセージの最初のバイトを占め、その長さは装置タイプによって異なります。CICS は EIB の EIBFMH フィールドをオン (X'FF') に設定して FMH が存在していることを通知し、INBFMH 状態も発生させます。これは、HANDLE CONDITION コマンドを使用するか、または RESP 値をテストすることにより検出できます。

アウトバウンド FMH

出力時に、アプリケーション・プログラムまたは CICS によって FMH を構築することができます。プログラムで FMH を提供する場合には、その FMH を出力データの前に配置し、SEND コマンドで FMH オプションを指定してください。CICS が FMH を構築する場合には、CICS が埋め込むためにメッセージの最初の 3 バイトを予約し、FMH オプションを省略します。CICS は FMH を必要とする装置に対してのみ FMH を構築します。FMH がオプションになっている装置には、それを提供する必要があります。

割り込みの防止 (ブラケット・プロトコル)

ブラケットは、2 つの LU 間の会話が第 3 の LU からの要求によって割り込まれないようにするための SNA プロトコルです。CICS はブラケット・プロトコルを使用して、CICS タスクとそのプリンシパル装置の間の会話のタスク期間中の割り込みを防ぎます。タスクが代替装置を持っている場合には、ブラケット・プロトコルが同じ理由でそこでも使用されます。論理装置がブラケットを開始するのは、非送信請求入力を送信して、タスクを開始する場合で、CICS がブラケットを開始するのは、タスクを自動的に開始する場合です。最終の RETURN コマンドに IMMEDIATE オプションが現れない限り、CICS はタスク終了時にブラケットを終了します。RETURN IMMEDIATE によって、入力できるようにすることなく、プリンシパル装置で別のタスクを開始させます。ブラケットが使用中の場合には、CICS は終了中のタスクとその後続のタスクの間のブラケットを終了しない でこれを行います。

CICS は、VTAM のもとで、多くの装置のためにブラケットの使用を必要としています。その他の場合には、ブラケットの使用は、端末定義の BRACKET オプションの値によって判別されます。ブラケット・プロトコルは SNA の機能であるため、SNA 以外の装置に対して BRACKET(YES) を指定すると、CICS は厳密なブラケット・プロトコルに従わず、実行もしません。

一般に、ブラケット・プロトコルはアプリケーション・プログラムに対して透過的ですが、SEND コマンドで LAST オプションを使用して、ブラケット・プロトコルに対するフローをなおも最適化することができます。特定の SEND がタスク内にある端末の最後のコマンドであることが分かっている場合、LAST オプションを追加してパフォーマンスを改善することができます。 LAST により、VTAM は「ブラケットの終わり」標識をデータと一緒に送信し、タスク終了時に送信する個別の伝送を節約します。プログラム構築チェーン (CNOTCOMPL を使用して) の中の最終出力を送信中の場合には、有効的にするために、そのチェーンに関する最初の SEND に LAST を指定しなければなりません。

タスクが膨大な作業を実行しようとしているか、あるいはその最終 SEND の後で著しい遅延に陥っている可能性がある場合には、FREE コマンドを実行したくなる場合があります。FREE は、別のタスクで使用するために端末を解放します。

第 40 章 順次端末サポートの使用

CICS がサポートするさまざまなタイプの端末の 1 つは、実際には端末ではなく、端末をシミュレートする一対の順次装置やファイルです。対の一方は端末の入力側を表し、カード読み取り装置、スプール・ファイル、またはテープまたは DASD 上の SAM ファイルである場合があります。対の他方は出力を表し、プリンター、穿孔装置、スプール、または SAM ファイルとすることができます。装置タイプの多くの組み合わせが使用でき、対のいずれか一方を省くことができます。すなわち、入力専用または出力専用端末を持つことができます。

端末制御コマンド、特に RECEIVE、SEND、および CONVERSE コマンドで、順次端末を構成する装置またはファイルの読み取りおよび書き込みを行います。(BMS は順次端末もサポートします。 796 ページの『3270 以外の端末の特殊オプション』を参照してください。)

順次端末サポートの本来の目的は、アプリケーション開発者が、実際の端末にアクセスする前にオンライン・コードをテストすることができるようにするためでした。この要件はもはやほとんど起こりませんが、順次端末はなおも次の場合に有用です。

印刷
756 ページの『非 CICS プリンターに関するプログラミング』を参照してください。順次端末は、特に、低速 CICS プリンターに向けることがある出力、BMS または端末制御コマンドを必要とする出力、および高速システム・プリンターに向けることがある出力 (スプールまたは一時データ・コマンド) に有用です。高速プリンターを順次端末として定義する場合には、端末制御コマンドまたは BMS コマンドを使用することができ、さらに両方のタイプのプリンターに同一のコードを使用することができます。 (装置データ・ストリームに相違がある場合には、完全な透過性のために BMS を使用する必要があります。)

レグレッション・テスト
順次端末から実行するテストでは、入力と出力の両方の永続的な記録が残ります。これにより、体系的および実証的な初期テストが促進されます。また、変更後にテストを反復して、ある一組の入力が、変更後に変更前と同じ一組の出力を生成することを確認することができます。

初期設定
一部のインストール・システムでは順次端末を使用して、プログラム・リスト・テーブルのプログラムに先立って、1 つ以上の初期設定トランザクションを実行します。順次端末から開始されるトランザクションは、端末がサービスを開始するとすぐに実行を始め、CICS が処理することができる速さと同様の速度で、入力を処理し終わるまで実行を続行します。このため、順次端末が最初から稼働している場合、またはこの少し後で (端末がサービスを開始するとき)、あるいは制御されたシャットダウンの一部として、順次端末からの入力を始動後すぐに処理することができます。

© Copyright IBM Corp. 1989, 2009

順次端末に対するコーディングの考慮事項

順次端末から送信される入力データは、通信装置から送られてくるとおりの形式になっていなければなりません。例えば、最初のレコードは、通常、CICS にどのトランザクションを開始するかを指示するために、トランザクション・コードで始まります。実際の端末でそうしなければならないのとまったく同様に、トランザクション・コードは、入力の先頭位置から始まっていなければなりません。この点が、複雑な形式の入力を必要とするアプリケーションをテストする機能の限界となることに注意してください。例えば、すべての複雑な制御シーケンスのために、形式設定 3270 入力ストリームを順次ファイルとして表す方法はなにもありません。しかし、入力に不定形式 3270 データ・ストリーム (またはその他の任意のストリーム) を使用することができ、なおも BMS を使用して出力を形式設定することができます。

入力ファイルを構築する場合には、データ終了標識 (EODI) 文字をユーザーの入力レコードのそれぞれの後ろに入れます。EODI 文字はシステム初期設定テーブルに定義されます。デフォルト値はバックスラッシュ (\、X'E0') ですが、ご使用のシステムでは別の値が定義されている場合があります。

入力ストリームを処理すると、CICS は EODI 文字を監視します。CICS は、入力ファイルまたは装置のレコード構造を分析しません。つまり、各入力は入力ファイル内のレコードをスパンできるということです。ただし、新規物理レコードの各入力を開始して、各入力が正しく処理されるようにする必要があります。

入力レコードの長さ (EODI の間の文字数) が入力バッファーのサイズ (順次端末定義の LINE コンポーネントの INAREAL 値) を超えてはいけません。超えた場合には、長いレコードを受信 (RECEIVE) しようとするトランザクションは異常終了し、CICS は、入力処理を再開する前に、入力ファイルを次の EODI の後に位置付けます。

入力内のファイルの終わりマーカーも EODI 標識として作用します。ファイルの終わりが検出された後で実行された RECEIVE コマンドも異常終了の原因になります。

印刷形式設定

順次端末の定義に、出力先がライン・プリンターであることが指示されている場合には、単一の SEND を使用して複数の行を書き込むことができます。このタイプの装置の場合には、各改行文字 (X'15') の後、あるいは行の長さとして定義される文字数の後のいずれかで、CICS はユーザーの出力メッセージを複数の行に分割します。行の長さは端末定義の LPLEN によって定義します。各 SEND は新規の行を始めます。

GOODNIGHT 規則

CICS は、CICS (またはトランザクションそのもの) がすべての入力を処理し終えるまで、あるいは端末がサービス休止になるまで、順次端末からのトランザクションの開始を続行します。CICS が入力ファイルの終わりを超えて読み取ろうとする (これは異常終了の原因になります) のを防ぐために、最後に実行されるトランザクションは、その最終出力の後で端末をサービス休止にすることができます。あるいは (この方が通常は容易ですが)、最終入力を CESF GOODNIGHT トランザクションと

することができます。これは端末にオフの信号を送り、それをサービス休止にします。 通常、そのトランザクションをサービスの対象外にせずに、一度 CICS が元の入力を処理してしまうと、順次端末からさらに入力することはできません。

第 41 章 TCAM の使用

重要

CICS Transaction Server for z/OS バージョン 4 リリース 1 では、ローカル TCAM 端末はサポートされません。サポートされている TCAM 端末は、TCAM の DCB インターフェース (ACB ではない) によって CICS TS 3.1 以前の端末専有領域に接続されているリモート端末のみです。そのため、TCAM に対応する方法は、トランザクション・ルーティングによる方法、または TCAM/DCB による端末の接続先であるリモートの CICS TS 3.1 以前の端末専有領域からの機能シップによる方法のみです。

TCAM/DCB インターフェースのコーディング

CICS は、TCAM の DCB インターフェースを直接にはサポートしていません。このインターフェースを通して端末を使用するには、589 ページの『第 33 章 端末アクセス方式のサポート』で説明したように、CICS の古いバージョンを介してそうする必要があります。一般に、そのような装置用には、VTAM を介して接続されたときに使用するのと同じ端末制御コマンドまたはオプションを使用します。しかし、ユーザーの CICS アプリケーション・プログラムと端末との間のパスはずっと複雑なので、その結果他にもっとたくさんのプログラミングの可能性があります。

CICS-TCAM インターフェースに関するプログラミングについて詳しくは、ご使用の古いバージョンの CICS (TCAM 端末の接続先である端末専有領域) の「*Customization Guide*」を参照してください。

© Copyright IBM Corp. 1989, 2009

第 42 章 バッチ・データ交換の使用

CICS バッチ・データ交換プログラムは、アプリケーション・プログラムと、外部コントローラー内のバッチ・データ交換論理装置の一部であるか、あるいはバッチ論理装置上または LU タイプ 4 論理装置上に選択したメディアを持つ、名前を指定したデータ・セット (または宛先) との間の通信を提供します。このメディアとは、プリンターまたはコンソールなどの必要な装置を意味します。

外部コントローラーという用語は、IBM 3770 データ・コミュニケーション・システム、IBM 3790 データ・コミュニケーション・システム、または DPCX を実行中の IBM 8100 システムなどの、SNA プロトコルを使用するプログラマブル・サブシステムを総称したものです。使用可能な SNA プロトコルおよびデータ・セットの詳細については、「*IBM 3767/3770/6670 Guide*」を参照してください。図 120 では、バッチ・データ交換の概要を説明しています。

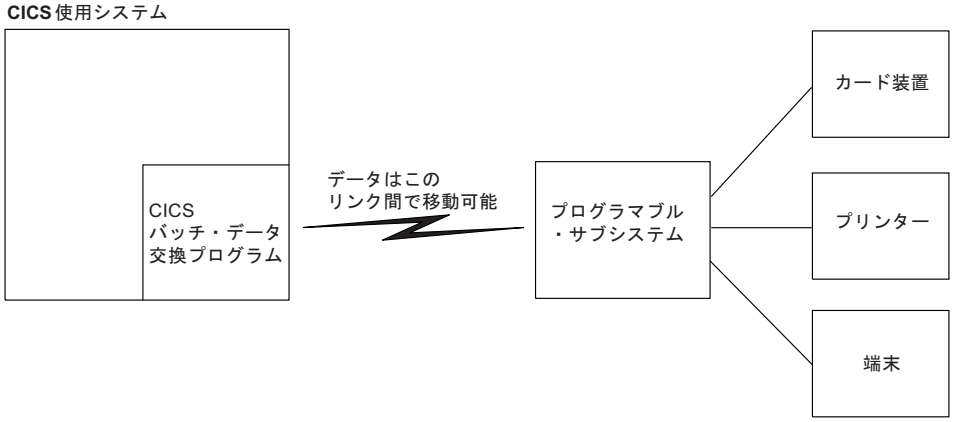


図 120. CICS バッチ・データ交換

次のバッチ・データ交換コマンドが提供されます。

ISSUE QUERY
CICS アプリケーション・プログラムへのデータ・セットの転送を開始します。

ISSUE RECEIVE
データ・セットからレコードを読み取るか、あるいは入力メディアからデータを読み取ります。

ISSUE SEND
名前付きデータ・セットまたは選択したメディアへデータを伝送します。

ISSUE ADD
データ・セットにレコードを追加します。

ISSUE REPLACE
データ・セットのレコードを更新します (置き換えます)。

© Copyright IBM Corp. 1989, 2009

ISSUE ERASE
データ・セットからレコードを削除します。

ISSUE END
データ・セットの処理を終了させます。

ISSUE ABORT
データ・セットの処理を異常終了させます。

ISSUE NOTE
データ・セットの次のレコード番号を要求します。

ISSUE WAIT
操作が完了するのを待ちます。

コントローラーが LU タイプ 4 論理装置である場合には、ISSUE ABORT、ISSUE END、ISSUE RECEIVE、ISSUE SEND、および ISSUE WAIT コマンドしか使用することができません。

データ・セットが DPCX/DXAM データ・セットである場合には、ISSUE ADD、ISSUE ERASE、および ISSUE REPLACE コマンドしか使用することができません。

バッチ・データ交換コマンドの実行中に起こる例外条件の取扱方法については、383 ページの『第 24 章 例外条件の取り扱い』を参照してください。

宛先の選択および識別

ISSUE RECEIVE 以外のすべてのバッチ・データ交換コマンドには宛先を指定するオプションが含まれます。これは、バッチ・データ交換論理装置内の名前付きデータ・セット、またはバッチ論理装置か LU タイプ 4 論理装置内で選択されたメディアのいずれかです。

名前付きデータ・セットを使用して宛先を選択する場合は、DESTID および DESTIDLENG の各オプションを常に指定し、データ・セット名およびその長さ (最大 8 文字まで) を提供する必要があります。ディスケットを持つ宛先の場合には、VOLUME オプションおよび VOLUMELENG オプションを指定して、ボリューム名とその長さ (最大 6 文字まで) を提供する場合があります。ボリューム名は操作に使用するデータ・セットが入っているディスケットを識別します。VOLUME オプションが複数ディスケット宛先に指定されていない場合には、必要なデータ・セットが見付かるまで、すべてのディスケットが検索されます。

宛先として、データ・セット名を指定する代わりに、CONSOLE、PRINT、CARD、または WPMEDIA1-4 の各オプションによってさまざまなメディアを指定できます。これらのメディアは ISSUE ABORT、ISSUE END、ISSUE SEND、または ISSUE WAIT コマンドの中だけでしか指定することができません。

確定応答 (DEFRESP オプション)

CICS は端末制御コマンドを使用して、バッチ・データ交換コマンドに指定された機能を実行します。端末管理出力要求を行わせるコマンドの場合には、DEFRESP オプションを指定することができます。このオプションは SEND 端末制御コマンドの DEFRESP オプションと同じ効果を持っています。すなわち、外部コントローラーか

ら確定応答を要求するためには、(システム・プログラマーによる) CICS タスクに関するメッセージ保全性の指定とは無関係になります。DEFRESP オプションは、ISSUE ADD、ISSUE ERASE、ISSUE REPLACE、および ISSUE SEND コマンドに指定することができます。

機能の完了の待機 (NOWAIT オプション)

端末管理出力要求を行わせるバッチ・データ交換コマンドの場合には、NOWAIT オプションを指定することができます。このオプションは、CICS タスク処理を続行することができる効果があります。NOWAIT オプションが指定されていない限り、バッチ・データ交換コマンドが完了するまで、タスク・アクティビティーは延期されます。NOWAIT オプションは、ISSUE ADD、ISSUE ERASE、ISSUE REPLACE、および ISSUE SEND コマンドだけにしか指定することができません。

NOWAIT オプションを指定したバッチ・データ交換コマンドを実行した後で、タスク・アクティビティーは ISSUE WAIT コマンドによってプログラム内の適切な地点で延期されて、コマンドが完了するのを待機することができます。

第 43 章 端末管理: パフォーマンスの設計

端末管理では、パフォーマンス上の考慮が多数あります。

- **端末に送信するデータ・ストリーム長を最小にします。**

  3270 ハードウェア機能の優れた画面設計および効率的使用が、テレプロセシング・リンクで伝送されるバイト数に重大な影響を与えることがあります。ほとんどの場合、これはトランザクションが使用する経路の中で最も速度が遅い部分なので、バイト数をできるだけ小さくすることは特に重要です。したがって、データ・ストリームの効率は応答時間と回線使用の両方に影響します。

- **物理的な SEND コマンドは、画面ごとに 1 つのみ使用します。**

  通常、一連の SEND MAP ACCUM コマンドで画面を作成するより、BMS への単一呼び出しで画面を作成する方がより効率的です。単一の物理的出力で画面を端末に送信することが重要です。画面をいくつかに分けて作成し、それぞれの部分を別個のコマンドで送信するのは、複数のコマンドを使用することによる追加のプロセッサー・オーバーヘッド、および追加の回線オーバーヘッドとアクセス方式オーバーヘッドのために、***非常に*** 非効率的です。

- **CONVERSE コマンドを使用します。**

  SEND および RECEIVE コマンド (あるいは、プログラムが会話型の場合は、SEND、WAIT、RECEIVE コマンド・シーケンス) ではなく、CONVERSE コマンドを使用してください。これらのコマンドは機能的には同等ですが、CONVERSE コマンドは CICS サービス・インターフェースを一度しか通らないので、プロセッサー時間の節約になります。

- **メッセージ保全性オプションの使用は制限します。**

  トランザクションの最後の SEND コマンドで WAIT オプションを指定するのと同じように、CEDA の MSGINTEG オプションは、最終メッセージが正常に送達されるまで、CICS にトランザクションの実行を続けさせます。 PROFILE 定義の PROTECT オプションは、暗黙のメッセージの保全を意味し、システムに入出力メッセージをすべてログをとらせます。このため、I/O およびプロセッサーのオーバーヘッドが増えます。

- **SEND コマンドでは DEFRESP オプションを使用しないようにします。**

  トランザクションで出力メッセージの正常な送達を検査していない限り、SEND コマンドで DEFRESP オプションの使用を避けてください。このオプションは MSGINTEG と同じ方法でトランザクションの終了を遅らせます。

- **不必要なトランザクションは使用しないようにします。**

  ユーザーが不正確なトランザクションを入力する可能性があるか、あるいは不必要に CLEAR キーを使用する可能性があるような状態を避けてください。こうすることにより、端末入力、タスク制御処理、端末出力、およびオーバーヘッドに負荷が加わるのを回避します。優れた画面設計および標準化された PF キーおよび PA キーの割り当てによって、この点が最小化されます。

© Copyright IBM Corp. 1989, 2009

- **不定形式データはマップなしで送信します。**

  端末への出力の全部またはほとんどが不定様式の場合には、BMS コマンドではなく、端末制御コマンド (すなわち、MAP オプションまたは TEXT オプションを指定しない BMS SEND コマンド) を使用して、送信することができます。

第 44 章 3270 ファミリーの端末

3270 は、サポートする制御装置を持つディスプレイ端末およびプリンター端末のファミリーで、共通する特性を共有し、エンコードされた同一データ形式を使用して端末とホスト・プロセッサーの間の通信を行います。このデータ形式は *3270* データ・ストリームと呼ばれます。

3270 は多くの機能と可能性を持つ複雑な装置です。ここでは基本的な操作のみを説明し、CICS が 3270 をサポートする方法を重点的に説明します。3270 の各機能、プログラミング、およびデータ・ストリーム・フォーマットに関する総合的な説明については、「*3270 Data Stream Programmer's Reference*」を参照してください。「*IBM 3270 Data Stream Device Guide*」にも、重要な情報が含まれています。この情報は、主として端末管理を使用するプログラマーのためのものですが、BMS プログラマーにも役立つ情報が含まれています。一部の特殊な機能に対する BMS のサポートについては、877 ページの『第 71 章 特殊ハードウェアのサポート』に説明されています。

ここではディスプレイ端末を中心に説明していますが、ほとんどの資料が 3270 プリンターに対しても同様に適用されます。3270 プリンターは、3270 ディスプレイと同じデータ・ストリームを受け入れ、画面イメージをハードコピー用紙に送達します。ほとんどの違いは、プリンターには (大部分が) 無い入力と関連しています。

しかし、追加の形式制御機能がプリンターに使用可能で、印刷出力を希望のプリンターに送る場合には特殊な考慮事項があります。詳細については、741 ページの『第 57 章 印刷の CICS サポート』を参照してください。

3270 バッファー

3270 装置との通信は、その**文字バッファー**を介して行われます。これはプロセッサー内のしており、一と同様のハードウェア・ストレージ機構です。3270 への出力はバッファーへ送られます。次に、バッファーはディスプレイ端末の表示およびプリンター端末の印刷機構を駆動します。

それとは逆に、635 ページの『3270 端末からの入力』に説明されているとおりに、キーボード入力はバッファーを介してホストに達します。

画面上のそれぞれの位置はバッファー内の位置と対応し、そのバッファー位置の内容は画面上の表示内容を判別します。画面がフィールドに形式設定されている場合には、各フィールドの先頭位置は、そのフィールドの一定の表示特性を保管するために使用しているので、データを表示するためには利用不能です (ブランクとして現れます)。3270 の初期のモデルでは、表示特性のすべてを保管するために、このバイトで十分でした。さらに多くのタイプの表示特性を持つ後期のモデルでは、画面上の固定位置とは関連していないバッファー・ストレージの区域に補足情報が保持されます。表示特性の詳細については、626 ページの『表示特性』を参照してください。

© Copyright IBM Corp. 1989, 2009

出力データ・ストリーム

3270 表示を作成するためには、以下のものから成るデータのストリームを送信します。

- 書き込みコマンド (1 バイト)
- 書き込み制御文字または **WCC** (1 バイト)
- 表示データ (可変バイト数)

WCC および表示データは常に存在しているわけではありません。書き込みコマンドは WCC が続いているかどうか、およびデータが存在していいか、あるいは存在しなければならないかを判別します。

BMS を使用する場合には、CICS がユーザーのデータ・ストリーム全体を構築します。WCC は SEND コマンドのオプションからアセンブルされ、書き込みコマンドは他の SEND オプションおよび実行中のトランザクションの PROFILE の中の情報から選択されます。表示データは、ユーザーが提供するマップまたはテキスト・データから構築され、BMS がこれを 3270 形式に変換します。

SEND などの端末管理コマンドを使用する場合には、CICS が同一情報から構築される書き込みコマンドを提供します。しかし、ユーザーが WCC を用意し、3270 形式で表示データを表現しなければなりません。

3270 書き込みコマンド

データまたは指示を端末に送信する 3270 コマンドには、以下の 5 つがあります。

- 書き込み
- 消去 / 書き込み
- 消去 / 書き込み代替
- 無保護フィールド全消去
- 構造化フィールド書き込み

3270 **書き込み**コマンドは、それに続くデータを、画面 (またはプリンター) が駆動される 3270 バッファーに送ります。**消去 / 書き込み**および**消去 / 書き込み代替**もこれを行いますが、これらは最初にバッファーを消去します (すなわち、バッファー全体にヌル値を設定します)。また、端末が**代替画面サイズ**と呼ばれる機能を持っている場合には、これらはバッファー・サイズ (画面上の行数および桁数) の判別も行います。

この機能を持つ端末は、2 つのサイズ、すなわち、**デフォルトのサイズ**および**代替サイズ**を持っています。消去 / 書き込みコマンドによって、デフォルトのサイズが (次の消去 / 書き込みコマンドまたは消去 / 書き込み代替コマンドまで) 後続の操作で使用され、消去 / 書き込み代替は名前が示唆するように代替サイズを選択します。

SEND コマンドに ERASE オプションを組み込まない限り、CICS は普通の書き込みコマンドを使用してデータを送信します。SEND に ERASE DEFAULT を指定した場合には、CICS は代わりに消去 / 書き込み (画面をデフォルトのサイズに設定) を使用し、ERASE ALTERNATE の場合は、CICS は消去 / 書き込み代替 (代替サイズを設定) を使用します。DEFAULT または ALTERNATE なしで ERASE を指

定した場合には、CICS は実行するトランザクションに関連した PROFILE 定義を見て、消去 / 書き込みを使用するか、あるいは消去 / 書き込み代替を使用するかを判別します。

アドレスまでの無保護消去コマンドによって、無保護フィールドに関するバッファーのスキャンが行われます (これらのさらに厳密な定義は、626 ページの『3270 フィールド属性』にあります)。このようなフィールドが見付かるとすべてヌルに設定されます。この選択的消去は、809 ページの『SEND CONTROL コマンド』 で説明されているように、データ入力操作の場合に役立ちます。このコマンドには WCC またはデータを続けません。コマンドだけを送信します。

構造化フィールド書き込みによって、それに続くデータは 3270 構造化フィールドとして解釈されます。**構造化フィールド**は 3270 の拡張機能の一部に必要です。ここでは、これについて説明しませんが、これは STRFIELD オプションを含んでいる端末管理 SEND コマンドを使用して書くことができます。詳しくは、「*IBM 3270 Data Stream Device Guide*」を参照してください。

書き込み制御文字

3270 書き込み、消去 / 書き込み、または消去 / 書き込み代替コマンドに続くバイトは**書き込み制御文字**または **WCC** です。WCC は、以下のことを行うかどうかを 3270 に指示します。

- 音響アラームの鳴動
- キーボードのアンロック
- 変更データ・タグをオフにする
- 印刷の開始 (端末がプリンターの場合)
- 構造化フィールドのリセット
- インバウンド応答モードのリセット

BMS では、CICS が、SEND MAP コマンドの ALARM、FREEKB、FRSET、および PRINT オプションから WCC を作成します。端末制御コマンドを使用する場合には、CTLCHAR オプションを使用して WCC を明示的に指定することができます。端末制御コマンドを使用しない場合には、CICS は、キーボードをアンロックし、変更データ・タグをオフにする WCC を生成します (これらの簡単な説明が、627 ページの『変更』にあります)。

3270 表示フィールド

表示データは、表示する文字の組み合わせおよびデータを表示する方法と場所についての装置への指示から構成されています。通常の環境のもとでは、639 ページの『不定形式モード』に説明されているように、フィールドを定義しないで画面を書き込むことができますが、このデータは一連のフィールド定義から構成されています。

消去する書き込みコマンドの後に、画面上のすべてのフィールドを定義する必要があります。その後で、通常の書き込みコマンドを使用し、変更したいフィールドだけを送信することができます。

フィールドを定義するためには、3270 に以下の点を指示する必要があります。

- フィールドの表示方法
- フィールドの内容
- 画面上の表示位置 (すなわち、バッファー内の開始位置)

表示特性

画面上の各フィールドは、**属性**と呼ばれる一組の表示特性を持っています。属性は 3270 にフィールドをどのように 表示するかを指示し、BMS を使用するのか、あるいは端末制御コマンドを使用するのかの実現性を理解する必要があります。属性は以下の 2 つのカテゴリーに分類されます。

フィールド属性

以下のことが含まれます。

- 保護 (オペレーターがフィールドを変更することができるかできないか)
- 変更 (オペレーターがフィールドを変更した かどうか)
- 表示輝度

すべての 3270 はフィールド属性をサポートします。フィールド属性の選択項目については、『3270 フィールド属性』に説明があります。

フィールド属性はフィールドの先頭の文字位置に保管されます。このバイトは画面上で 1 桁を占め、フィールド属性を保管するだけではなく、フィールドの先頭もマークします。フィールドは次の属性バイト (すなわち、次のフィールドの先頭) まで継続します。次のフィールドが同一行で始まっていない場合には、現行のフィールドが現在の行の終わりから、別のフィールドが見付かるまで、次の行の始めに折り返します。最後の行で終わっていないフィールドは先頭に戻ります。

拡張フィールド属性

(通常、**拡張属性**と短縮されます。) これらは 3270 の全モデルに存在しているわけではありません。したがって、エンド・ユーザー・インターフェースを設計する場合には、どれが使用可能かに注意する必要があります。拡張属性には、強調表示およびアウトラインの特殊形式、複数のシンボル・セットを使用する機能、および 2 バイト文字セットの用意が含まれます。 628 ページの表 34 には、7 つの拡張属性、およびその属性で使用可能な値がリストされています。

3270 フィールド属性

前述のとおり、フィールド属性バイトはフィールドの保護、変更、および表示輝度属性を保持します。ここでは、これらの属性のそれぞれの選択項目について、BMS が形式の定義で使用する用語を使用して説明します。端末制御コマンドを使用する場合には、属性バイト内のビットを、選択した値を反映するように設定する必要があります。

(ビットの割り当てについては、「*IBM 3270 Information Display System Data Stream Programmer's Reference*」を参照してください。また、この分野における CICS のヘルプについては、 806 ページの『属性値の定義: DFHBMSCA』を参照してください。)

保護

属性バイト内で最大 2 ビットの位置を使用する保護属性の選択項目は以下の 4 つです。それは、以下のとおりです。

無保護 オペレーターは、無保護フィールドには任意の文字を入力することができます。

数値専用
この指定の効果は端末のキーボード・タイプによって異なります。データ入力キーボードで、オペレーターがシフトしないで数字をキー入力できるように、数字シフトが行われます。「数値ロック」特殊機構が装備されたキーボードでは、キーボードがロックされるのは、オペレーターが数字 0 ～ 9 の 1 つ、ピリオド (10 進小数点)、ダッシュ (マイナス符号)、または DUP キー以外の任意のキーを使用した場合です。受信プログラムはなおも入力を検査して、それが必要としている形式の数値になっていることを確認する必要はあるものの、これは、オペレーターがフィールドの中に英字データをキー入力することを防ぎます。数値ロック機能を使用しないで、数値専用によって、任意のデータをフィールドに入力することができます。

保護 オペレーターは保護フィールドにキー入力することはできません。入力しようとするとキーボードがロックされます。

自動スキップ
オペレーターは自動スキップ・フィールドにはキー入力することはできませんが、カーソルが異なった動作をします。 (カーソルは、オペレーターが次にキー・ストロークするはずの場所を指示します。この点の詳細については、 635 ページの『3270 端末からの入力』を参照してください。) カーソルが (直前のフィールドが埋め込まれたためか、あるいはフィールド前進キーが使用されたために) 新規フィールドに進む場合には常に、カーソルはそのパス上のすべての自動スキップ・フィールドをスキップして、無保護または数値専用のいずれかである最初のフィールドに進みます。

変更

フィールド属性内で 1 ビットしか占めない 2 番目の項目は**変更データ・タグ**または **MDT** と呼ばれます。MDT は、フィールドが変更されたか、あるいは変更されなかったかを示します。オペレーターがフィールド内容になにか変更を行った場合には常に、ハードウェアがこのビットを自動的にオンにします。CICS が通常使用する読み取りコマンドの場合は、インバウンド・データにフィールドが含まれているか、あるいは含まれていないかをそのコマンドが判別するために、MDT ビットは非常に重要です。このビットがオンになっている (すなわち、フィールドが変更された) 場合には、3270 はそのフィールドを送信し、そうでない場合には、そのフィールドは送信しません。

また、あるフィールドを画面に送信する場合には、プログラムによって MDT をオンにすることもできます。この機能を使用すると、オペレーターがフィールドを変更できなかったか、あるいは変更しなかったとしても、読み取り時にそのフィールドが確実に返されます。BMS SEND コマンドの FRSET オプションによって、画面上のすべてのフィールドのタグをプログラムによってオフにすることができます。

プログラムによって個別のタグをオフにすることはできません。端末制御コマンドを使用している場合には、個別タグをオフにするために、WCC のビットをオンにします。

輝度

属性バイトに保管される 3 番目の特性はフィールドの表示輝度です。3 つの選択項目があり、これらは一緒には使用できません。

通常輝度
フィールドは装置の通常輝度で表示されます。

高輝度 フィールドは、強調表示されるように、通常輝度より高い輝度で表示されます。

非表示 フィールドはまったく表示されません。フィールドにはバッファー内のデータが入ることがあり、オペレーターはそのフィールド (保護または自動スキップとして用意されていない場合) にキー入力することができますが、データは画面上で見えません。

2 つのビットが表示輝度用に使用され、これにより、上にリストされている 3 つより 1 だけ多い値を表現することができます。関連した特殊ハードウェア機能のいずれかを持つ端末の場合には、これら同一の 2 ビットを使用して、フィールドが検出可能なライト・ペンか、あるいは選択可能なカーソルかを判別します。2 ビットしかないので、輝度と選択可能性のすべての組み合わせが可能なわけではありません。高輝度フィールドは常に検出可能で、非表示フィールドは検出されることはなく、通常輝度フィールドはいずれにもできるというのが妥協案です。これらの機能について詳しくは、881 ページの『カーソルおよびペンで検出可能なフィールド』を参照してください。

基本カラー

端末の中には、**基本カラー**をサポートし、拡張属性に含まれる**拡張カラー**をサポートしないものや、それをサポートするものもあります。このような端末の正面には、オペレーターが基本カラーまたはデフォルト・カラーを選択することができるように、モード・スイッチがあります。フィールド属性が高輝度 (この場合は白になります) を指定しない限り、デフォルト・カラーは文字を緑で表示します。基本カラー・モードの場合には、保護および輝度ビットを組み合わせて使用して、通常の白、赤、青、および緑の 4 色から選択します。保護ビットはカラーを判別するだけでなく保護機能を保存します。(3270 端末に対して基本カラーではなく拡張カラーを使用する場合は、カラーに「白色」を指定できないことに注意してください。端末では白で表示される「中間色」を指定する必要があります。)

拡張属性

ここで説明したフィールド属性に加えて、3270 端末の中には、この他の拡張属性を持つものがあります。表 34 には、1 列目に拡張属性のタイプが、2 列目に各タイプごとの使用される値がリストされています。

表 34. 3270 拡張属性

属性タイプ	値
拡張カラー	青、赤、ピンク、緑、空色、黄色、中間色

表 34. 3270 拡張属性 (続き)

属性タイプ	値
拡張強調表示	明滅、反転表示、下線
フィールド・アウトライン	上線、下線、左右縦線の任意の組み合わせ
背景透明	背景透明、背景不透明
フィールド妥当性検査	入力する必要があるフィールド、埋める必要があるフィールド、トリガーになるフィールド。
プログラム式シンボル・セット	シンボル・セットを識別する番号 **注:** 端末と関連した制御装置にはデフォルト・シンボル・セットが含まれ、最大 5 つまでの追加シンボル・セットを保管することができます。これら他の 1 つを使用するためには、使用に先立って、シンボル・セットをコントローラーの中にロードする必要があります。これは、端末管理 SEND コマンドを使用して行うことができます。
SO/SI 作成	2 バイト文字を示すシフト文字が存在する、シフト文字が存在しない。

「*IBM 3270 Information Display System Data Stream Programmer's Reference*」には、拡張属性の詳細、およびデフォルト値の決定方法が説明されています。ユーザーの特定端末が持つ拡張属性を判別するためには、ASSIGN コマンドおよび INQUIRE コマンドを使用することができます。これらのコマンドの説明は、603 ページの『第 38 章 ご使用の端末情報の検出』にあります。

また、3270 の一部のモデルによって、全体としてフィールドの値とは異なっているフィールド内の個別の文字に拡張属性値を割り当てることができます。一般に、BMS は文字属性に明示の割り当てを行わないので、これは端末制御コマンドを使用して行う必要があります。ただし、849 ページの『テキスト行』に説明されているように、BMS においては、テキスト出力の文字属性に制御シーケンスを挿入できます。このようなシーケンスの形式は、632 ページの『属性設定オーダー』に説明されています。

データ・ストリーム中のオーダー

このタスクについて

次のセクション以降では、フィールドの属性、位置、および内容を表現するためにアウトバウンド・データを形式設定する方法について説明します。この情報を知っておく必要があるのは、端末制御コマンドを使用して 3270 に書き込む場合です。BMS を使用する場合には、このすべてが行われるので、635 ページの『3270 端末からの入力』に進むことができます。

3270 データ・ストリームにフィールドを定義する場合には、**フィールド開始 (SF)** オーダーまたは**フィールド開始拡張 (SFE)** オーダーで始めます。**オーダー**は 3270 への指示です。オーダーはバッファーのロード方法を 3270 に指示します。オーダーの長さは 1 バイトで、通常はその後にオーダー固有の形式のデータが続きます。

フィールド開始オーダー

SF オーダーはすべてのモデルでサポートされ、フィールドの属性および表示内容を指定することができますが、拡張属性は指定できません。 SF によってフィールドを定義するには、図 121 に示されているように、データ・ストリーム内にシーケンスを挿入します。

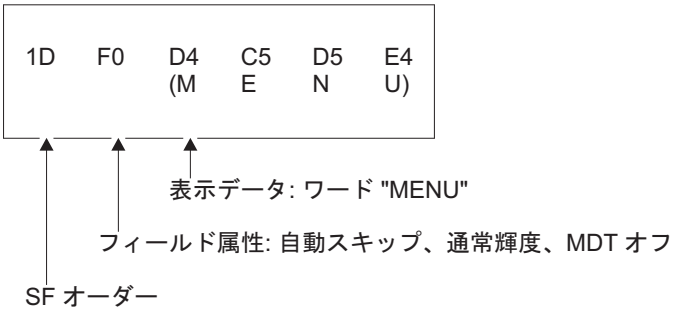


図 121. SF オーダーを使用したフィールド定義

拡張属性を指定する必要があり、端末がそれをサポートしている場合には、代わりにフィールド開始拡張オーダーを使用します。さらに複雑な属性情報のために、SFE には異なる形式が必要です。拡張属性はバイトの対として表現されます。最初のバイトは定義する属性のタイプがどれかを指示するコードで、2 番目のバイトはその属性の値です。フィールド属性は、追加属性タイプとして集合的に取り扱われ、バイトの対としても表されます。SFE オーダーの直後に属性の対の 1 バイト数を、その後に属性の対を、最後に表示データを、それぞれ与えます。シーケンス全体は、図 122 に示されています。

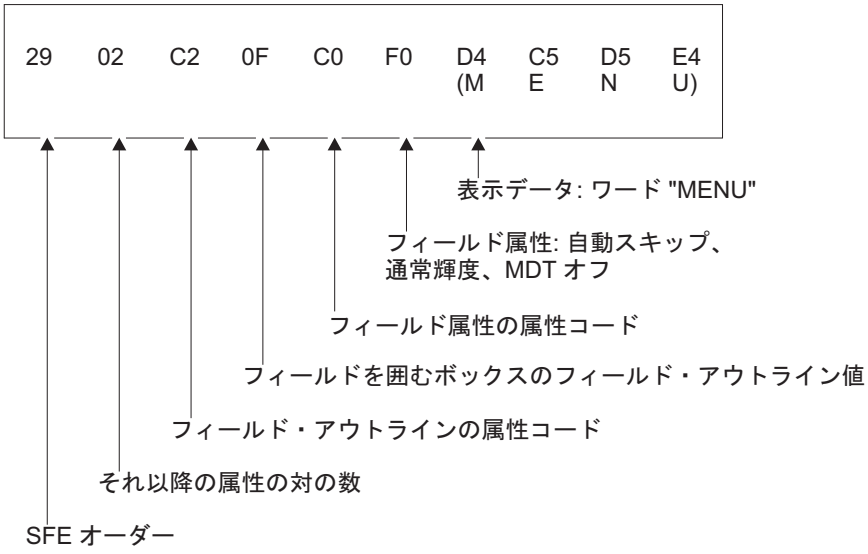


図 122. SFE オーダーを使用したフィールド定義

フィールド変更オーダー

フィールドが画面上にある場合には、SFE の形式とほとんど同一の、**フィールド変更 (MF)** と呼ばれるコマンドによってそのフィールドを変更することができます。SFE との相違は、以下の点だけです。

- フィールドは既に存在している必要がある。
- コマンド・コードは X'29' ではなく X'2C' である。
- 現行値から変更したい属性しか送信せず、変更したい場合には、表示データだけを送信する。
- ヌル値は、ユーザーの特定端末のデフォルトに、属性を設定し直す (SFE の場合には、属性を省略することによって同じことを行います)。

例えば、前出の例の「menu」フィールドを端末のデフォルト・カラーに返し、それに下線を引くには、図 123 に示されているシーケンスが必要です。

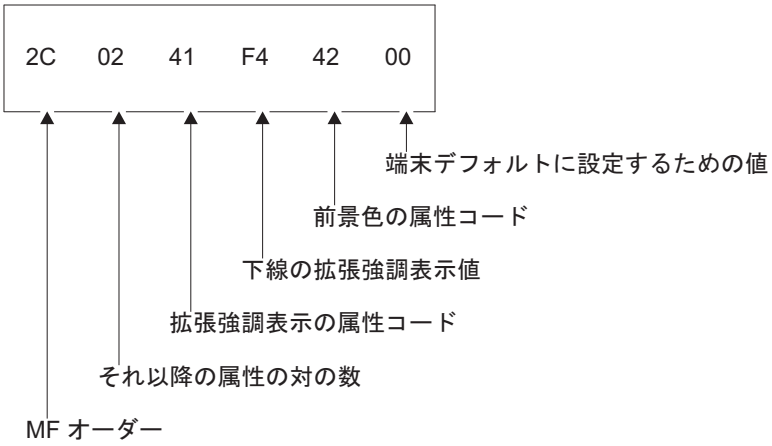


図 123. MF オーダー内のフィールド属性の変更

バッファー・アドレス設定オーダー

SF オーダーおよび SFE オーダーは、定義するフィールドをバッファーの現在位置に配置し、MF はこの位置のフィールドを変更します。フィールドが送信される最後の文字の後にない (すなわち、現行バッファー位置から始まっている) 限り、**バッファー・アドレス設定 (SBA)** オーダーをこれらのオーダーより先行させて、フィールドを配置または変更する位置を指示する必要があります。これを実行するには、632 ページの図 124 に示されているように、SBA オーダーに続けて 2 バイト・アドレスを送信します。

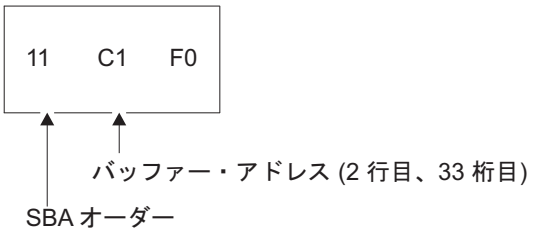


図 124. SBA シーケンス

図のアドレスは、80 桁画面における 2 行目の 33 桁目である位置 112 (X'70') の「12 ビット」アドレスです。最初の行および桁 (0 の位置) で数え始め、行に沿って進むことに注意してください。使用されるアドレッシング構造には、この他に「14 ビット」と「16 ビット」の 2 つがあります。バッファー位置のすべてに順次番号が付けられますが、12 ビットおよび 14 ビット・アドレッシングの場合には、アドレスのすべてのビットが使用されるわけではないので、順次には現れません。(図中の X'70' (B'1110000') は、アドレスの右端の下位 6 ビットに B'110000' として現れ、左のバイトの下位 6 ビットに B'000001' として現れます。) 「*IBM 3270 Information Display System Data Stream Programmer's Reference*」には、アドレスを形成する方法が説明されています。

SF、SFE、または MF オーダーの後、現行バッファー・アドレスは埋め込まなかったバッファー内の先頭位置 (ある場合には、データの直後、ない場合には、フィールド属性バイトの後ろ) を指しています。

属性設定オーダー

単一文字位置の属性を設定するためには、指定したい属性ごとに**属性設定 (SA)** オーダーを使用します。例えば、文字を明滅させるためには、図 125 にあるシーケンスが必要です。

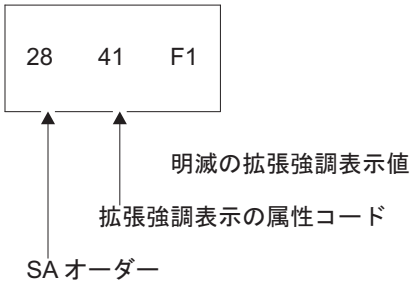


図 125. 文字を明滅させる SA シーケンス

SA オーダーによって指定する属性は、フィールド定義が現行バッファー位置に配置されるのと同じ方法で現行バッファー位置に割り当てられるので、一般に、SBA シーケンスを SA より先行させる必要があります。

アウトバウンド・データ・ストリームの例

このセクションでは、特定の 3270 画面をペイントするために必要なデータ・ストリームの注釈付きの例を示して、データ・ストリームの構築方法の説明を補足します。

図 126 は、作業現場で従業員が使用する自動車を記録し、さらに新規自動車を登録するために使用するアプリケーションの一部である画面例を示しています。入力は従業員識別番号、ライセンス・プレート (タグ) 番号、および州外からの自動車の場合には登録している州です。

```
                Car Record
Employee No: ______   Tag No: ________   State: __
```

図 126. データ入力画面の例

注: これは、説明を簡潔にまとめるために設計された、非現実的なまでに単純な画面です。この例は画面設計で一般に受け入れられる標準に適合するものではなく、これをモデルとして使用してはいけません。

この画面には、8 つのフィールドがあります。

1. 画面の表題「Car Record」(1 行目の 26 桁目)
2. ラベル・フィールド「Employee No:」(3 行目の 1 桁目)。オペレーターが次のフィールドに入力する内容を指示します。
3. 従業員番号 (Employee No) の入力フィールド (3 行目の 14 桁目)。長さは 6 桁。
4. ラベル・フィールド「Tag. No:」(3 行目の 21 桁目)。
5. 長さ 8 桁の入力フィールド (Tag.no) (3 行目の 31 桁目)。
6. ラベル・フィールド「State:」(3 行目の 40 桁目)。
7. 長さ 2 桁の入力フィールド (State) (3 行目の 49 桁目)。
8. 直前の入力フィールド (State) の終わりをマークするフィールド (3 行目の 52 桁目)。

表 35 には、アウトバウンド・データ・ストリームが示されています。

表 35. 3270 出力データ・ストリーム

バイト	内容	ノーツ
1	X'F5'	データ・ストリームを開始する 3270 コマンド。この場合には、消去 / 書き込み。
2	X'C2'	WCC。この値はキーボードをアンロックしますが、アラームの鳴動または MDT のリセットは行いません。
3	X'11'	最初のフィールドを ... に位置決めする SBA オーダー。
4-5	X'40D6'	12 ビット・アドレッシングを使用した、24 x 80 のスクリーン上で 1 行目の 23 桁目を指すアドレス。
6	X'1D'	最初のフィールド定義を始めるための SF オーダー。

表 35. 3270 出力データ・ストリーム (続き)

バイト	内容	ノーツ
7	X'F8'	フィールド属性バイト。この組み合わせは、自動スキップおよび高輝度で、MDT が最初はオフになっているフィールドを示します。
8-17	'Car record'	このフィールドの表示内容。
18-20	X'11C260'	現行バッファー位置を 2 番目のフィールド用に 3 行目の 1 桁目にリセットするための SBA シーケンス。
21	X'1D'	2 番目のフィールド用の SF オーダー。
22	X'F0'	フィールド属性バイト: 自動スキップ、通常輝度、MDT オフ
23-34	'Employee No:'	このフィールドの表示内容。
35	X'29'	4 番目のフィールドを開始するための SFE オーダー。拡張属性を指定する必要があるので、SF の代わりに SFE が必要です。このフィールドは、直前のフィールドが終わったすぐ後から始まるので、この前には SBA は必要ありません。
36	X'02'	指定される属性タイプ の数 (ここでは、フィールド・アウトラインおよびフィールド属性の 2 つ)。
37	X'41'	拡張強調表示の属性タイプを示すコード。
38	X'F4'	下線を示す拡張強調表示の値。
39	X'C0'	フィールド属性の属性タイプを指示するコード。
40	X'50'	数値専用、通常輝度、MDT オフのフィールド属性値。このフィールドのすべての初期データが次に表示されますが、ここではありません。
41	X'13'	カーソルを現行バッファー位置に位置付けることを 3270 に指示する、カーソル挿入 (IC) オーダー。カーソルは、オペレーターが埋め込む必要がある最初のフィールドの先頭に位置付ける必要があり、これは現行バッファー位置です。
42-44	X'11C2F4'	従業員番号 (Employee No) に必要な 6 桁を残しておくため、3 行目の 21 桁目に位置付けるための SBA シーケンス。「Tag No」ラベル・フィールドの始めは、ユーザーが長すぎる番号をキー入力しようとした場合にすぐ気付くように、従業員番号 (Employee No) 入力フィールドの終わりにマークされます。
45	X'1D'	フィールドを開始するための SF オーダー。
46	X'F0'	フィールド属性バイト: 自動スキップ、通常輝度、MDT オフ
47-55	' Tag No:'	表示データ。フィールド間のスペースを多くするためにラベルに 2 個の先行ブランクを付加します。(別個のフィールドを使用することもできますが、この方が文字を少し追加するだけなので容易です。)
56	X'29'	SFE (次のフィールドは別の入力フィールドです。ここはアウトラインを付けたいので、再び SFE を使用します)。
57	X'02'	属性タイプの数。

表 35. 3270 出力データ・ストリーム (続き)

バイト	内容	ノーツ
58-59	X'41F4'	下線の値を持つ拡張強調表示のコード。
60-61	X'C040'	フィールド属性および無保護、通常輝度、MDT オフの属性のコード。
62-64	X'11C3C7'	タグ用の 8 桁を残しておくために、3 行目の 40 桁目に位置付け直すための SBA シーケンス。
65	X'1D'	フィールドを開始するための SF。
66	X'F0'	フィールド属性バイト: 自動スキップ、通常輝度、MDT オフ
67-74	' State:'	フィールド・データ (スペーシングのために再び 2 個の先行ブランク)。
75-80	X'290241F4C040'	州 (State) 入力フィールドに関する SFE オーダーおよび属性指定 (属性はタグ入力フィールドの属性と同一です)。
81-82	X'0000'	州 (State) フィールドの (初期) 内容。他の入力フィールドの場合に省略したように、この値を省略することもできますが、現行バッファー位置をフィールドの終わりに移動するためには SBA シーケンスが必要で、この方が簡単です。
83	X'1D'	SF。最後のフィールドは、ユーザーが州コードに 2 文字より多くキー入力しようとしないように、直前のフィールドの終わりを示しています。この初期データはなく、属性バイトだけです。この種のフィールドを「ストッパー」フィールドと呼ぶことがあります。
84	X'F0'	フィールド属性バイト: 自動スキップ、通常輝度、MDT オフ

注: 端末制御コマンドを使用し、ユーザー独自のデータ・ストリームを構築する場合には、SEND コマンドの FROM パラメーターに指定するデータは、上の表のバイト 3 から始まります。CICS は、SEND コマンドのオプションから、書き込みコマンドと WCC を提供します。

3270 端末からの入力

以前に説明したように、キーボード入力はバッファーを介してホストに達します。3270 端末に使用可能なキーボード配置には異なるものがたくさんありますが、どんな配置の場合にも、キーは以下の 3 つのカテゴリーに分類されます。

- データ・キー
- キーボード制御キー
- アテンション・キー

データ・キー

データ・キーには、おなじみの文字、数字、句読記号、および特殊文字が含まれます。データ・キーを押すと、**カーソル**によって指示された時点のバッファーの内容 (したがって、画面) が変更されます。カーソルは、画面上 (つまり、バッファー内)

で、次のデータ・キー・ストロークが格納される位置を指す可視ポインターです。オペレーターがデータをキーによって入力したときに、カーソルは、形式設定されている画面上の、自動スキップ属性に定義されているフィールドをスキップして、画面上の次の位置に進みます。

キーボード制御キー

キーボード制御キーは、カーソルを新規位置に移動し、フィールドまたは個別バッファー位置を消去し、文字を挿入し、あるいはキーボードがバッファーを変更する場所または方法を変更します。

アテンション・キー

上記 2 つのグループ、すなわちデータ・キーおよびキーボード制御キーのどのキーも、ホストとの対話を引き起こしません。これら全体が装置およびその制御装置によって処理されます。これに反して、アテンション・キーは、バッファーがホストに伝送可能になっているというシグナルを発します。ホストが端末への読み取りを発行した場合には、CICS の通常の状態では、この時点で伝送が行われます。

アテンション・キーには、以下の 5 タイプがあります。

- ENTER
- PF (プログラム機能) キー
- CLEAR
- PA (プログラム・アテンション) キー
- CNCL (一部のキーボード・モデルにしか存在しない取り消しキー)

アテンション・キーを押す以外で、伝送を行わせるオペレーターの処置には、以下のものがあります。

- ID カード読み取り機構の使用
- 磁気スロット読み取り装置およびハンド・スキャナーの使用
- ライト・ペンまたはカーソル選択キーによるアテンション・フィールドの選択
- トリガー・フィールドの外側へのカーソルの移動

トリガー・フィールドの機能は一部の端末モデルで拡張属性によって提供されますが、上にリストされている他の処置のすべてが特殊なハードウェアを必要とし、ほとんどの場合には、画面 (バッファー) を前もって適切にセットアップしなければなりません。これらの機能は、877 ページの『第 71 章 特殊ハードウェアのサポート』で説明されています。このセクションでは、標準機能を中心に説明します。

AID

3270 は、インバウンド・データ・ストリームの最初のバイトのエンコードされた値によって伝送を行わせるキーを識別します。この値は、**アテンション ID** または **AID** と呼ばれます。

通常、端末オペレーターがデータを伝送するために選択するキーは、アプリケーション設計者によって指図されます。設計者は特定の意味を各種アテンション・キーに割り当て、ユーザーはアプリケーションを使用するためにこれらの意味を知っている必要があります。 (多くの場合、このように使用されるキーはいくつかしかありません。例えば、正常入力の場合の ENTER、アプリケーションの制御を終了する

ための 1 つの PF キー、部分的に完了したトランザクション・シーケンスを取り消すための別の PF キーなどです。選択項目がいくつかある場合には、ユーザーがそれを覚えておく必要がなくなるように、画面上にキー定義をリストしたくなることがあります。)

アテンション・キーの 2 つのグループの間には、アプリケーション設計者が留意する必要がある重要な違いがあります。ホストが「変更読み取り」コマンド (CICS によって普通に使用されるコマンド) を実行した時に、ENTER キーおよび PF キーはデータをバッファーから伝送します。しかし、CLEAR、CNCL、および PA キーでは、AID (すなわち、使用したキーの識別) は入手できますが、データは伝送されません。これらは、**短縮読み取り**キーと呼ばれます。これらのキーは、「取り消し」などの単純な要求を伝送する場合には役立ちますが、データをともなう要求を伝送する場合には役立ちません。実際に、多くの設計者は非データ要求にすら PF キーを使用し、それにともなうデータはすべて廃棄しています。

注: CLEAR キーには、送信するデータが文字通りなくなるように、バッファー全体をヌルに設定する追加の効果があります。また、CLEAR が画面サイズをデフォルト値に設定するのは、639 ページの『不定形式モード』 に説明されているように、その端末に代替画面サイズ機能があり、画面を不定形式モードにした場合です。

3270 端末からの読み取り

3270 には、以下の 2 つの基本読み取りコマンドがあります。

- **バッファー読み取り**
- **変更読み取り**

いずれかのコマンドに対して、インバウンド・データ・ストリームは、以下のような 3 バイトの**読み取りヘッダー**で始まります。

- 1 バイトのアテンション ID (AID)
- 2 バイトのカーソル・アドレス

直前のセクションの注のように、AID は伝送を行わせる処置またはアテンション・キーを示します。カーソル・アドレスは、伝送時点のカーソル位置を示しています。 CICS は、すべての RECEIVE コマンドの完了時に、この EIB の情報を EIBAID および EIBCPOSN に保管します。

バッファー読み取りコマンドは読み取りヘッダーに続きバッファー全体を入手し、位置を基礎にして必要な情報を抽出するのは受信プログラムの責任です。これは、主に診断および他の特殊目的のためのもので、CICS が RECEIVE コマンドの実行中にそれを使用するのは、BUFFER オプションが指定されている場合だけです。CICS が読み取りバッファーを使用して、非送信請求端末入力を読み取ることはないので、この方法で開始されたトランザクションの最初の RECEIVE で BUFFER オプションを使用することはできません。

CICS が普通に使用する変更読み取りコマンドによって、はるかに少ないデータが伝送されます。短縮読み取りキー (CLEAR、CNCL、および PA) の場合には、読み取りヘッダーしか到着しません。その他のアテンション・キー (ENTER および PF) の場合には、変更された画面上のフィールド (厳密にはオンの MDT を持つフィールド) が読み取りヘッダーに続いています。次のセクションでは、形式について説

明します。伝送が行われた原因がトリガー・フィールドか、ライト・ペンによる検出か、あるいはカーソル選択かによって、情報の量と形式がわずかに異なります。これらの特別形式については、 877 ページの『第 71 章 特殊ハードウェアのサポート』で説明しています。SCS プリンターにある**プログラム・アテンション**・キーからの入力も例外になります。該当するデータ・ストリームについては、 749 ページの『SCS 入力』を参照してください。

インバウンド・フィールド形式

次のいくつかのセクションでは、3270 が伝送するデータの形式について説明します。これは、端末制御コマンドを使用する場合に理解する必要があります。BMS を使用している場合には、BMS が入力を変換するので、 639 ページの『不定形式モード』にスキップすることができます。

各変更フィールドは、以下のとおりに到着します。

- SBA オーダー
- フィールドの最初のデータ 位置の 2 バイト・アドレス
- SF オーダー
- フィールドの内容

フィールド内の非ヌル文字だけしか伝送されません。ヌルが現れた場合には常にスキップされます。したがって、入力によってフィールドが埋め込まれず、そのフィールドが最初からヌルになっていた場合には、キー入力された文字だけが伝送され、インバウンド・データの長さが削減されます。ヌル (X'00') とブランク (X'40') は、画面上では区別できませんが、同じではありません。ブランクは伝送されるので、通常、フィールドはブランクではなくヌルで初期設定して、伝送を最小化します。

3270 読み取りコマンドでは、端末がフィールドの内容に沿って属性値を返すように指定することができますが、CICS はこのオプションを使用しません。したがって、バッファー・アドレスは、ではなく、(対応するアウトバウンド・データ・ストリームの場合のように) 先行する属性バイトフィールド・データの最初のバイトの位置です。

注: カーソル選択キー、トリガー・フィールド、磁気スロット読み取り装置などの、入力用 3270 の特殊機構は異なる入力形式を生成します。詳しくは、 880 ページの『フィールド選択の機能』を参照してください。

入力データ・ストリームの例

インバウンド・データ・ストリームを説明するために、 633 ページの図 126 に示されている画面を使用するオペレーターが以下の作業を行ったものとします。

- 従業員 ID フィールドに「123456」と入力する
- タグ番号に「ABC987」と入力する
- 州フィールドを埋め込まずに ENTER キーを押す

結果のインバウンド・データ・ストリームは以下のとおりです。

表 36. 3270 入力データ・ストリーム

バイト	内容	ノーツ
1	X'7D'	AID (この場合には ENTER キー)。
2-3	X'C3C5'	カーソル・アドレス: オペレーターが最終データ・キー・ストロークの後にとどまった 3 行目の 38 桁目。
4	X'11'	後にバッファー・アドレスが後に続くことを示す SBA。
5-6	X'C26E'	後に続くフィールドの開始位置を示す、3 行目の 15 桁目のアドレス。
7-12	'123456'	入力。オペレーターが入力した従業員番号。
13-15	X'11C3D1'	3 行目の 32 桁目のバッファー・アドレスを示す SBA シーケンス。
16	X'1D'	別の入力フィールドが続くことを示す SF。
17-22	'ABC987'	入力フィールド: プレート番号。オペレーターが残りの位置をヌルのままにしたため、8 文字のフィールドから到着するのが 6 文字だけであることを通知する。

3 番目の入力フィールド (州コード) は入力データ・ストリームには現れないことに注意してください。この理由は、その MDT がオンにされなかったためです。MDT は最初にオフにされており、オペレーターがフィールドにキー入力しなかったためにオンになりませんでした。また、CICS は通常、全変更読み取りを発行するため、バイト 7 では SF が不要であることにも注意してください。

不定形式モード

3270 の高機能はそのフィールド構造体を使用することを前提としているとしても、フィールドを使用しないで 3270 を使用することができます。これは、**不定形式モード**と呼ばれます。このモードでは、定義されているフィールドはなく、画面 (バッファー) 全体が、初期の単純な端末に非常によく似ていて、データの単一ストリング (インバウンドおよびアウトバウンド) として動作します。

不定形式モードで書き込む場合には、SBA オーダーを組み込んでデータを画面上の特定位置に指示することができるとしても、データの中にフィールドを定義しません。すべての SBA オーダーに先行するデータはカーソルの現在位置の始めに書き込まれます。(消去コマンドまたは書き込みコマンドを使用した場合には、カーソルはゼロ、すなわち画面の左上隅に自動的に設定されます。)

不定形式画面を読み取った場合には、形式設定画面を読み取った場合とまったく同様に、最初の 3 バイトは読み取りヘッダー (AID およびカーソル・アドレス) です。残りのバイトは、位置ゼロから始まるバッファー全体の内容です。フィールドはないので、SBA オーダーまたは SF オーダーは存在していません。読み取りコマンドが変更読み取りだった場合には、ヌルが抑制され、したがって、常に、画面上で入力データが見つかった場所を正確に判別することができるわけではありません。

BMS RECEIVE MAP コマンドを使用して不定形式画面を読み取ることはできません。BMS は、829 ページの『MAPFAIL およびその他の例外状態』 に説明されて

いるように、不定形式入力を検出すると MAPFAIL 条件を発生させます。不定形式データを読み取ることができるのは、CICS で端末管理 RECEIVE コマンドを使用する場合だけです。

注: CLEAR キーはバッファーをヌルに設定し、それによりフィールドを区切る属性バイトのすべてを消去するので、CLEAR キーは画面を不定形式モードにします。

第 9 部 インターバル制御およびタスク制御

タスクのタイミングとアクティビティーを制御します。

© Copyright IBM Corp. 1989, 2009

第 45 章 インターバル制御

CICS インターバル制御サービスは、時間に関連した機能を提供します。

Java および C++

ここで説明するアプリケーション・プログラミング・インターフェースは、Java プログラムでは使用されない EXEC CICS API です。JCICS クラスを使用してインターバル制御機能サービスにアクセスする Java プログラムについて詳しくは、「*Java Applications in CICS*」のJCICS クラス・ライブラリーおよび JCICS Javadoc 文書を参照してください。CICS C++ クラスを使用した C++ プログラムについて詳しくは、「*CICS C++ OO Class Libraries*」を参照してください。

インターバル制御コマンドを使用すると、以下のことができます。

- タスクを指定の時刻に開始するか、または指定のインターバルの経過後に開始し、データをタスクに渡す (START コマンド)。
- START コマンドで渡されたデータを取り出す (RETRIEVE コマンド)。
- タスクの処理を遅らせる (DELAY コマンド)。
- 指定した時間が経過したときの通知を要求する (POST コマンド)。
- イベントの発生を待つ (WAIT EVENT コマンド)。
- 直前のインターバル制御コマンドの効力を取り消す (CANCEL コマンド)。
- 現在の日付および時刻を要求する (ASKTIME コマンド)。
- 日付および時刻の形式を選択する (FORMATTIME コマンド)。21 世紀の日付を処理するのに役立つオプションが利用可能です。

注: リモート・トランザクションを開始するために、EXEC CICS START TRANSID() TERMID(EIBTRMID) を使用しないでください。代わりに EXEC CICS RETURN TRANSID() IMMEDIATE を使用してください。この方法で使用された START は、不必要に通信リソースをタイアップし、接続領域をまたがって性能低下になることがあります。

WAIT EVENT、START、WAIT オプションを使用した RETRIEVE、CANCEL、DELAY、または POST の各コマンドを使用する場合、動的トランザクション・ルーティングを実行する能力に悪影響を及ぼす、トランザクション間の類縁性が生じることがあります。

ISOLATE(YES) を指定した場合、WAIT EVENT の時刻イベント制御域用のストレージは、共用ストレージになければなりません。

CICS が実行中の場合、トランザクション分離機能があってもなくても、CICS は、時刻イベント制御域が読み取り専用ストレージにないことを検査します。

これらのコマンドを発行するプログラムにおいて発生する可能性のある問題の識別を容易にするため、CICS Interdependency Analyzer を使用できます。このユーティリティーについて詳しくは「*CICS Interdependency Analyzer for z/OS User's Guide*

© Copyright IBM Corp. 1989, 2009

and Reference」を参照し、トランザクションの類縁性について詳しくは 339 ページの『第 22 章 類縁性』を参照してください。

この章では、以下のことについて説明します。

- 『満了時刻』
- 645 ページの『要求 ID』

満了時刻

時間制御機能を開始する時刻は**満了時刻**です。(TIME オプションを使用して) 時刻として、または (INTERVAL オプションを使用して) 機能を実行する前に経過する間隔として、絶対値で満了時刻を指定することができます。DELAY コマンドには、FOR および UNTIL オプションを使用できます。POST および START コマンドには、AFTER および AT オプションを使用できます。

注: C および C++ 言語では、TIME および INTERVAL オプションによって使用されるパック 10 進タイプのサポートが提供されていません。

現在の時刻から指定された時間数、分数、および秒数でいつトランザクションを開始するかを、**間隔**を使用して CICS に通知します。ゼロ以外の INTERVAL 値は、現在時刻に指定した間隔を追加した、将来の時刻を常に指示します。時間は 0 から 99 に指定できますが、分および秒は 59 より大きくすることはできません。例えば、40 時間と 10 分でタスクを開始するためには、次のようにコーディングします。

```
EXEC CICS START INTERVAL(401000)
```

特定の時間にトランザクションを開始するよう CICS に通知するには、hhmmss を再び使用して**絶対時間**を使用します。例えば、午後 3:30 にトランザクションを開始するためには、次のようにコーディングします。

```
EXEC CICS START TIME(153000)
```

絶対時間は、常に、現在の時刻より前の午前 0 時を起点とするので、現在の時刻より前になることがあります。TIME は、コマンドが実行される時刻と相対して将来または過去とすることができます。CICS は以下の規則を使用します。

- タスクの開始時刻として指定した時刻が、過去 6 時間以内の場合、タスクは即時に始まります。これは、過去 6 時間に午前 0 時が含まれているかどうかに関係なく起こります。例:

  ```
  EXEC CICS START TIME(123000)
  ```

  このコマンドを月曜日の 05:00 または 07:00 に出した場合には、同じ日の 12:30 に満了します。

  ```
  EXEC CICS START TIME(020000)
  ```

  このコマンドを月曜日の 05:00 または 07:00 に出した場合には、指定した時刻が過去 6 時間以内に含まれているので、即時に満了します。

  ```
  EXEC CICS START TIME(003000)
  ```

このコマンドを月曜日の 05:00 に出した場合には、指定した時刻が過去 6 時間に含まれているので、即時に満了します。しかし、このコマンドを月曜日の 07:00 に出した場合は、指定した時刻が過去 6 時間以内に含まれていないので、火曜日の 00:30 に満了します。

```
EXEC CICS START TIME(230000)
```

このコマンドを月曜日の 02:00 に出した場合には、指定した時刻が過去 6 時間以内に含まれているで、即時に満了します。

- 時刻を指定するときに、その時分秒の中の時が 23 より大きい場合、現在日より後の日の時刻を指定することになります。例えば、250000 という時刻は現在日の翌日の 1 a.m. を意味し、490000 は翌々日の 1 a.m. を意味します。

DELAY、POST、または START コマンドで満了時刻または間隔オプションを指定しない場合には、CICS は即時を意味するデフォルトの INTERVAL(0) を使用して応答します。

システム間リンクの両端はそれぞれ、異なる時間帯に属していることがあるので、開始するトランザクションがリモート・システムにある場合は、INTERVAL 形式で満了時刻を指定してください。

システムで障害が発生した場合には、満了していない START コマンドと関連した時刻が、再始動を通じて記憶されています。

注:

1. 簡易に使用されるシステムでは、指定する間隔時間は 4 分の 1 秒を超えることがあります。
2. 満了時刻が起こりうる CICS シャットダウンの間にある場合には、タスクが実行を試みる前に CICS の状況をテストする必要があるかどうかを考慮しなければなりません。INQUIRE SYSTEM の CICSSTATUS オプションを使用して、これを実行できます。通常のシャットダウンの間、タスクは PLT プログラムと同じ時刻に実行することができ、その結果はユーザーごとに異なります。

要求 ID

要求およびそれに関連するデータを識別する手段として、CICS により、それぞれの DELAY、POST、および START の各コマンドに固有の要求 ID が割り当てられます。REQID オプションによってユーザー独自の要求 ID を指定することができます。指定しない場合には、CICS が固有の要求 ID を割り当て (POST および START コマンドの場合にのみ)、EXEC インターフェース・ブロック (EIB) の EIBREQID フィールドに入れます。CANCEL コマンドを使用して後から要求を取り消す場合には、要求 ID を指定する必要があります。

第 46 章 タスク制御

CICS タスク制御機能は、タスク・アクティビティーを同期化したりリソースの使用を制御したりする機能を提供します。

Java および C++

ここで説明するアプリケーション・プログラミング・インターフェースは、Java プログラムでは使用されない EXEC CICS API です。JCICS クラスを使用してタスク制御サービス CICS にアクセスする Java プログラムについて詳しくは、「*Java Applications in CICS*」のJCICS クラス・ライブラリーおよび JCICS Javadoc 文書を参照してください。CICS C++ クラスを使用した C++ プログラムについて詳しくは、「*CICS C++ OO Class Libraries*」を参照してください。

CICS は、CICS システム・プログラマーによって設定された値に基づいて、優先順位を割り当てます。プロセッサーの制御は、処理の準備ができているタスクのうち最高の優先順位のタスクに与えられ、CICS またはアプリケーション・プログラムによって実行される作業がこれ以上ない時に、オペレーティング・システムに返されます。

以下のことを行うことができます。

- タスクを中断 (SUSPEND コマンド) し、それより優先順位の高いタスクの処理を可能にします。これにより、プロセッサーを集中的に使用するタスクによってプロセッサーが占有されることを防ぐことができます。優先順位の高い他のタスクの処理が終了するか中断されると、元のタスクに制御権が返されます。すなわち、元のタスクはディスパッチ可能なままになっています。
- あるタスクによるリソースの使用をスケジュールします (ENQ および DEQ コマンド)。これは、1 つのリソースが複数のタスクによって同時に使用されることがないように保護するのに利用できる場合があります。すなわち、これは、リソースを逐次再使用可能にする方法です。リソースを使用する各タスクはエンキュー・コマンド (ENQ) を発行します。これを行う最初のタスクはすぐにそのリソースを使用できますが、HANDLE CONDITION ENQBUSY コマンドが発行されていない場合、別のタスクによって発行されたリソースに対する後続の ENQ コマンドでは、リソースが利用可能になるまでそのタスクが中断されることになります。

  ENQ コマンドに NOSUSPEND オプションをコーディングした場合、制御権は常にプログラムの次の命令に返されます。EIBRESP フィールドの内容を検査することによって、ENQ コマンドが正常に行われたかどうかがわかります。

  リソースを使用する各タスクは、そのリソースの使用が終了したら、デキュー・コマンド (DEQ) を発行する必要があります。ただし、エンキュー/デキューのメカニズムを使用する場合、複数のタスクが相互の関係においてある決まったシーケンスで ENQ コマンドおよび DEQ コマンドを発行することを保証する手段はありません。アクセス・シーケンスを制御する方法については、648 ページの『リソースへのアクセス・シーケンスの制御』を参照してください。

© Copyright IBM Corp. 1989, 2009

- タスクに割り当てられた優先順位を変更します (CHANGE TASK PRIORITY コマンド)。
- MVS 形式の ECB が完了したときにそれを通知するイベントを待機する。

  WAITCICS および WAIT EXTERNAL の 2 つのコマンドが利用可能です。これらのコマンドによって、このコマンドを出したタスクは、いずれかの ECB が通知されるまで、すなわち、いずれかのイベントが発生するまで中断されます。タスクは 1 つ以上の ECB について待機することができます。複数の ECB を待機する場合、タスクはいずれかの ECB が通知されるとただちにディスパッチ可能になります。ECB はそれぞれ、遅くとも通知される前までに、クリアしておく (すなわち 2 進ゼロにセットしておく) 必要があります。CICS が代わりに行うことはできません。以前に通知されていた ECB がその後クリアされていないと、その ECB で待機しても、ユーザー・タスクは中断されず、WAITCICS または WAIT EXTERNAL が発行されなかったかのように実行が継続されます。

  WAIT EXTERNAL は、通常オーバーヘッドは少ないほうですが、関連した ECB は、MVS POST 機能、または (比較交換 (CS) 命令を使用した) 最適化通知を使用して常に通知されなければなりません。他の方式で通知してはいけません。通知の方式について疑問がある場合には、WAITCICS コマンドを使用してください。WAIT EXTERNAL コマンドで渡された ECB を扱うとき、CICS は ECB を拡張し、MVS POST 出口機能を使用します。ある ECB は、一度に複数のタスクで待機してはいけません (1 つのタスクの ECBLIST に 2 回現れてはいけません)。この規則に従わないと、INVREQ 応答を返すことになります。

  ECB が、MVS POST 機能以外の他の方法によって、または最適化通知によって通知されることになっている場合は、WAITCICS が使用されなければなりません。例えば、アプリケーションが、ECB に値を移動することによって通知する場合は、WAITCICS を使用する必要があります。 (WAITCICS コマンドは、MVS POST 機能または最適化通知を使用して通知された ECB にも使用できます。) CICS は、MVS WAIT の状態になる場合に、WAITCICS コマンドを発行したタスクによって待機しているすべての ECB のリストを MVS に渡します。MVS WAIT で CICS によって渡された ECBLIST は、重複アドレスを含んでおり、MVS は CICS を異常終了します。

  MVS POST、WAIT EXTERNAL、WAITCICS、ENQ、または DEQ コマンドを使用すると、動的トランザクション・ルーティングを実行する能力に悪影響を及ぼす、トランザクション間の類縁性が生じることがあります。

  このコマンドを発行するプログラムにおいて発生する可能性のある問題の識別を容易にするため、CICS Interdependency Analyzer を使用できます。このユーティリティーについて詳しくは「*CICS Interdependency Analyzer for z/OS User's Guide and Reference*」を参照し、トランザクションの類縁性について詳しくは 339 ページの『第 22 章 類縁性』を参照してください。

リソースへのアクセス・シーケンスの制御

複数のタスクから特定の順序で 1 つのリソースにアクセスする場合は、ENQ および DEQ コマンドではなく、1 つ以上の WAITCICS コマンドと 1 つ以上の手動通知 ECB を組み合わせて使用してください。

ECB に手動通知するには、CICS タスクは、2 進ゼロのクリアされた状態、または X'40008000' の通知済みの状態に 4 バイトのフィールドをセットします。タスクは START コマンドを使用して、別のタスクを開始し、ECB のアドレスを渡すことができます。開始されたタスクは、RETRIEVE コマンドを使ってアドレスを受け取ります。

タスクは ECB を設定するか、またはそれで待機することができます。ECB を使用して、リソースにアクセスするタスクのシーケンスを制御します。必要な場合には、2 つのタスクが複数の ECB を共用することができます。この手法を、必要な数だけタスクを制御するように拡張することができます。

注: 一時点で、ある 1 つの ECB について待つことのできるタスクは 1 つだけです。

図 127 の例には、手動通知 ECB および WAITCICS コマンドを使用することにより、2 つのタスクが一時記憶キューを順次にアクセスする方法が示されています。

この例は、 650 ページの図 128 に示されているポインターによってアドレスされた 2 つの ECB (ECB1 と ECB2) を使用しています。

定理では、これらのタスクは、永遠に一時記憶キューを通してデータを交換することができます。実際には、何らかのコードを組み込んで、処理を正常にクローズする必要があります。

```
Task A                                          Task B
一時記憶キューを削除する
Clear ECB1 (set to X'00000000')
ECB2 をクリアする
EXEC CICS START task B ( pass addresses         EXEC CICS RETRIEVE
of PTR_ECB1_ADDR_LIST and                         (addresses passed)
PTR_ECB2_ADDR_LIST

LOOP:                                           LOOP:
 EXEC CICS WAITCICS                              Write to TS queue
  ECBLIST(PTR_ECB1_ADDR_LIST)                    Post ECB1 (set to X'40008000)
  NUMEVENTS(1)                                   EXEC CICS WAITCICS
 Clear ECB1                                       ECBLIST(PTR_ECB2_ADDR_LIST)
 Read TS queue                                    NUMEVENTS(1)
 Process data
 Delete TS queue
 Write to TS queue                               ClearECB2
 Post ECB2                                       Read TS queue
Go to start of loop                              Process data
                                                 Delete TS queue
                                                 Go to start of loop
```

図 127. WAITCICS を使用して共用リソースへのアクセスを制御する 2 つのタスク

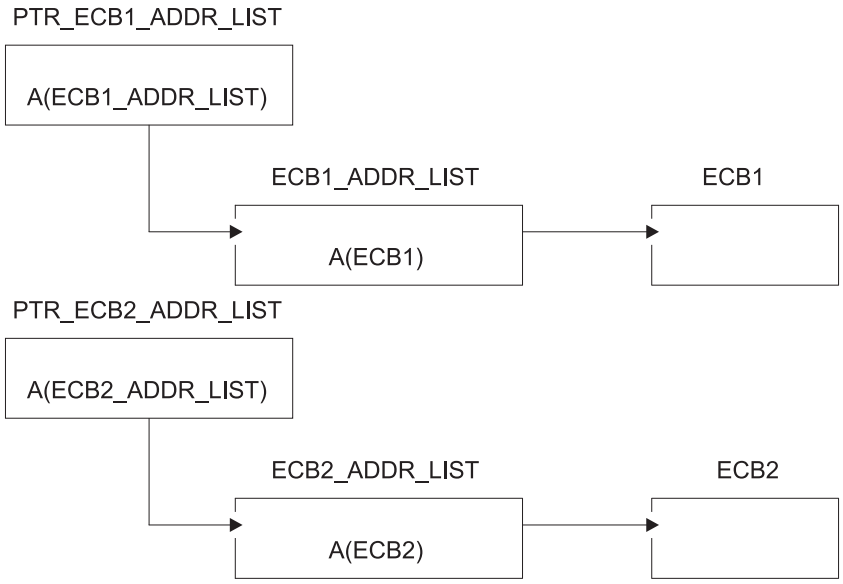


図 128. WAITCICS 例によって使用される ECB ポインター

383 ページの『第 24 章 例外条件の取り扱い』に、タスク制御コマンドの処理中に起こり得る例外条件の処理方法の説明があります。

第 10 部 ストレージ保護およびトランザクション分離

CICS コード、制御ブロック、およびその他のトランザクションがアプリケーション・プログラムによって上書きされないように保護します。

© Copyright IBM Corp. 1989, 2009

第 47 章 CICS のストレージ保護およびトランザクション分離

ストレージ管理は、ストレージ保護およびトランザクション分離に影響されます。ストレージ保護は CICS コードおよび制御ブロックをアプリケーションから保護し、トランザクション分離はそれぞれのタスクを別のタスクから保護します。

サブシステム・ストレージ保護機能は、CICS コードおよび制御ブロックが、ユーザーのアプリケーション・プログラムによって誤って上書きされることがないように機能します。CICS コードまたは制御ブロックの意図的な上書きに対する保護は提供し**ません**。CICS は、アプリケーションが CICS ストレージを修正するのに必要なアクセス (実行キー) を入手しないようにすることはできません。

トランザクション分離は、このストレージ保護を拡張して、トランザクション・データの保護を提供します。他のトランザクションのアプリケーション・プログラムによるトランザクション・データの意図しない上書きは、CICS システムの信頼性と可用性、およびシステム内のデータの保全性に影響することがあります。

ストレージ保護の使用は、オプションです。「*CICS System Definition Guide*」のに記載されている CICS システム初期設定パラメーターを介してストレージ保護機能を使用するかどうかを選択できます。

ストレージ制御

CICS ストレージ管理機能は、主記憶装置に対する要求を管理して、トランザクションの処理に必要な中間作業域およびその他の主記憶装置を提供します。

Java および C++

ここで説明するアプリケーション・プログラミング・インターフェースは、Java プログラムでは使用されない CICS API です。JCICS クラスを使用してストレージ管理サービスにアクセスする Java プログラムについて詳しくは、「*Java Applications in CICS*」のJCICS クラス・ライブラリーおよび JCICS Javadoc 文書を参照してください。CICS C++ クラスを使用した C++ プログラムについて詳しくは、「*CICS C++ OO Class Libraries*」を参照してください。

CICS は、アプリケーション・プログラムからの特定の要求なしで、各コマンド・レベル・プログラム内で作業用ストレージを自動的に使用可能にし、また、タスクの内部またはタスク間の両方の中間ストレージ用にその他の機能を提供します。 271 ページの『第 18 章 パフォーマンスの設計』 には、個々のプログラムにおけるストレージについて説明されています。ただし、CICS によって自動的に提供される作業用ストレージの他に作業用ストレージが必要な場合は、以下のコマンドを使用することができます。

- 主記憶装置を取得して初期設定する GETMAIN
- 主記憶装置を解放する FREEMAIN

© Copyright IBM Corp. 1989, 2009

GETMAIN コマンドで INITIMG オプションを指定することによって、獲得した主記憶装置を、例えば、ゼロまたは EBCDIC ブランクなど、任意のビット構成に初期設定することができます。

CICS は、タスクが正常終了または異常終了した時に、タスクと関連したすべての主記憶装置を解放します。これには、アプリケーション・プログラムによって獲得され、その後、解放されていないストレージが含まれますが、SHARED オプションで獲得された区域は例外です。GETMAIN コマンドのこの SHARED オプションは、ストレージがタスクの終了時に自動的に解放されないようにします。

SHARED オプションを指定して GETMAIN コマンドを使用し、FREEMAIN コマンドを使用した場合、動的トランザクション・ルーティングを実行する能力に不都合な影響を及ぼすトランザクション間の類縁性を生じることがあります。

これらのコマンドを発行するプログラムにおいて発生する可能性のある問題の識別を容易にするため、CICS Interdependency Analyzer を使用できます。このユーティリティーについて詳しくは「*CICS Interdependency Analyzer for z/OS User's Guide and Reference*」を参照し、トランザクションの類縁性について詳しくは 339 ページの『第 22 章 類縁性』を参照してください。

要求を発行した時に使用可能なストレージがない場合には、NOSUSPEND オプションを指定しない限り、CICS は、スペースが使用可能になるまでタスクを中断します。タスクが中断されている間に、トランザクション定義に SPURGE (YES) および DTIMOUT (mmss) が指定されている場合には、そのタスクが取り消される (タイムアウトになる) ことがあります。NOSUSPEND は、ストレージが利用不能の場合にはユーザー・プログラムに制御を返し、適切な代替処理を実行できるようにします。

このセクションでは、以下について説明します。

- 653 ページの『第 47 章 CICS のストレージ保護およびトランザクション分離』
- 657 ページの『アプリケーション用のストレージ・キーの定義』
- 661 ページの『実行およびストレージ・キーの選択』
- 666 ページの『トランザクション分離の使用』
- 668 ページの『MVS サブスペース』

ストレージ保護

CICS では、ユーザー・キー・ストレージ、CICS キー・ストレージのいずれにおいてもアプリケーション・プログラムを実行することができます (ユーザー・キーおよび CICS キーという用語の定義については、 655 ページの『ストレージのカテゴリー』を参照してください)。CICS ストレージは、ユーザー・キー・ストレージで実行するアプリケーション・プログラムによって上書きされないように自動的に保護されています (デフォルト)。CICS コードおよび制御ブロック (CICS 内部データ域) をユーザー・アプリケーション・プログラムから分離するという概念は、 655 ページの図 129 に示されています。

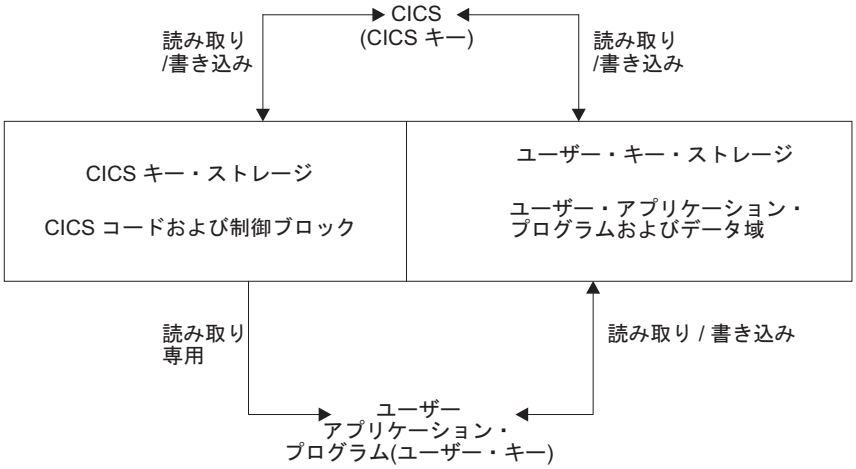


図 129. CICS コードおよび制御ブロックのユーザー・アプリケーション・プログラムからの保護

ストレージ・キーおよび実行キーに関連した図 129 内の用語は、『ストレージのカテゴリー』で説明しています。

ストレージのカテゴリー

ストレージ保護機能をアクティブにして実行しているとき、CICS はストレージを CICS キー・ストレージおよびユーザー・キー・ストレージという 2 つのカテゴリーに分けます。

CICS キー・ストレージ
CICS システム・コードおよび制御ブロック用に使用されます。また、インストール時の判断で、上書きの防止が必要な他のコードおよびデータ域に使用されます。

トランザクション分離がアクティブの CICS 領域では、CICS キー・プログラムは CICS キー・ストレージおよびユーザー・キー・ストレージへの読み取り / 書き込みアクセス権を持ちます。

ユーザー・キー・ストレージ
アプリケーション・プログラムおよびそれらのデータ域が通常存在するところです。

以下のように 2 つの関連した実行モードがあります。

1. CICS システム・プログラムは **CICS キー**で実行する。CICS キーで実行すると、プログラムは CICS キー・ストレージとユーザー・キー・ストレージの両方への読み取り / 書き込みアクセスが可能になります。
2. アプリケーション・プログラムは、通常**ユーザー・キー**で実行する。ユーザー・キーで実行すると、プログラムは、ユーザー・キー・ストレージに対しては読み取り / 書き込みアクセスが可能になりますが、CICS キー・ストレージに対しては読み取りアクセスだけが可能になります。

「ユーザー・キー」および「CICS キー」という用語は、このように、ストレージとそのストレージに関連したプログラムの実行の両方に適用されます。これらのことは、TRANSACTION 定義で使用されるリソース定義キーワードにも反映します。詳しくは「*CICS Resource Definition Guide*」の、 を参照してください。

実行キーは、ユーザーのアプリケーション・プログラムが持っている、CICS キー・ストレージへのアクセスのタイプを制御します。デフォルトでは、アプリケーション・プログラムにユーザー・キーでの制御が与えられています。CICS キーは、CICS キーで実行する必要があるプログラムのためだけに定義してください。CICS キーでの実行を選択するプログラムは、通常システム・プログラマーによって作成されたもので、通常はユーザー・アプリケーションのサポートにおいて特別の機能を提供するために設計されたものです。このようなプログラムは、一般にアプリケーションの一部というよりは CICS の拡張として考えられています。このようなプログラムのいくつかの例が、663 ページの『CICS キーのアプリケーション』 に記載されています。

ストレージ保護機能は、このタイプのプログラムによって、または CICS キーでの実行をユーザーが選択する通常アプリケーション・プログラムによって上書きされないように、CICS コードおよび制御ブロックを保護するものではありません。

実行キーの定義

CICS キーでプログラムを実行するには、プログラム・リソース定義で実行キー・パラメーター (EXECKEY) を使用してください。EXECKEY の説明については、661 ページの『実行およびストレージ・キーの選択』を参照してください。EXECKEY パラメーターは、CICS がアプリケーション・プログラムに制御を渡す際のキーを判別します。

トランザクション分離

トランザクション分離は、MVS サブスペース・グループ機能を使用して、トランザクション間の保護を提供します。このことによって、1 つのトランザクションに関連したアプリケーション・プログラムが別のトランザクションのデータを誤って上書きすることが絶対にないようにします。

トランザクション分離の利点、および関連したサポートは以下のとおりです。

- システム障害の縮小
- アプリケーション・データの保護
- 無効なアドレスを渡すアプリケーション・プログラムからの CICS の保護
- アプリケーション開発の補助

システム障害の縮小

トランザクション分離は、ユーザー・キー・トランザクションのストレージが誤って上書きされてしまうというユーザー・キー・アプリケーション・プログラムでのコーディング・エラーによって起こる、データ破壊と計画外の CICS システムの停止が起こらないようにします。誤ってトランザクション・データを上書きしないようにすることは、CICS 領域の信頼性と可用性を高めるという点で非常に重要です。

アプリケーション・データの保護

アプリケーション・プログラムが CICS コードまたはデータを上書きすると、結果的に CICS は障害を起こします。アプリケーション・プログラムが別のアプリケーション・プログラムの**コード**を上書きすると、その上書きされたアプリケーション・プログラムに障害が起こる恐れがあります。このことは実動領域では重大な中

断であるのに、その影響は即時に現れ、一般にプログラムは端末ユーザーが障害のあるトランザクションを再試行できるようにリカバリーできます。しかし、1 つのトランザクションのアプリケーション・プログラムが別の アプリケーション・プログラムの**データ**を上書きした場合、多くの場合、その結果は即時に現れません。エラー・データがデータベースに書き込まれる可能性があり、エラーが後々まで見逃されたままになって、エラーの原因を調べるのが不可能になってしまうかもしれません。データ上書きの結果は、しばしばコードの上書きよりはるかに重大な問題です。

無効なアドレスを渡されることからの CICS の保護

CICS はまた、結果的に CICS が記憶保護違反を起こす原因となる無効アドレスを渡すアプリケーションから、自身を保護します。このことは、アプリケーション・プログラムが、自分が所有していないストレージを自分に代わって CICS に修正させる EXEC CICS コマンドを発行すると生じます。以前のリリースでは、CICS は渡されたアドレスによって参照されるストレージの所有権を検査しなかったので、このようなコマンドを実行して記憶保護違反が起こりました。

CICS はストレージの開始アドレスの妥当性検査をし、コマンドを実行する前に、そのアプリケーション・プログラムがそのアドレスで始まるストレージに書き込みアクセスできることを確認します。

このアドレス検査は CMDPROT システム初期化パラメーターを使って制御されます。プログラムが CICS に API の出力フィールドとして無効なアドレスを渡すと、AEYD 異常終了が生じます。これは、ストレージ保護およびトランザクション分離とは完全に独立しています。

アプリケーション開発の補助

トランザクション分離は、テストおよびデバッグの段階でアプリケーション開発を補助します。アプリケーションが CICS や他のアプリケーションに上書きしようとすると、あるいは、CICS の書き込み用に持っているのではないストレージ・アドレスを渡そうとすると、CICS は即時にタスクを異常終了し、上書きしようとした問題のプログラムの名前とアドレスを報告します。このことによって、アプリケーション開発環境における共通の問題をデバッグするための時間が大幅に短縮されます。

アプリケーション用のストレージ・キーの定義

このタスクについて

CICS では、アプリケーションが使用可能な多数の CICS データ域およびアプリケーション・プログラム・データ域のために、ユーザー・キー・ストレージと CICS キー・ストレージのどちらかを選択できます。データ域に合わせて、以下のものを使用してストレージ・キーを選択します。

- システム初期設定パラメーター
- リソース定義オプション
- GETMAIN コマンドでのオプションの選択

アプリケーションがアクセスする必要のあるストレージ域用のストレージ・キーの定義については、次のセクションで説明しています。

システム全体のストレージ域

各 CICS 領域で、共通作業域 (CWA) および端末管理テーブル・ユーザー域 (TCTUA) に対して、ユーザー・キー・ストレージか CICS キー・ストレージのいずれかを、ご使用のシステムが選択できます。これらの区域がユーザー・キー・ストレージにある場合は、すべてのプログラムがこれらの区域への読み取り / 書き込みアクセス権を持ちます。CICS キー・ストレージにある場合は、ユーザー・キー・アプリケーション・プログラムは読み取り専用アクセスに制限されます。CWA と TCTUA 用のストレージ・キーは、それぞれ、システム初期設定パラメーターの CWAKEY と TCTUAKEY によって設定されます。どちらの場合も、デフォルト・オプションでは CICS がユーザー・キー・ストレージを入手します。

上記、およびその他のストレージ保護関連のシステム初期設定パラメーターの指定方法について詳しくは、「*CICS System Definition Guide*」のCWAKEYを参照してください。

タスク存続期間ストレージ

トランザクション開始時に CICS が獲得するストレージのために、またトランザクションの個々のアプリケーション・プログラムに直接関連したストレージのこれらのエレメントのために、ユーザー・キー・ストレージと CICS キー・ストレージのどちらを使用するかを指定することもできます。トランザクション・リソース定義で TASKDATAKEY オプションを使用してこれを指定します。これにより、以下のストレージ域に割り振られたストレージ域のタイプが決定されます。

- トランザクション作業域 (TWA) と EXEC インターフェース・ブロック (EIB)
- アプリケーション・プログラムのそれぞれの実行のために CICS が入手する作業用ストレージのコピー
- 以下のものの応答としてアプリケーション・プログラムのために獲得した任意のストレージ
  - GETMAIN コマンドを使用した明示的なストレージ要求
  - SET オプションを使用した CICS コマンドの結果としての暗黙的なストレージ要求

TASKDATAKEY パラメーターの指定方法について詳しくは、「*CICS Resource Definition Guide*」のTRANSACTION 定義属性を参照してください。

660 ページの図 130 は、TASKDATAKEY がタスク存続時間ストレージおよび作業用ストレージの両方のために制御することを示しています。

EXEC CICS コマンドに関するプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

出口および PLT プログラム専用のプログラム作業用ストレージ

CICS は、呼び出し側トランザクションの TASKDATAKEY オプションを使用して、グローバル・ユーザー出口、タスク関連ユーザー出口、ユーザー置換可能モジュール、および PLT プログラムのために獲得するストレージ用のストレージ・キーを判別します。さまざまなタイプのプログラムに対する影響の詳細など、ストレージ・キーに関するプログラミング情報については、「*CICS Customization Guide*」のを参照してください。

COMMAREA によるデータの引き渡し

疑似会話型アプリケーションでは、CICS は、RETURN コマンドで指定した COMMAREA を、読み取り / 書き込みモードで会話の次のプログラムへ常にアクセス可能にします。複数のプログラム (LINK または XCTL 使用) を含むトランザクション内で COMMAREA を渡すときにも同様です。CICS は、ターゲット・プログラムが COMMAREA への読み取り / 書き込みアクセスを確実に持つようにします。

GETMAIN コマンド

GETMAIN コマンドは、関連したトランザクション・リソース定義で指定された TASKDATAKEY オプションに関係なく、アプリケーション・プログラムがユーザー・キー・ストレージまたは CICS キー・ストレージを明示的に要求できるように、USERDATAKEY および CICSDATAKEY オプションを提供します。

例えば、このオプションによって、TASKDATAKEY(CICS) を指定して実行しているアプリケーション・プログラムが、ユーザー・キーで実行するプログラムに渡すため、あるいは戻すために、ユーザー・キー・ストレージを入手できます。

EXECKEY(CICS) で定義されたプログラム内で発行した GETMAIN コマンドによって取得された CICS キー・ストレージは、FREEMAIN コマンドが EXECKEY(CICS) で定義されたプログラムによって発行される場合にのみ明示的に解放されます。EXECKEY(USER) で定義されたアプリケーション・プログラムが、FREEMAIN コマンドを使用して CICS キー・ストレージを解放しようとすると、CICS が INVREQ 条件を返します。ただし、アプリケーションは EXECKEY オプションに関係なく、FREEMAIN コマンドによってユーザー・キー・ストレージを解放することができます。

アプリケーションによって獲得されたすべてのタスク存続時間ストレージは、CICS キーであってもユーザー・キーであっても、タスク終了時に CICS によって解放されます。関連したトランザクション・リソース定義のこのオプションに、STORAGECLEAR(YES) を指定することもできます。これによってストレージがクリアされ、誤って別のタスクが重要データを参照することを防ぎます。

各コマンドのプログラミング情報については、「*CICS アプリケーション・プログラミング・リファレンス*」、リソース定義の情報については、「*CICS Resource Definition Guide*」を参照してください。

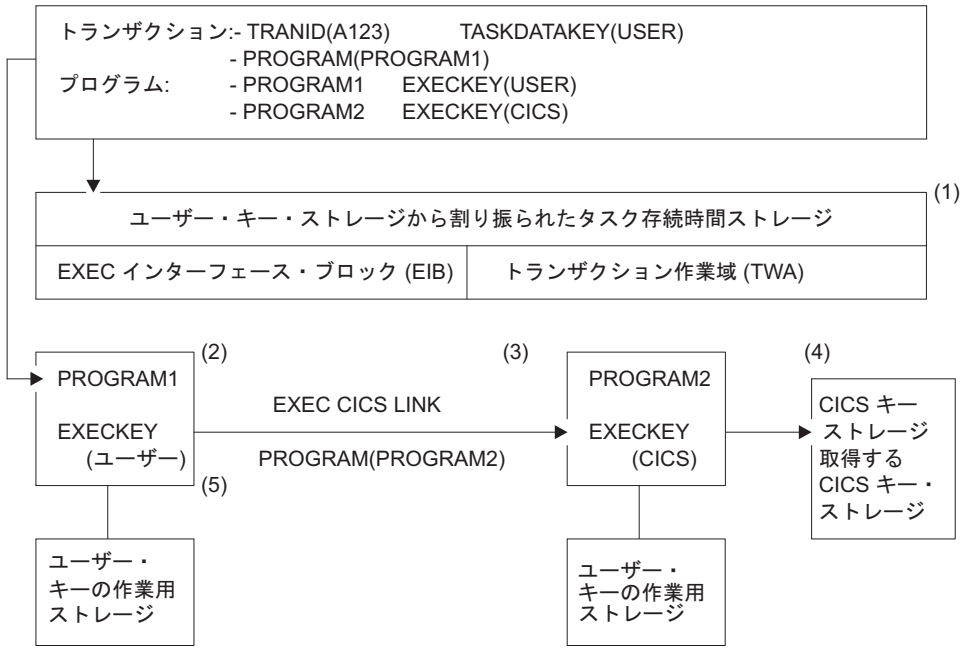


図 130. TASKDATAKEY および EXECKEY オプションの使用の図

注:

1. TASKDATAKEY オプションは、EXECKEY(USER) で指定されるユーザー・キーで実行する PROGRAM1 に必要な TWA および EIB が、ユーザー・キー・ストレージから確実に割り振られるようにします。
2. PROGRAM1 は (EXECKEY によって制御される) ユーザー・キーで実行され、ユーザー・キー・ストレージの中で獲得した作業用ストレージを (TASKDATAKEY オプションによって制御される) 持っています。GETMAIN コマンドによって、または CICS コマンド上の SET オプションを使用することによって、プログラムが取得するその他のストレージもまたユーザー・キー・ストレージの中で取得されます。
3. PROGRAM2 は、(EXECKEY で制御される) CICS キーで実行されますが、これもまた TASKDATAKEY によって制御されるユーザー・キー・ストレージで取得した作業用ストレージを持っています。
4. PROGRAM2 は、CICSDATAKEY オプションを使用して明示的な GETMAIN コマンドを発行します。また、これは CICS キーで実行されるので、制御を PROGRAM1 に戻す前に、データを CICS キー保護ストレージに保管することができます。
5. PROGRAM1 は、PROGRAM2 が獲得した CICS キー保護ストレージに書き込むことはできませんが、PROGRAM2 がそこに書き込んだものを読み取ることができます。

   EXECKEY(CICS) および TASKDATAKEY(CICS) を指定する必要があるかどうかを決めるとき、これらのオプションを必要とするすべての理由を考慮しなければなりません。

ストレージ保護キーを変更するプログラムは、ストレージへのアクセスを試行する際に正しいキーで実行されていることを確認しなければなりません。CICS はプログラムを呼び出す際に、プログラム定義で定義されている EXECKEY しか使用することができません。

実行およびストレージ・キーの選択

このタスクについて

ストレージ保護を使用して CICS を実行中であるとき、ユーザーのアプリケーション・プログラムの大多数は、すべてのストレージをユーザー・キーで取得し、ユーザー・キーで実行しなければなりません。ユーザー・キーでは許可されていない機能を使用しているプログラム、または特別な「システム・タイプ」のトランザクションまたはベンダー・パッケージに対してのみ、プログラム定義で EXECKEY(CICS) を定義し、関連したトランザクション定義で TASKDATAKEY(CICS) を定義する必要があります。

すべてのコンポーネント・プログラムが EXECKEY(CICS) を持っているトランザクションに対してと、ユーザーが自分のタスク存続時間ストレージおよび作業用ストレージをユーザー・キー・アプリケーションによって上書きされないように保護したいトランザクションに対してのみ、TASKDATAKEY(CICS) を指定すべきです。例えば、CEDF などの CICS 提供のトランザクションを TASKDATAKEY(CICS) で定義します。

TASKDATAKEY(CICS) で定義されたトランザクションの一部を構成するプログラム上では、EXECKEY(USER) を指定できないということに注意してください。なぜなら、このような状態ではユーザー・キー・プログラムは自分の作業用ストレージに書き込みをすることができないからです。TASKDATAKEY(CICS) で定義されているトランザクション内で、EXECKEY(USER) で定義されているプログラムがあると、トランザクションは、ストレージ保護がアクティブかどうかには関係なく、AEZD で異常終了します。

呼び出し側の実行キーを継承するように、プログラムを定義することはできません。実行キーおよびデータ・ストレージ・キーは、各プログラムごとに、そのプログラム・リソース定義および関連するトランザクション・リソース定義からそれぞれ得られます。これは、ユーザーが明示的に指定するか、またはデフォルトを受け入れます。デフォルトは、常にユーザー・キーです。表 37 には、さまざまなオプションの組み合わせが概説されています。

表 37. KEY オプションの組み合わせ

EXECKEY	TASKDATAKEY	推奨される使用法と説明
USER	USER	CICS API を使用した通常のアプリケーション用
USER	CICS	**許可されていない。**CICS は TASKDATAKEY(CICS) で定義されているトランザクションのもとで呼び出された、EXECKEY(USER) で定義されたプログラムはすべて異常終了させます。

表 37. KEY オプションの組み合わせ (続き)

EXECKEY	TASKDATAKEY	推奨される使用法と説明
CICS	USER	制限付き MVS 要求の発行、または CICS キー・ストレージの修正が必要なプログラム用。
CICS	CICS	CICS 提供トランザクションなど CICS の拡張として機能するか、または同様の保護を必要とするトランザクション (およびコンポーネント・プログラム) 用。

ユーザー・キーのアプリケーション

ほとんどのアプリケーションに対して、EXECKEY(USER) でプログラムを定義して、TASKDATAKEY(USER) で関連のトランザクションを定義してください。CICS ストレージ保護機能を最大限に利用するには、ユーザー・キー・ストレージでアプリケーション・プログラムを実行することをお勧めします。これらのオプションで USER を指定すると、以下の影響があります。

EXECKEY(USER)

プログラムが呼び出されたときに、CICS はそのプログラムにユーザー・キーで制御を与えることを指定します。EXECKEY (USER) で定義されたプログラムは、CICS キー・ストレージへのアクセスは読み取り専用に制限されています。以下のことが含まれます。

- CICS 自体に属しているストレージ
- TASKDATAKEY(CICS) で定義されたユーザー・トランザクションに属す CICS キー・ストレージ
- EXECKEY(CICS) で定義され、そのため CICS キー・ストレージにロードされたアプリケーション・プログラム
- トランザクション分離がアクティブである CICS 領域では、ユーザー・キー・プログラムは、自分自身のトランザクションのユーザー・キー・タスク存続時間ストレージおよび共用 DSA ストレージへの読み取り / 書き込みアクセスを持っています。

TASKDATAKEY(USER)

トランザクション作業域 (TWA)、EXEC インターフェース・ブロック (EIB) などのすべてのタスク存続時間ストレージが、ユーザー・キー・ストレージから入手されることを指定します。

このことはまた、トランザクション内のプログラムに直接関連したすべてのストレージが、ユーザー・キー・ストレージから入手されるということを意味します。

ただし、ISOLATE(YES) で定義されたトランザクションのユーザー・キー・プログラムには、自分自身のタスクのユーザー・キー・タスク存続時間ストレージへのアクセスしかありません。

USER は EXECKEY および TASKDATAKEY オプションの両方のデフォルトです。したがって、既存のアプリケーション・プログラム用のリソース定義を変更する必要はまったくありません。

CICS キーのアプリケーション

ほとんどのアプリケーション・プログラムは、デフォルト値である EXECKEY(USER) で定義できます。これは大多数の場合に使用が推奨されているオプションです。これには DL/I または DB2 を使用するプログラム、およびリソース・マネージャー・インターフェース (RMI) または LINK コマンドを通じてベンダー製品にアクセスするプログラムが含まれます。

ただし、アプリケーション・プログラムの中には、後でリストする特定の機能を使用する必要があるために、EXECKEY(CICS) で定義する必要があるアプリケーション・プログラムもあります。EXECKEY(CICS) を広く使用すると、CICS キーで実行するアプリケーション・プログラムによって CICS コードおよび制御ブロックが上書きされないように保護されていないので、ストレージ保護機構による保護が弱まります。トランザクション定義の ISOLATE 属性は、CICS キーで実行するアプリケーション・プログラムに対して、(つまり EXECKEY(CICS) で定義されたプログラムからの) 保護を一切提供しません。EXECKEY(USER) で定義されたときに記憶保護例外を起こすアプリケーション・プログラムはすべて、修正が許可されていないストレージを修正しようとしている理由を判別するための検査を受けなければなりません。アプリケーション・プログラムが以下に記載する機能を正しく使用することが確認できている場合にのみ、EXECKEY(CICS) にプログラム定義の変更をしてください。

- プログラムは、CICS API を介してよりむしろ MVS マクロまたはサービスを直接使用します。ユーザー・キー・プログラムでサポートされている MVS マクロは、SPIE、ESPIE、POST、WAIT、WTO、および WTOR だけです。EXECKEY(USER) プログラムから GTF トレース要求を発行することも可能です。プログラムが他の MVS マクロまたはサービスを使用する場合は、EXECKEY(CICS) で定義されなければなりません。以下は特別な例です。
  - 動的割振り (DYNALLOC マクロ、SVC 99) の使用
  - MVS GETMAIN および FREEMAIN または STORAGE 要求の使用
  - MVS OPEN、CLOSE、またはその他のファイル・アクセス要求の使用

  MVS マクロおよびサービスには、EXECKEY(CICS) を定義した CICS アプリケーションの中でさえ、直接使用するのが望ましくないものがあります。それは、要求が満たされるまで、MVS が CICS 領域全体を中断させる原因となるかもしれないからです。

  COBOL、PL/I、C、および C++ の言語ステートメントおよびコンパイラー・オプションには、オペレーティング・システム機能を呼び出すものがあります。これらのうち CICS アプリケーション・プログラムで使用できないものについて詳しくは、25 ページの『第 3 章 COBOL でのプログラミング』、53 ページの『第 4 章 C および C++ でのプログラミング』、および 65 ページの『第 5 章 PL/I でのプログラミング』を参照してください。これらの機能のあるものが CICS の以前のリリースで作用をしたかもしれない、あるいは少なくともアプリケーションが障害を起こさなかったかもしれないという可能性があります。プログラムが EXECKEY(USER) で定義されているときには、それらは機能**しません**。禁止されているオプションまたはステートメントの使用が記憶保護例外の原因であるとき、プログラムを EXECKEY(CICS) で 再定義するよりむしろ、プログラムから

これらのオプションまたはステートメントを除去してください。禁止されているステートメントおよびオプションの使用は、CICS の実行全体に副次作用を及ぼすことがあるので、除去してください。

- プログラムが CWA を修正する必要があり、CWA は CICS キー・ストレージにある (CWAKEY=CICS)。

  CWAKEY(CICS) を指定することによって CWA を保護することにした場合、CWA を修正することが許可されているプログラムは、できるだけ少なく、あるいは 1 つだけに制限してください。保護された CWA へのアクセスの制御方法について詳しくは、281 ページの『共通作業域 (CWA) の使用』を参照してください。
- プログラムが TCTUA を修正する必要があり、TCTUA は CICS キー・ストレージにある (TCTUAKEY=CICS)。

  ストレージ保護環境での TCTUA の使用に関する詳細については、285 ページの『TCTTE ユーザー域 (TCTUA) の使用』を参照してください。
- プログラムは、PLT プログラムから、TASKDATAKEY(CICS) を定義したトランザクションから、タスク関連あるいはグローバル・ユーザー出口プログラムから、またはユーザー置換可能プログラムから呼び出すことができます。
- プログラムが、CICS 制御ブロックを修正する。(例えば、CICS 制御ブロックを操作する必要のあるベンダー製品など。) これらは EXECKEY(CICS) で定義されなければなりません。
- プログラムが、CICS に対するユーザー拡張を提供し、CICS システム・コードと同じように、保護およびデータ・アクセスを必要とする。例えば、このようなプログラムは、ユーザーの CICS システムの重要な一部分であり、関連したストレージは (CICS ストレージのように) 通常のアプリケーションから保護される必要があります。
- CICS は、プログラム・リソース定義で指定されているオプションに関係なく、常に CICS キーで以下のタイプのユーザー作成プログラムに制御を与える。
  - グローバル・ユーザー出口 (GLUE)
  - タスク関連ユーザー出口 (TRUE)
  - ユーザー置換可能モジュール (URM)
  - プログラム・リスト・テーブル (PLT) プログラム

  CICS は、制御が PLT プログラム、グローバル・ユーザー出口やタスク関連ユーザー出口、またはユーザー置換可能プログラムに渡されるとき、そのプログラム・リソース定義で指定された EXECKEY に関係なく、最初に呼び出されたプログラムは必ず CICS キーで実行されるようにします。ただし、この最初のプログラムが他のプログラムに対して LINK または XCTL を発行すると、これらのプログラムは、そのプログラム定義で指定されたキーで実行します。これらの次のプログラムで CICS キー・データ域への書き込みが必要な場合、このような状態はよく起こることですが、そのプログラムは EXECKEY(CICS) として定義されなければなりません。

  トランザクション分離がアクティブの CICS 領域では、CICS が出口プログラムに制御を与えるときの現行モードによって、基本スペースまたはサブスペース (668 ページの『MVS サブスペース』 を参照) のいずれかでこれらの TRUE お

よび GLUE が実行されます。これらの出口は、いかなるアプリケーション・ストレージでも修正することができます。 URM および PLT プログラムは基本スペースで実行されます。

ストレージ保護で実行中の CICS 領域における GLUE、TRUE、URM、および PLT の各プログラムの実行に関するプログラミング情報については、「*CICS Customization Guide*」の出口プログラムおよび CICS ストレージ保護機能を参照してください。

2 つのトランザクションがタスク存続時間ストレージを共用するという利点によって類縁性を持つ場合、トランザクションが ISOLATE(NO) として定義されるか、またはプログラムが EXECKEY(CICS) として定義されなければなりません。CICS Interdependency Analyzer を使用すると、トランザクション類縁性の原因を検査できます。このユーティリティーについて詳しくは、「*CICS Interdependency Analyzer for z/OS User's Guide and Reference*」を参照してください。CICS システム・コードおよびデータは保護されたままなので、これらのオプションの最初のものが推奨のオプションです。

テーブル

実行可能プログラムの他にも、プログラム・リソースとして、テーブル、マップ・セット、および区分セットを定義することができます。これらのオブジェクトは実行されないので、EXECKEY がこれらのオブジェクトに与える影響は少なくなります。ただし、EXECKEY は実行不可能なオブジェクトがどこにロードされるかを制御し、そのため、他のプログラムがその中に保管できるかどうかに影響を及ぼします。

マップ・セットおよび区分セット

マップ・セットは再入可能ではありません (絶対画面位置を計算するときは、BMS 自身がマップ中のフィールドを更新します)。ただし、マップ・セットはアプリケーション・プログラムによって修正してはいけません。つまり、常に CICS キーで実行される CICS によってのみ修正されなければなりません。CICS は常にマップ・セットと区分セットを CICS キー・ストレージにロードします。

ストレージ保護の例外条件

ユーザー・キーで実行しているアプリケーション・プログラムが CICS キー・ストレージを変更しようとすると、記憶保護例外が生じます。記憶保護例外は通常の CICS プログラム・エラー処理によって処理され、問題のトランザクションは ASRA で異常終了します。例外条件は、他の保護ストレージを参照しようとした時と同じようにトランザクションに現れます。CICS エラー処理は、参照が CICS キーの動的ストレージ域 (DSA) に対するものであるかどうかを検査して、コンソールにメッセージ (DFHSR0622) を送信します。それ以外の場合は、CICS は他のどの ASRA 異常終了とも同じようにこの障害を処理します。ストレージ保護の例外条件について詳しくは、「*CICS Problem Determination Guide*」のトランザクション異常終了コードの処理を参照してください。

トランザクション分離の使用

このタスクについて

トランザクション分離はストレージ保護の一番上に構築されます。つまり、STGPROT=YES が指定されなければならないということです。トランザクション分離はストレージ保護によって導入されたパラメーター、EXECKEY および TASKDATAKEY を利用します。

ユーザー・トランザクション用にストレージと実行キーを指定できるということの他に、トランザクション分離が必要かどうかも指定できます。TRANISO システム初期設定パラメーターを使って、CICS 領域全体にわたってグローバルにトランザクション分離を制御することができます。個々のトランザクションのためには、トランザクション・リソース定義の ISOLATE オプションによって、ユーザーがそれぞれのトランザクションとプログラムに適用すべき保護のレベルを指定できます。

```
ISOLATE [YES or NO]
```

これらのオプションのデフォルトは、ほとんどの場合、既存のアプリケーションには、リソース定義への変更が必要ないことを意味します。ただし、必要があれば、デフォルトである全保護の基準に合わない場合でもトランザクションが継続して機能できるように、保護が調整できます。これは、トランザクションのユーザー・キー・タスク存続時間ストレージが他のトランザクションのユーザー・キー・プログラムからは保護されているが、CICS キー・プログラムからは保護されていないということを意味します。これについては、図 131を参照してください。

トランザクション A (TXNA) によって呼び出されたユーザー・キー・プログラムは、TXNA のユーザー・キー・タスク存続時間ストレージ、および共用ユーザー・ストレージに対して、読み取りおよび書き込みをすることがあります。さらに、TXNA にはトランザクション B (TXNB) のユーザー・キー・タスク存続時間ストレージへのアクセスがありません。

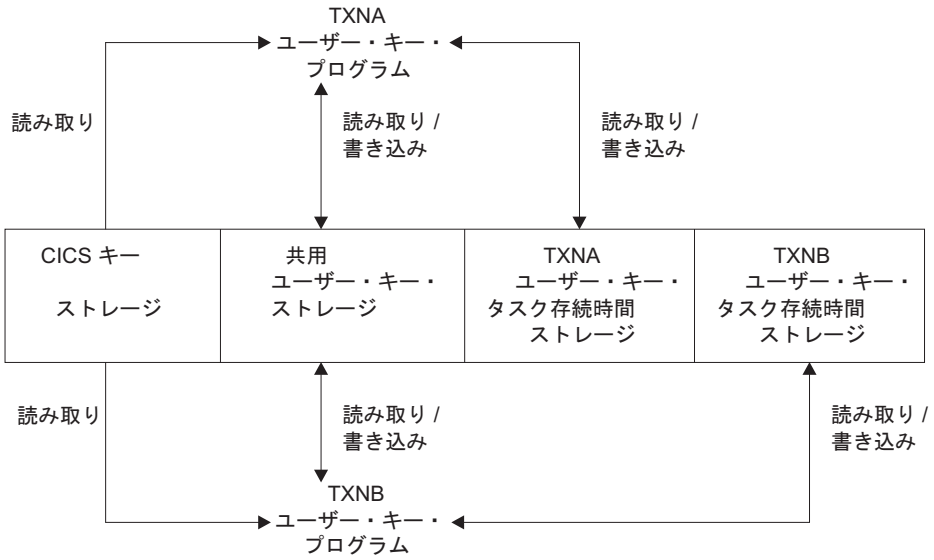


図 131. ISOLATE(YES) として定義された 2 つのトランザクション

トランザクションが ISOLATE(NO) として定義されている場合、そのユーザー・キー・タスク存続時間ストレージは、ISOLATE(NO) として定義された他のすべてのトランザクションにとって可視です。ただし、ISOLATE(YES) として定義されたトランザクションからは保護されています。

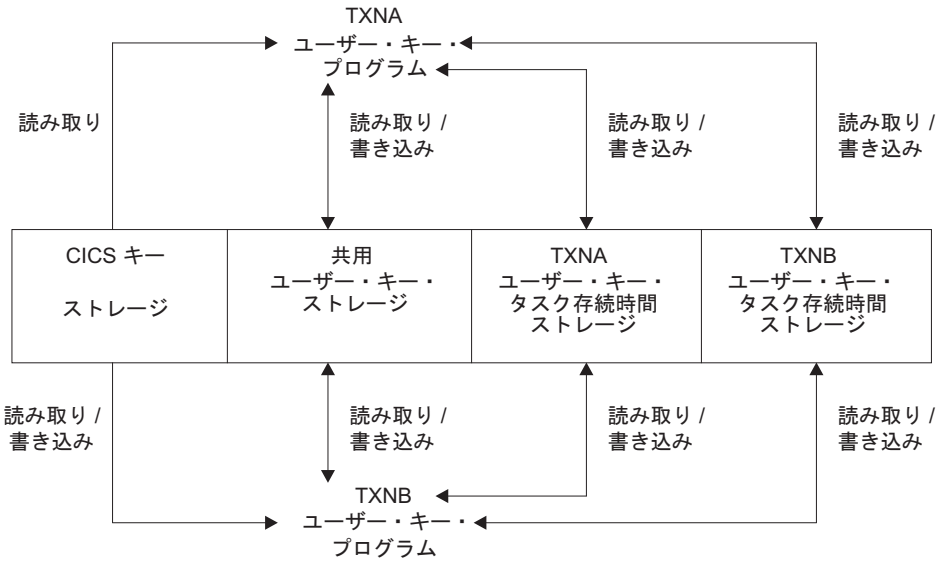


図 132. お互いのタスク存続時間ストレージに読み取り / 書き込みアクセスのある ISOLATE(NO) として定義された 2 つのトランザクション

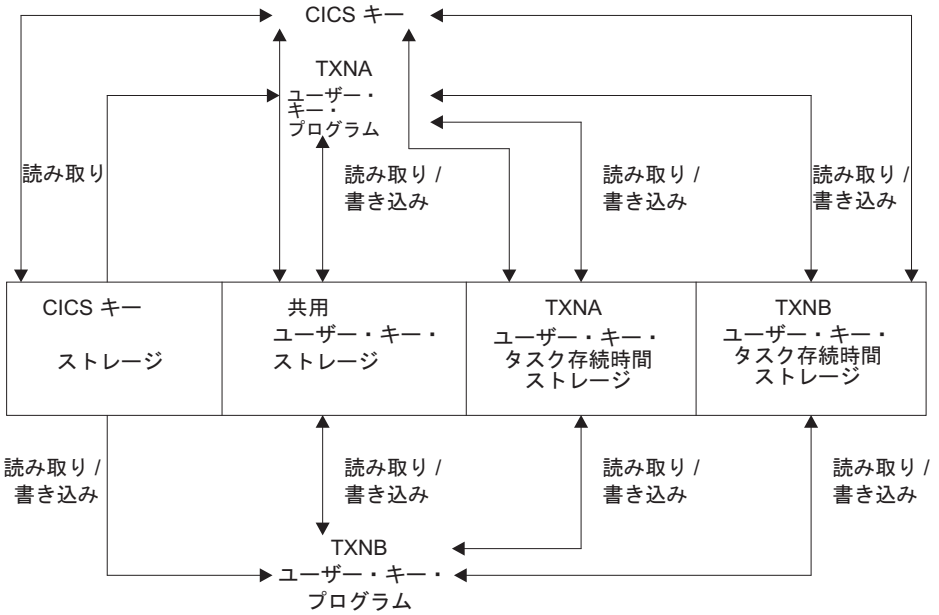


図 133. CICS キー・ストレージおよびユーザー・キー・ストレージの両方に読み取り / 書き込みアクセスのある CICS キー・プログラムに、ISOLATE(YES) として定義された 2 つのトランザクション

MVS サブスペース

ストレージ分離がアドレス・スペース内でデータ保全性を保持するために、MVS のサブスペース・グループ機能を使用することができます。サブスペース・グループ機能は、ハードウェアを使ってトランザクション・データの保護を提供します。サブスペース・グループはサブスペースと単一の基本スペースのグループで、基本スペースは、通常の MVS アドレス・スペースです。

サブスペース・グループ機能は、基礎の基本スペースの部分的マッピングを提供します。そのため基本スペース内のストレージの指定された区域だけが特定のサブスペースに現れます。各サブスペースは、基本スペース内のストレージの異なるサブセットを示します。トランザクション分離が指定されると、EXECKEY(USER) を指定して定義されたプログラムが、確実に自分のサブスペースで、あらゆる共用ストレージまたは CICS ストレージへの適切なアクセス権を持って実行されます。そのため、ユーザー・トランザクションは、アドレス・スペースの自分の「表示」に限定されます。

EXECKEY(CICS) で定義されたプログラムは、基本スペースで実行します。

トランザクションのサブスペースおよび基本スペース

一般に、トランザクション分離はユーザー・キー・プログラムが個々の (固有な) サブスペースに確実に割り振られるように、そして以下のアクセスを持つようにします。

- 自分のタスクのユーザー・キー・タスク存続時間ストレージへの読み取りおよび書き込みアクセス。このストレージ域は、ユーザー動的ストレージ域 (UDSA または EUDSA) の 1 つから割り振られたものです。
- 共用ストレージへの読み取りおよび書き込みアクセス。これは、SHARED オプション (SDSA または ESDSA) を使って GETMAIN コマンドによって獲得されたストレージです。
- その他のタスクの CICS キー・タスク存続時間ストレージ (CDSA または ECDSA) への読み取りアクセス。
- CICS コードへの読み取りアクセス
- CICS API によってアクセス可能な CICS 制御ブロックへの読み取りアクセス

他のタスクのユーザー・キー存続時間ストレージへのアクセスは持っていません。

新しいトランザクション・リソース定義属性のデフォルトは、既存のアプリケーション・プログラムがトランザクション分離と共に動作するように指定します (ISOLATE オプションのデフォルトは YES)。既存のアプリケーションは、トランザクション分離要件に適合している場合には、修正をしないで実行しなければなりません。

ただし、少数のアプリケーションは以下のような場合、特別な定義が必要なことがあります。

- MVS マクロを直接発行する場合、または
- CICS 制御ブロックを修正する場合、または
- 1 つのタスクが他のタスクのストレージにアクセスしたり、そのストレージを共用したりするための正当な必要性がある場合

既存のトランザクションにはさまざまな方法でタスク存続時間ストレージを共用するものがあります。そしてこの共用によって、トランザクションが相互に分離して実行されることを防ぎます。このようなトランザクションが継続して実行できるように、すべてのこのようなトランザクションが実行可能な単一の共通サブスペースが提供されます。これらは自分のサブスペースで実行中の、システムの他のトランザクションから分離していますが、共通サブスペース内では相互のデータを共用することができます。詳しくは、『共通サブスペースおよび共用ストレージ』を参照してください。

CICS キー・プログラムは、基本スペースで実行するので、すべての CICS キー・ストレージおよびユーザー・キー・ストレージへの読み取り / 書き込みのアクセス権を持っています。

共通サブスペースおよび共用ストレージ

トランザクションの中には、アプリケーション・プログラムが有効な方法でお互いのストレージにアクセスするものがあります。その 1 つのケースとしては、MVS POST または 他のタスクによる「ハンド POST」のいずれかによって、後で通知される 1 つ以上のイベント制御ブロック (ECB) でタスクが待機している場合です。

例えば、タスクは、別のタスクに (一時記憶キューまたはその他のいくつかの方法で) 自分のストレージの一部のアドレスを渡すことができます。そして、ストレージを更新したことを他方のタスクが ECB に通知するのを WAIT します。元のタスクが固有なサブスペースで実行中である場合、通知タスクは、CICS キーで実行されない限り、更新と ECB への通知を試みたときに障害を起こします。

CICS は、ストレージを共用する必要のあるトランザクションが確実にサブスペース・グループ環境で作業を継続できるように、以下の方式をサポートします。

- すべての関連したトランザクションを共通サブスペースで実行するように指定することができる。共通サブスペースは、ストレージを共用する必要があるタスクが共存できるようにします。一方で、タスクをシステム内の他のトランザクションから分離させます。共通サブスペースに割り当てられたトランザクションには、以下の特性があります。
  - お互いのタスク存続時間ストレージへの読み取りおよび書き込みアクセスがある。
  - 固有なサブスペースで実行されるトランザクションのどのような種類のストレージへのアクセスもない。
  - CICS ストレージへは読み取りアクセスのみがある。

  以前のリリース のユーザー・キーで作業する関連したトランザクションのグループはどれも、確実に共通サブスペースで実行するために ISOLATE(NO) で定義されている場合、CICS Transaction Server for z/OS バージョン 4 リリース 1 のもとで作業しなければなりません。このことで、CICS および関連したサポートをインストール後、トランザクションの分離が自分自身の固有なサブスペース内で徐々に実行されるようにして、アップグレードのためのサポートを提供します。
- SHARED オプションを使ってストレージを取得することによって、必ず SHARED ストレージの中に共用ストレージがあるようにできます。
- ストレージを共用しているトランザクションのアプリケーション・プログラムが必ず、すべて EXECKEY(CICS) で定義されるようにできます。このことによっ

て、確実に、基本スペースでプログラムを実行し、すべてのストレージへの読み取り / 書き込みアクセスをプログラムが持つようにします。ただし、この方式は、ストレージ保護がないので、推奨されていません。

CICS Interdependency Analyzer を使用すると、WAIT EVENT、WAITCICS、WAIT EXTERNAL、および MVS POST などのコマンドを含むトランザクションを識別する場合に役立ちます。このユーティリティーについて詳しくは、「*CICS Interdependency Analyzer for z/OS User's Guide and Reference*」を参照してください。

第 11 部 一時データと一時記憶域

一時データおよび一時ストレージ・キューの制御。

© Copyright IBM Corp. 1989, 2009

第 48 章 一時データ管理

CICS 一時データ管理機能は、汎用キューイング機能を提供します。データは、後で内部処理または外部処理するためにキューイング (保管) しておくことができます。アプリケーション・プログラムで指定されて選択されたデータを、事前定義されているシンボリック*区画内*または*区画外*一時データ・キューとの間でルーティングすることができます。

一時データ・キューは、CICS 領域 に割り振られた機能に関連している場合は区画内となり、CICS 領域 の外部の宛先にデータが送信される場合には区画外となります。一時データ・キューは、アプリケーション・プログラムによる最初の参照が行われるより前に、定義およびインストールしておかなければなりません。

以下のことを行うことができます。

- 一時データ・キューにデータを書き込む (WRITEQ TD コマンド)
- 一時データ・キューからデータを読み取る (READQ TD コマンド)
- 区画内一時データ・キューを削除する (DELETEQ TD コマンド)

TD キーワードを指定しないと、コマンドは一時記憶域のためのコマンドと見なされます。(一時記憶域の詳細については、677 ページの『第 49 章 一時記憶域制御』を参照してください。)

Java および C++

ここで説明するアプリケーション・プログラミング・インターフェースは、Java プログラムでは使用されない CICS API です。JCICS クラスを使用して一時データ・サービスにアクセスする Java プログラムについて詳しくは、「*Java Applications in CICS*」のJCICS クラス・ライブラリーおよび JCICS Javadoc 文書を参照してください。CICS C++ クラスを使用した C++ プログラムについて詳しくは、「*CICS C++ OO Class Libraries*」を参照してください。

区画内一時データ・キュー

「区画内」というのは、別個のタスクとして実行中の 1 つ以上のプログラムで使用するための直接アクセス記憶装置上のデータのことです。これらの内部キューとの間で送受信されるデータが、区画内データと呼ばれています。これらのデータは、可変長レコードから構成されている必要があります。すべての区画内一時データ宛先は、CICS が管理する同じ VSAM データ・セットのキューとして保持されます。区画内宛先は、区画内データ・セット内のキューを位置指定する情報を含むリソース定義を必要とします。区画内キューは、端末または出力データ・セットのいずれかに関連付けできます。ユーザー・タスクがキューにデータを書き込むと、CICS 領域内の他のタスクが入力データとしてキューを引き続き使用することができます。すべてのアクセスは順次に行われ、読み取りポインターおよび書き込みポインターによって管理されます。1 度レコードが読み取られると、その後そのレコードは他

© Copyright IBM Corp. 1989, 2009

のタスクから読み取ることができなくなります。区画内データは、最終的には要求に応じて端末に転送されるか、または出力データ・セットから順次に取り出されます。

区画内データの典型的な用途として、以下のものがあります。

- メッセージ交換
- ブロードキャスト
- データベース・アクセス
- いくつかの端末への出力のルーティング (例えば、注文の配布など)
- データのキューイング (例えば、到着順に注文番号または優先順位を割り当てる場合など)
- データ収集 (例えば、2780 データ伝送端末からのバッチ入力など)

区画内一時データ・キューには、次の 3 つのタイプがあります。

- **リカバリー不能**、リカバリー不能区画内一時データ・キューは、CICS のウォーム・スタート時にのみリカバリー可能です。作業単位 (UOW) がリカバリー不能区画内キューを更新して、その後にその更新をバックアウトした場合、そのキューに対して行われた更新はバックアウトされ**ません**。
- **物理的リカバリー可能**、物理的リカバリー可能区画内一時データ・キューは、ウォームリスタート時、および緊急リスタート時にリカバリーされます。UOW が物理的リカバリー可能区画内キューを更新して、その後にその更新をバックアウトした場合、そのキューに対して行われた更新は、バックアウトされ**ません**。
- **論理的リカバリー可能**、論理的リカバリー可能区画内一時データ・キューは、ウォーム・リスタート時、および緊急リスタート時にリカバリーされます。UOW が論理的リカバリー可能区画内キューを更新して、その後その更新による変更内容をバックアウトした場合、そのキューに対して行われた変更もまたバックアウトされます。ウォーム・リスタート時または緊急リスタート時に、論理的リカバリー可能区画内キューのコミット状態はリカバリーされます。未完了 UOW は無視されます。

  アプリケーションが、読み取り、書き込み、または削除要求の発行を試行中で、未確定の障害が起こった場合、キュー定義で WAIT(YES) および WAITACTION(REJECT) が指定されていれば、このアプリケーションは LOCKED 応答を受信することがあります。

区画外キュー

区画外キュー (データ・セット) は、CICS 領域の外側 (または内側) にあるプログラムからアクセス可能なすべての順次装置 (DASD、テープ、プリンターなど) にあります。

一般に、順次区画外キューは、CICS 領域の外側にあるデータを保管および取得するために使用されます。例えば、あるタスクでは、リモート端末からそのデータを読み取り、そのデータを編集して、その結果を別の領域での後続の処理のためにデータ・セットに書き込みます。ロギング・データ、統計、およびトランザクション・エラー・メッセージは、区画外キューへ書き込むことができるデータの例です。一

般に、CICS によって作成される区画外データは、非 CICS プログラムへの後続のバッチ入力のためのものです。また、データはプリンターなどの出力装置にルーティングすることもできます。

外部宛先との間でやりとりされるデータは、区画外データと呼ばれるもので、固定長または可変長のブロック化または非ブロック化された順次レコードから構成されます。区画外宛先のレコード形式は、システム・プログラマーが TDQUEUE リソース定義の中で定義しなければなりません。キュー定義について詳しくは、「*CICS Resource Definition Guide*」のを参照してください。

JCL (ジョブ制御言語) を使用して区画外キュー用のデータ・セット定義を作成する場合、そのデータ・セットの DD ステートメントには FREE=CLOSE オペランドを含めないでください。FREE=CLOSE を指定した場合は、キューがクローズされた後にキューの読み取りを試行し、再オープンすると、IOERR 状態が発生することがあります。CICS へのデータ・セットの定義の詳細については、「*CICS System Definition Guide*」のを参照してください。

間接キュー

区画内および区画外キューは、間接キューとして使用することができます。間接キューは、プログラム保守にある程度の柔軟性を提供しています。これにより、プログラムそれ自体ではなく一時データ定義のみを変更することで、複数のキューの 1 つにデータをルーティングすることができます。

一時データ定義が変更されたとき、アプリケーション・プログラムは元の記号名を使用してデータをそのキューにルーティングし続けますが、その時点でこの名前は、新しい記号名を表す間接キューです。間接キューは一時データ・リソース定義を使用して設定されるので、通常アプリケーション・プログラマーはどのように設定されるのかについて考える必要はありません。一時データ・リソース定義について詳しくは、「*CICS Resource Definition Guide*」のTDQUEUE リソース定義 を参照してください。

自動トランザクション開始 (ATI)

区画内キューに対して、CICS は自動トランザクション開始 (ATI) のオプションを提供します。

ATI の基本は、特定の区画内宛先にゼロ以外のトリガー・レベルを指定することによって、システム・プログラマーが設定します。キュー内の項目 (1 つ以上のプログラムによって発行された WRITEQ TD コマンドで作成されたもの) の数が、指定したトリガー・レベルに達したとき、キューの定義で指定されたトランザクションが自動的に開始されます。制御は、キュー内のデータを処理するプログラムに渡されます。キュー内の項目を使いきるためには、プログラムは繰り返し READQ TD コマンドを発行しなければなりません。

キューが空になると、ATI の新しいサイクルが始まります。すなわち、指定されたトリガー・レベルに再度到達すると、前のタスクの実行が終わっていてもいなくても、新しいタスクを開始するようにスケジュールされています。ATI の新しいサイクルが開始する正確な時点は、キューが論理的にリカバリー可能として定義されているかどうかによって決まります。キューがリカバリー可能性属性

(RECOVSTATUS) が No または Physical で定義されている場合は、キューを QZERO に読み取ると ATI の新しいサイクルが開始します。しかし、キューがリカバリー可能性属性の Logical で定義されているときは、キューを QZERO に読み取ってタスクが終了した後にのみ、ATI の新しいサイクルが開始します。

自動的に開始されたタスクがキューを空にしていない場合、キューへのアクセスは禁止されません。キューが空になる前に (すなわち、READQ TD コマンドの応答が QZERO 条件になる前に) タスクが正常に終了することも、異常終了することもあります。キューの内容が端末に送られることになっていて、直前のタスクが正常に完了している場合、QZERO に到達していないという事実は、トリガー処理がリセットされていないために同じタスクがもう一度開始されることを意味します。トリガー処理がリセットされていない場合、後続の WRITEQ TD コマンドによって、新しいタスクが起動されることはありません。

キューの内容がファイルに送られることになっている場合、タスクの終了は QZERO と同じ効果を持ちます (すなわち、トリガー処理がリセットされています)。次の WRITEQ TD コマンドによって、(トリガー・レベルに到達している場合には) トリガー・トランザクションが開始されます。

キューのトリガー・レベルがゼロの場合には、タスクが自動的に開始されることはありません。

キューが論理的にリカバリー可能な場合、トリガー・トランザクションの開始は次の同期点まで据え置かれます。

キューを消去する前に最後のトリガー・トランザクションが異常終了したため、あるいは MXT 限界に達したのでトランザクションが開始されなかったために、トリガー・レベルが既に超過されている場合には、別のタスクはスケジュールされません。これは、トリガー処理をリセットするための QZERO が起こらなかったからです。キューの内容がファイルに送られることになっている場合、タスクの終了がトリガー処理をリセットします。すなわち、次の WRITEQ TD コマンドが新しいタスクを起動するということです。

自動的に開始されたタスクを、キューが空になった時点で必ず完了させるには、アプリケーション・プログラムで、アプリケーションに依存する他の要素 (予定レコード数など) に優先して、QZERO 条件をテストする必要があります。キューが空であることを示すのは、QZERO 条件だけです。

キューの内容が別のシステムに送られることになっている場合、セッション名は EIBTRMID 内にあります。トランザクション (システムの宛先で開始されたもの) が異常終了する場合には、端末と同じ方法で新しいトランザクションが開始されます。

一時データ・トリガー機構と共に ATI を使用すると、動的トランザクション・ルーティングを実行する能力に不都合な影響を及ぼすトランザクション間の類縁性が生じることがあります。トランザクション類縁性の詳細については、339 ページの『第 22 章 類縁性』を参照してください。

トリガー・トランザクションは、未確定の障害がある場合、回避されます。回避された UOW が再同期化の後に行った変更をコミットまたはバックアウトするまでは、別のトリガー・トランザクションは生成されません。

第 49 章 一時記憶域制御

アプリケーション・プログラマーは、CICS 一時記憶域管理機能を使用することにより、主記憶装置の一時記憶キュー、直接アクセス記憶装置の補助記憶装置の一時記憶キュー、または一時データ共用プールの一時記憶キューに、データを保管することができます。一時記憶キューに保管されているデータを一時データと呼びます。

Java および C++

ここで説明するアプリケーション・プログラミング・インターフェースは、Java プログラムでは使用されない CICS API です。JCICS クラスを使用して一時記憶サービスにアクセスする Java プログラムについて詳しくは、「*Java Applications in CICS*」のJCICS クラス・ライブラリーおよび JCICS Javadoc 文書を参照してください。CICS C++ クラスを使用した C++ プログラムについて詳しくは、「*CICS C++ OO Class Libraries*」を参照してください。

以下のことを行うことができます。

- 一時記憶キューにデータを書き込む (WRITEQ TS コマンド)。
- 一時記憶キューのデータを更新する (WRITEQ TS REWRITE コマンド)。
- 一時記憶キューからデータを読み取る (READQ TS コマンド)。
- 一時記憶キューから次のデータを読み取る (READQ TS NEXT コマンド)。
- 一時記憶キューを削除する (DELETEQ TS コマンド)。

TS キーワードは省略してもかまいません。これを指定しなければ、一時記憶域が想定されます。

一時記憶域管理コマンドの実行中に発生した例外条件は、 383 ページの『第 24 章 例外条件の取り扱い』で説明されているように処理されます。

これらのコマンドを使用すると、動的トランザクション・ルーティングを実行する機能に悪い影響を与えるトランザクション間の類縁性が生じます。

これらのコマンドを発行するプログラムにおいて発生する可能性のある問題の識別を容易にするため、CICS Interdependency Analyzer のスキャナーおよびコレクターのコンポーネントを使用できます。このユーティリティーについて詳しくは「*CICS Interdependency Analyzer for z/OS User's Guide and Reference*」を参照し、トランザクションの類縁性について詳しくは 339 ページの『第 22 章 類縁性』を参照してください。

一時記憶域キュー

一時記憶キューは、親タスクによって割り当てられた 16 文字までのシンボル名によって識別されます。一時データは、親タスクによって、またはそれに割り当てられたシンボル名を使用している他のタスクによって取り出すことができます。重複名による競合を避けるためには、命名規則を設定しなければなりません。例えば、

© Copyright IBM Corp. 1989, 2009

オペレーター ID または端末 ID を、プログラマーが提供した各シンボル名の接尾部として使用することができます。キュー内の特定の項目 (論理レコード) は、相対位置の番号によって参照されます。

一時記憶キューは、親タスク、他のタスク、あるいは、初期スタートまたはコールド・スタートによって削除されるまで、そのまま残っています。削除の前には、何度でもアクセスすることができます。親タスクを終了した後でも、一時データは、保管されたシンボル名に対する参照によって他のタスクがアクセスすることができます。

一時データは、主記憶装置または補助記憶装置に記憶することができます。一般に、データが短期間だけ必要な場合には、主記憶装置を、また、データを長期間保持しておく場合には、補助記憶装置を使用すべきです。補助記憶装置に保管されたデータは、CICS の終了後に保存され、後続の再始動でリカバリーすることができますが、主記憶装置のデータはリカバリーすることができません。主記憶装置は、タスク間でデータを渡すために、あるいは、プログラムを準再入可能 (すなわち、プログラムの入り口点と出口点の間で逐次再使用可能) にするという、CICS の要件にプログラムを合わせることができる固有のストレージ用に、使用することができます。

一時データの共用は、同時にサポートすることができる別のタイプの一時記憶キューを提供します。一時記憶キューは、ローカル、リモート、または共用として定義することができ、カップリング・ファシリティーの一時記憶プールに保管することができます。

一時記憶管理の標準的な使用法

レコードが 1 つしかない一時記憶キューは、その記号名を使用してアクセスできるデータの単一の単位として取り扱うことができます。この方法で一時記憶域管理を使用すると、典型的なスクラッチ・パッド機能が提供されます。このタイプのストレージは、ITEM オプションを指定した READQ TS コマンドを使用してアクセスしなければなりません。そうしなければ ITEMERR 条件が起こる場合があります。

一般に、複数のレコードからなる一時記憶キューを使用するのは、レコードの直接アクセスまたは反復アクセスが必要な場合のみにしてください。一時データ管理では、順次データ・セットを効率的に処理する機能が提供されています。

一時記憶キューの用途には、以下のものがあります。

端末ページング
タスクでは、直接アクセス・データ・セットから 1 つの大きいマスター・レコードを取得し、(BMS を使用して) それを複数の画面イメージに形式設定し、その画面イメージを一時的に補助記憶装置に保管して、どの「ページ」(画面イメージ) が必要であるかを端末オペレーターに要求します。アプリケーション・プログラマーは、ページ単位で進めたり、ページの相対番号に進めたり返したりするためのプログラムを (汎用ルーチンまたは単一アプリケーションに固有のものとして) 提供することができます。

中断データ・セット
データ収集タスクが端末で進行中であるとします。タスクは、1 つ以上の単位の入力を読み取り、その後ある種のコード化入力によって端末オペレータ

ーが処理に割り込めるようにします。割り込まれなかった場合には、タスクはデータ収集処理を繰り返します。割り込まれた場合には、タスクは、その未完成データを一時記憶域に書き込んで終了します。端末は別のトランザクション (高優先順位照会と考えられる) を処理するために、ここで解放されます。端末が使用可能でデータ収集を続行できる場合、オペレーターは「再開」モードでタスクを開始します。これにより、タスクは一時記憶域からその中断データを再呼び出しして、中断などなかったかのように処理を続行します。

事前印刷用紙

アプリケーション・プログラムは、事前印刷用紙に出力として書き込むデータを受け入れることができます。このデータは、到着した通りに一時記憶域に保管することができます。すべてのデータが保管されたら、最初に妥当性検査され、事前印刷用紙の形式によって必要な順序で転送されます。

第 12 部 CICS 文書

CICS 文書ハンドラーを使用すると、フォーマット済みデータ域を作成することができます。これを文書と呼びます。

これらのフォーマット済みエリア、つまり文書の使用方法の例は、以下のとおりです。

- COMMAREA を構築する。
- Web クライアントで表示される HTML データを送信する。
- 印刷用の標準形式を作成する (例えば、自分用のレターヘッドやアドレスを用いて)。

© Copyright IBM Corp. 1989, 2009

第 50 章 文書および文書テンプレートの概要

アプリケーション・プログラムは、文書を作成して、EXEC CICS DOCUMENT アプリケーション・プログラミング・インターフェースのコマンドを使用し、データを文書内に配置します。文書テンプレートは、オフライン、または他の CICS プログラムで作成できる文書の部分であり、アプリケーション・プログラムにより文書の作成で使用されます。

文書および文書テンプレートは、CICS Web サポートにより提供され、Web ページの作成で最も一般的に使用されます。HTTP 要求または応答の本文として使用される HTML を含めることができます。ただし、用途はこれに制限されません。

文書

EXEC CICS DOCUMENT CREATE コマンドを使用してアプリケーション・プログラム内に空の文書を作成し、続けて DOCUMENT INSERT コマンドを使用して内容を構築できます。または、DOCUMENT CREATE を使用して、文書を 1 ステップで作成および構築できます。アプリケーション・プログラムにより指定されるデータを使用、文書テンプレートを使用、または別の文書を使用して文書を作成できます。文書ハンドラーはトークン (DOCTOKEN) を返します。このトークンは、以後の呼び出しで文書を識別するのに使用されます。

文書を作成すると、1 つ以上の DOCUMENT INSERT コマンドを実行することによって、内容を拡張できます。また、アプリケーション・プログラム、文書テンプレート、または別の文書により指定されるデータを追加できます。文書のデータ・ブロックとデータ・ブロックの間にブックマークを挿入し、そのブックマークを使用して文書の途中にデータを追加したり、文書の途中のデータを置き換えたりすることもできます。

アプリケーションが作成した文書は、それが作成された CICS タスクの有効期間の間のみ存在します。つまり、CICS タスクの最後のプログラムが CICS に制御権を返したときに、そのタスクの存続期間の間に作成された文書は、すべて削除されます。その文書を別のタスクで使用する場合、終了前に文書を保存するのはアプリケーションの仕事です。DOCUMENT RETRIEVE コマンドを使用して、文書のコピーをとることができます。アプリケーションはこのコピーを、好みの場所 (例えば一時記憶キュー) に保存することができます。そのコピーを使用して、文書を再作成できます。

文書テンプレート

文書テンプレートは、アプリケーション・プログラムで使用される方法に合わせて、いくつかの異なるソースから取得できます。文書テンプレートのソースは、以下のいずれかになります。

- 区分データ・セット
- CICS プログラム
- CICS ファイル

© Copyright IBM Corp. 1989, 2009

- z/OS UNIX システム・サービス・ファイル
- 一時記憶キュー
- 一時データ・キュー
- 出口プログラム

文書テンプレートは、文書テンプレートのソースを指定する DOCTEMPLATE リソース定義を使用して定義されます。

文書テンプレートには、静的データおよび記号を含めることができます。記号は、テンプレートが文書に追加されるとき、つまり、DOCUMENT CREATE または DOCUMENT INSERT コマンドが実行されるときに解決される変数データを表します。記号を置換する値は、DOCUMENT CREATE コマンドの SYMBOLLIST オプション、または DOCUMENT SET コマンドの SYMBOLLIST か SYMBOL オプションを使用してアプリケーション・プログラムにより指定されます。

また、文書テンプレートには、記号のデフォルト値を設定して、記号を識別し、別のテンプレートを埋め込むための、埋め込みコマンドを含めることもできます。

文書テンプレートは、アプリケーション・プログラムにより使用されると同時に、URIMAP 定義で指定されて Web クライアントの HTTP 要求への静的応答を返すこともできます。この際、アプリケーション・プログラムは必要ありません。CICS は Web ページの本文として文書テンプレートを使用して応答を作成します。

記号および記号リスト

文書テンプレートで記号を使用すると、アプリケーション・プログラムは、ユーザー名やオーダー番号など、現行タスクにふさわしいデータを含めるように文書をカスタマイズできます。記号は、テンプレートが文書に追加されるとき、つまり、DOCUMENT CREATE または DOCUMENT INSERT コマンドが実行されるときに解決される変数データを表します。

各記号には名前があり、値を割り当てることができます。記号を含む文書テンプレートが文書に挿入されると、CICS 文書ハンドラーは**記号置換**処理のために、記号に割り当てられている値を使用します。ここで、文書テンプレート内の記号は、その値に置換されます。例えば、記号 ORDER_NUMBER に値 0012345 が割り当てられ、文書テンプレート内で使用されると、以下のようになります。

```
Thank you! Your order number is &ORDER_NUMBER;.
```

記号置換後の完成した文書は、以下のとおりです。

```
Thank you! Your order number is 0012345.
```

記号参照は、文書テンプレート内に組み込まれ、完成した文書のテキスト内の記号が値に置換される場所に含まれます。記号参照は、最初にアンパーサンド (&)、最後にセミコロン (;) の付いた記号名により構成されます。記号は文書テンプレート内の任意の場所で使用できます。また、埋め込みテンプレート・コマンド #set を使用して、文書テンプレート内の任意の場所の記号に対してデフォルト値を含めることもできます。

文書テンプレートを使用して文書を作成するアプリケーション・プログラムは、テンプレートを使用するときに、置換される記号に値を割り当てる必要があります。DOCUMENT SET コマンドまたは DOCUMENT CREATE コマンドを使用して、この処理を実行できます。

DOCUMENT SET コマンドで SYMBOL および VALUE オプションを使用して、個々の記号の値を指定することもできます。または、**記号リスト**を指定することにより、複数の記号を単一のコマンドで定義できます。記号リストとは、それぞれが名前、等号、および値を含み、1 バイトの分離文字 (デフォルトではアンパーサンド) の付いた 1 つ以上の定義で構成される文字ストリングのことです。DOCUMENT CREATE または DOCUMENT SET コマンドの SYMBOLLIST オプションを使用して指定できます。例えば、記号 ORDER_NUMBER に値「0012345」を、記号 COUNTRY に値「Germany」を指定する記号リストは、以下のとおりです。

```
ORDER_NUMBER=0012345&COUNTRY=Germany
```

記号に割り当てる値は、**記号テーブル**に保持されます。各文書テンプレートに関連付けられた記号テーブルがあります。#set コマンドを使用する場合、またはアプリケーションで記号に値を提供する場合に、記号名および関連する値が記号テーブルに追加されます。記号が記号テーブルに既に存在する場合には、その値が新規の値で置換されます。アプリケーションが提供する記号定義は、#set コマンドにより提供されるデフォルト値を指定変更します。

DOCUMENT CREATE または DOCUMENT INSERT コマンドにより文書テンプレートが文書に挿入された場合、記号テーブルで指定された現行値を使用して、記号テーブルに存在するすべての記号に対して記号置換が実行されます。記号がアプリケーション・プログラムまたは #set コマンドにより指定されていない場合、記号置換は実行されず、完成した文書には記号名または記号を指定する #echo コマンドが含まれます。

記号を含むテンプレートを使用した場合、テンプレートを文書に挿入する**前**に、記号に対して必要な値を指定する必要があります。テンプレートを挿入するが、値をその中の記号に割り当てていない場合、記号は置換されません (これは、記号リストまたは記号の値を指定せずにテンプレートから文書を作成した場合に発生します)。テンプレートを文書に挿入した後は、CICS が記号の代わりに文書に入れた値は変更できません。テンプレートを挿入した後で値を記号に対して指定した場合、その値は記号テーブルに配置され、次にテンプレートが文書に挿入されたときに使用されますが、その変更は既に文書に挿入された値には影響を与えません。

記号および記号リストのデータ・フォーマット

DOCUMENT アプリケーション・プログラミング・インターフェースにおける記号および記号リストのサポートは、HTML 仕様 (http://www.w3.org/TR/html401/interact/forms.html#h-17.13.4) の解説に従って、データが **application/x-www-form-urlencoded** のコンテンツ・タイプで解釈されるように設計されています。これは、Web ページのフォームに入力されたデータが返されるという形式になっています (Web ページは、CICS 文書テンプレートで最も一般的に使用されます)。

url でエンコードされたデータのフォーマットは、名前と値の組のシーケンスで、アンパーサンドにより区切られています。例:

```
firstname=Irving&lastname=Berlin
```

通常、値はユーザーがフォームに入力したデータです。

ただし、フォームに入力したデータが、url でエンコードしたデータのフォーマットで特殊な意味を持つ文字を含む場合、フォームに入力した文字は、名前と値の組を区切るために使用される文字と区別される必要があります。このために、文字エスケープの処理が実行されます。エスケープ文字とは、3 文字のシーケンス %xx により置換される文字のことです。ここで、xx は、文字の ASCII エンコードの 16 進値です。

アンパーサンド (&)、等号 (=)、およびパーセント記号 (%) は、エスケープ・シーケンスを導入するために使用されるので、url でエンコードされるデータではすべてエスケープされる必要があります。スペース文字は区切り文字として使用される場合が多いため、常にエスケープされます。ただし、スペースは一般的であるため、url でエンコードされたデータ型では、スペースをエスケープ・シーケンスとしてではなく、正符号 (+) としてエンコードできます。つまり、このことは、フォームに入力された正符号も、エスケープされる必要があることを意味します。

例えば、フォームに以下の名前および値が含まれているとします。

```
sum         8+11=19
rate        19%
composers   George & Ira Gershwin
```

このデータのエスケープされた url でエンコードされた表記は、以下のとおりです。

```
sum=8%2b11%3d19&rate=19%25&composers=George+%26+Ira+Gershwin
```

値のスペース文字は正符号としてエンコードされ、値の正符号、等号、パーセント記号、およびアンパーサンドは、エスケープ・シーケンスとしてエンコードされます。

EXEC CICS DOCUMENT CREATE または DOCUMENT SET コマンドの SYMBOLLIST オプションを使用して指定する記号リストには、原則として、url でエンコードされたフォーマットの名前と値の組のリストが含まれます。ただし、CICS はこの構文を以下の方法で拡張します。

- スペースはエスケープする必要がありません。スペースは、シングル・スペース、正符号、またはエスケープ・シーケンス %20 として表記できます。
- 名前と値の組の間の区切り文字は、アンパーサンドである必要はありません。コマンドの DELIMITER オプションを使用して、代わりの区切り文字を指定できます。

通常、アンエスケープ処理は、記号が記号テーブルに書き込まれる場合に記号の値に対して実行され、特殊なコーディングを使用して指定された文字は、意図された文字に変換されます。例えば、正符号はスペースに変換され、エスケープ・シーケンスは適切な文字に変換されます。ただし、DOCUMENT CREATE または DOCUMENT SET コマンドで UNESCAPED オプションを指定すると、変換は行われず、記号値は、入力したとおりに記号テーブルに書き込まれます。

HTML コメントの記号

文書テンプレートの HTML コメント内では、通常、記号は無視され、CICS 文書ハンドラーは記号置換を実行しません。HTML コメントは、開始のマークアップ <!-- および終了のマークアップ --> で区切られます。

文書テンプレートの HTML コメント内で記号置換を実行する場合は、コメントの開始区切り文字および終了区切り文字の代わりに記号を使用して、文書テンプレートで #set コマンドを使用し、それらの記号の値として開始区切り文字および終了区切り文字を指定します。この場合、CICS は区切り文字の代わりに記号を使用した箇所の周辺のコメント全体に対して記号置換を実行します。

例えば、文書テンプレートには、以下を含めることができます。

```
<!--#set var=OC value='<!--'-->
<!--#set var=CC value='-->'-->

&OC; A comment containing my text &SYM; &CC;
```

アプリケーション・プログラムが値「Example text」を記号 SYM に割り当てている場合、CICS はその記号の値、および記号 OC と CC の値を置換して、以下のような HTML 出力を生成します。

```
<!-- A comment containing my text Example text -->
```

文書テンプレートのキャッシングおよびリフレッシュ

パフォーマンスの改善のために、CICS 文書ハンドラーは、ほとんどの文書テンプレートのコピーをキャッシュに入れます。アプリケーションがテンプレートを参照するとき、キャッシュに入っているコピーが使用され、パフォーマンスが向上します。文書テンプレートを変更した場合は、キャッシュに入っているコピーをいつでもリフレッシュできます。また、文書テンプレートとして定義されたプログラムおよび出口プログラムの新しいコピーをフェーズできます。

CICS は、以下のタイプの文書テンプレートのコピーを、常にキャッシュに入れます。

- 区分データ・セット内のテンプレート
- CICS ファイル内のテンプレート
- z/OS UNIX システム・サービス・ファイル内のテンプレート
- 一時記憶キュー内のテンプレート
- 一時データ・キュー内のテンプレート

これらのタイプの文書テンプレートの 1 つが、CICS を実行している間に個別にインストールされた場合、その文書テンプレートは CICS 文書ハンドラーのストレージに読み込まれます。アプリケーションから文書テンプレートにアクセスする要求では、キャッシュに入っているテンプレートのコピーを受け取るため、CICS は、文書テンプレートが保管されている場所に毎回アクセスする必要はありません。CICS の開始中にインストールされた文書テンプレートは、このときにはキャッシュされません。これらの各文書テンプレートは、アプリケーションにより初めて参照されたときにキャッシュされます。

キャッシュに入っている文書テンプレートを変更したい場合、CEMT または EXEC CICS SET DOCTEMPLATE NEWCOPY コマンドを使用して、キャッシュ内の文書テンプレートのコピーをリフレッシュできます (EXEC CICS DOCUMENT API の一部ではない SET DOCTEMPLATE コマンドを使用する場合、テンプレートの 48 文字の名前の代わりに、文書テンプレートを定義する DOCTEMPLATE リソース定義の名前を指定する必要があります)。

上記で示したタイプの文書テンプレートでは、SET DOCTEMPLATE NEWCOPY コマンドで、CICS 文書ハンドラーにより現在キャッシュに入れられている文書テンプレートのコピーを削除して、文書テンプレートが保管されている場所から読み取ったコピーに置き換えます (区分データ・セットのテンプレートでは、CICS はまず BLDL (ビルド・リスト) を実行して現行ディレクトリー情報を取得し、その後でメンバーを再度読み取ります)。新しくキャッシュ内のコピーが作成された場合、文書テンプレートを使用する後続の要求は、新しいコピーを使用します。新しいコピーは、同じタスク内の後続の要求、および他のタスクでの要求により使用されます。

CICS システムがストレージ不足になった場合、文書ハンドラーはキャッシュ内の文書テンプレートのコピーを一部削除して、ストレージの制約を解消しようとします。削除される文書テンプレートは、サイズが大きいものから順に選択され、キャッシュ内にコピーが作成された後の経過時間も考慮されます (このため、新しく作成されたコピーは、すぐには解放されません)。

CICS システムがウォーム・スタートで再始動された場合、以前にキャッシュに入れられた文書テンプレートは、再ロードされません。キャッシュは、アプリケーションにより初めて各文書テンプレートが参照された時点で、再度作成されます。

文書テンプレートに関して収集された CICS 統計では、各文書テンプレートが参照された回数、キャッシュ内にコピーが作成された回数、リフレッシュされた回数、使用された回数、および削除された回数を示します。

CICS プログラム内のテンプレート

CICS プログラムから取得された文書テンプレートは、文書ハンドラーによってキャッシュに入れられることはありません。これは、プログラムが既に CICS のどこかでキャッシュに入れられているためです。

このタイプの文書テンプレートでは、SET DOCTEMPLATE NEWCOPY コマンドを使用して、プログラムの新しいコピーをフェーズできます。このコマンドは、指定されたプログラムの SET PROGRAM PHASEIN に相当するものです。同じタスク内の後続の要求を含めて、文書テンプレートを使用する後続の要求では、新しいコピーが使用されます。

出口プログラムのテンプレート

出口プログラムにより生成された文書テンプレートでは、文書テンプレートのコピーを文書ハンドラーによりキャッシュに入れる必要があるかどうかを、出口プログラムが (出口パラメーター・リストで) 指定します。デフォルトでは、文書テンプレートはキャッシュに入れられません。動的に変更されるテンプレートはキャッシュに入れないようにする必要がありますが、テンプレートが変更されない場合は、キャッシングにより要求のパフォーマンスが向上するので、キャッシングは適切で

す。出口プログラムがキャッシングを指定した場合、キャッシュ内のコピーは、文書テンプレートがアプリケーションにより初めて参照されたときに作成されます。

このタイプの文書テンプレートでは、SET DOCTEMPLATE NEWCOPY コマンドを使用して、出口プログラムの新しいコピーをフェーズできます。このコマンドは、指定された出口プログラムの SET PROGRAM PHASEIN に相当するものです。このコマンドを実行すると、CICS はキャッシュ内の文書テンプレートのコピーを削除して、プログラムの新しいコピーをフェーズし、出口プログラムでキャッシングが指定されている場合は、キャッシュ内に新しい文書テンプレートのコピーを作成します。リフレッシュされた出口プログラムは、キャッシングを行う必要があるかどうかに対して異なる設定を指定でき、CICS は変更を引き継ぎます。

文書のコード・ページ変換

例えば、文書を使ってクライアントに Web ページを提供する場合など、アプリケーションが作成する文書は、他のプラットフォームで稼働するシステムに転送できます。CICS システムにより使用されるコード・ページ内のテキスト・データは、ターゲット・システム上で使用されるコード・ページに変換する必要があります。この処理は、コード・ページ変換として知られています。

CICS システムにより使用されるコード・ページは、CICS が HTTP クライアントとして動作する場合を除き、通常、ホスト・コード・ページとして記述されます。ターゲット・システムにより使用されるコード・ページは、クライアント・コード・ページとして記述されます。また、ターゲット・システムが ASCII を使用している Web クライアントまたはサーバーの場合、文字セットとして参照されることがあります。

CICS 文書には、それらの文書が作成されたコード・ページに関する情報を含めるようにすることができます EXEC CICS DOCUMENT CREATE および EXEC CICS DOCUMENT INSERT コマンドを使用して文書を作成する場合、HOSTCODEPAGE オプションを、TEXT、FROM、TEMPLATE、および SYMBOL オプションのいずれかと共に指定して、そのデータ・ブロックのコード・ページを示すことができます。個々のブロックは、それぞれ異なるコード・ページに指定できます。

EXEC CICS DOCUMENT RETRIEVE コマンドを使用して、送信する文書を取得する場合、CHARACTERSET オプションを指定して、文書ハンドラーがすべての個別ブロックを、それぞれのホスト・コード・ページからターゲット・システムで使用するのに適切な単一のクライアント・コード・ページに変換できます。

CICS Web サポートでは、CICS 文書が EXEC CICS WEB API コマンドを使用した Web 対応アプリケーション・プログラムにより送信されるよう指定した場合、アプリケーション・プログラムで EXEC CICS DOCUMENT RETRIEVE コマンドは使用されません。代わりに、CICS 文書の文書トークンが指定され、CICS が文書の取得を管理します。クライアント・コード・ページへの変換は、アプリケーション・プログラムが EXEC CICS WEB API コマンドで指定したオプションに従って、CICS により処理されます。

また、CICS Web サポートでは、CICS 文書テンプレートが URIMAP 定義で静的応答を Web クライアントに提供するよう指定された場合に、クライアント・コード・ページへの変換が CICS により処理されます。テンプレートが存在するホス

ト・コード・ページ、および変換結果となるクライアント・コード・ページは、URIMAP 定義で指定されます。静的応答が必要な場合、CICS はテンプレートを使用して文書を作成し、その文書を取得して、適切なコード・ページ変換を実行します。

第 51 章 文書テンプレートの設定

文書テンプレートとは DOCTEMPLATE リソース定義を使用して定義する CICS リソースのことです。

EXEC CICS DOCUMENT API コマンドでテンプレートを参照するために使用されるテンプレートの名前は、リソース定義の TEMPLATENAME 属性で指定されます。また、DOCTEMPLATE リソース定義は、テンプレートのソース、データのフォーマット (バイナリーまたは EBCDIC)、および CICS がテンプレートの各レコードに復帰改行を追加するかどうかについても指定します。

区分データ・セット内のテンプレート

BMS マップ・セット定義から作成された HTML テンプレートは、区分データ・セットに保管されます。

区分データ・セットに保管された文書テンプレートには、DOCTEMPLATE リソース定義内で属性 MEMBERNAME が含まれています。

CICS は、対応する DOCTEMPLATE 定義をインストールするときに、区分データ・セットからテンプレートのコピーをロードします。CICS の稼働中に、区分データ・セット内のテンプレートを変更し、変更を活動化するために SET DOCTEMPLATE NEWCOPY コマンドを実行します。CICS はまず BLDL (ビルド・リスト) を実行して現行ディレクトリー情報を取得し、次にメンバーを再度読み取ります。

テンプレートの保管に使用される区分データ・セットは、以下のいずれかのレコード形式を持っている場合があります。

- FB (ブロック化された固定長)
- VB (ブロック化された可変長)
- U (非ブロック化)

以下の場合に、レコードにはシーケンス番号が含まれます。

- レコード形式が FB で、レコード長が 80 である場合、シーケンス番号は 73 から 80 の間でなければなりません。
- レコード形式が VB である場合、シーケンス番号は 1 から 8 の間でなければなりません。

その他の場合、レコードにはシーケンス番号はありません。シーケンス番号を使用する場合は、すべてのレコードにその番号が付いている必要があります。部分的にシーケンス・メンバーを使用しないでください。

区分データ・セットに保管されるテンプレートは、CICS 文書ハンドラーによりキャッシュされます。

© Copyright IBM Corp. 1989, 2009

z/OS UNIX システム・サービス・ファイル内のテンプレート

z/OS UNIX システム・サービス・ファイルは、文書テンプレートとして定義できます。z/OS UNIX ファイルは、作成および編集に最適なタイプの文書テンプレートです。

z/OS UNIX ファイルに保管される文書テンプレートは、DOCTEMPLATE リソース定義内に属性 HFSFILE を持ちます。

CICS Web サポートでは、URIMAP 定義を使用して、z/OS UNIX ファイルを静的応答として直接配信できます。この場合、z/OS UNIX ファイルを文書テンプレートとして定義する必要はありません (定義する場合もあります)。ただし、シンボル置換を実行する場合や、Web 対応アプリケーション・プログラムがファイルにアクセスする必要がある場合には、文書テンプレートとして定義する必要があります。

DOCTEMPLATE 定義では、ファイルの完全修飾名 (最大 255 文字長) を指定する必要があります。CICS 領域は、z/OS UNIX へのアクセス権と、ファイルを含むディレクトリー、およびファイル自身へのアクセス権を持っている必要があります。これを実現する方法については、「*Java Applications in CICS*」の z/OS UNIX ディレクトリーおよびファイルへのアクセス権を CICS 領域に付与するで説明しています。

ファイル内のすべてのデータは、文書テンプレートとして使用されます。

z/OS UNIX ファイルに保管されるテンプレートは、CICS によりキャッシュされます。SET DOCTEMPLATE NEWCOPY コマンドを使用して、現在キャッシュに入っている文書テンプレートのコピーを削除し、文書テンプレートが保管されている場所から読み取ったコピーで置換できます。

CICS ファイル、一時記憶、または一時データ内のテンプレート

文書テンプレート内のアプリケーション・プログラムから動的データを使用する場合は、これらのリソースのいずれかを使用することを考えてください。

どのリソースを使用するかは、以下によって決まります。

- アプリケーション・プログラムがそのデータを保管する方法
- 既存のデータをテンプレート内で直接使用できるかどうか、または変更する必要があるかどうか
- テンプレート内で使用した後でデータを保存する必要があるかどうか

一般に、テンプレートが文書に挿入されると、リソース内に含まれるすべてのデータが使用されます。

これらのリソースに保管される文書テンプレートには、DOCTEMPLATE リソース定義内で属性 FILE、TSQUEUE、および TDQUEUE が含まれています。

属性 TSQUEUE または TDQUEUE を使用して定義される DOCTEMPLATE を使用している場合、2 バイト (復帰および改行) が DOCTEMPLATE バッファーの終わりに追加されます。

一時記憶域

キューは、順番に ITEM 番号によって読み取られます。したがって、キュー内のレコードは、他のアプリケーションによってどのレコードが読み取られたかに関係なく、すべて読み取られます。

一時データ

文書テンプレートとして使用する場合、**区画外一時データ・キュー**が適切です。キュー内のすべてのレコードが読み取られます。

文書テンプレートとして使用する場合、**区画内一時データ・キュー**は不適切です。一時データは破壊読み出しを使用するため、一時データからテンプレートにデータを挿入すると、キューの内容は他のアプリケーションでは使用できなくなります。

CICS ファイル

- 入力順データ・セット (ESDS) は、相対バイト・アドレス順に読み取られます。
- 相対レコード・データ・セット (RRDS) は、相対レコード番号順に読み取られます。
- その他のデータ・セットは、キー・フィールドのシーケンスで読み取られます。

文書テンプレートとして使用する CICS ファイルは、BROWSE 属性を使用して指定する必要があります。

これらのフォーマットのいずれかで保管されるテンプレートは、CICS によりキャッシュされます。SET DOCTEMPLATE NEWCOPY コマンドを使用して、現在キャッシュに入っている文書テンプレートのコピーを削除し、文書テンプレートが保管されている場所から読み取ったコピーで置換できます。

CICS プログラムのテンプレート

プログラムの文書テンプレートは、テンプレートを検索するためのオーバーヘッドが最も小さくなりますが、その他のフォーマットとしては、キャッシングも最初の要求後のオーバーヘッドを最小化します。同じテンプレートが複数のアプリケーションにより使用されている場合は、プログラムが、選択すべき適切な形式です。プログラムは簡単には編集できず、再コンパイルが必要なため、テンプレートがあまり頻繁に変更されない場合に、CICS プログラムが最も適切です。

このタスクについて

CICS プログラムに含まれる文書テンプレートには、DOCTEMPLATE リソース定義内で属性 PROGRAM が含まれています。

CICS プログラムに含まれるテンプレートは、CICS 文書ハンドラーによってキャッシュに入れられることはありません。プログラムは既に CICS ローダーによりキャッシュに入れられています。

このタイプの文書テンプレートでは、SET DOCTEMPLATE NEWCOPY コマンドを使用して、プログラムの新しいコピーをフェーズできます。このコマンドは、指定

されたプログラムの SET PROGRAM PHASEIN に相当するものです。同じタスク内の後続の要求を含めて、文書テンプレートを使用する後続の要求では、新しいコピーが使用されます。

テンプレートを含むプログラムをコーディングするには、以下のようにします。

1. 以下を含むアセンブラー CSECT をコーディングします。
   a. テンプレートの開始を指示する ENTRY ステートメント。
   b. テンプレートに含めるテキストを定義する文字定数 (DC ステートメント)。
   c. END ステートメント。
   d. 文書テンプレートの prolog および epilog を生成する DFHDHTL マクロの起動。 このマクロについて詳しくは、『DFHDHTL - プログラム・テンプレート prolog マクロおよび epilog マクロ』を参照してください。

   例:

```
DOCTPROG CSECT
DOCTPROG AMODE 31
DOCTPROG RMODE ANY
         DFHDHTL TYPE=INITIAL,ENTRY=WKLYHDR
         DC    CL4'<HR>'
         DC    CL29'<H2>Weekly Status Report</H2>'
         DFHDHTL TYPE=FINAL
         END   WKLYHDR
```

   テンプレート・プログラムは、8 バイトの正確な倍数である必要があるため、DFHDHTL マクロが使用されます。8 バイトの正確な倍数でないと、バインド・プログラムがテンプレートの終わりに偽のバイナリー文字を挿入する場合があります。そのような場合にテンプレートが文書で使用されると、予測不能な出力を生成することがあります。DFHDHTL マクロは、必要な埋め込みを作成します。
2. プログラムをアセンブルし、CICS アプリケーション・プログラム・ライブラリー内にリンク・エディットします。 プログラムに指定する名前は、エントリー・ポイントの名前とは異なる可能性があります。
3. PROGRAM 属性にプログラムの名前を指定する DOCTEMPLATE リソース定義を作成して、インストールします。

次のタスク

CICS は、最初の参照でプログラムを自動インストールします。あるいは、ユーザーは PROGRAM リソース定義を作成して、インストールできます。

DFHDHTL - プログラム・テンプレート prolog マクロおよび epilog マクロ

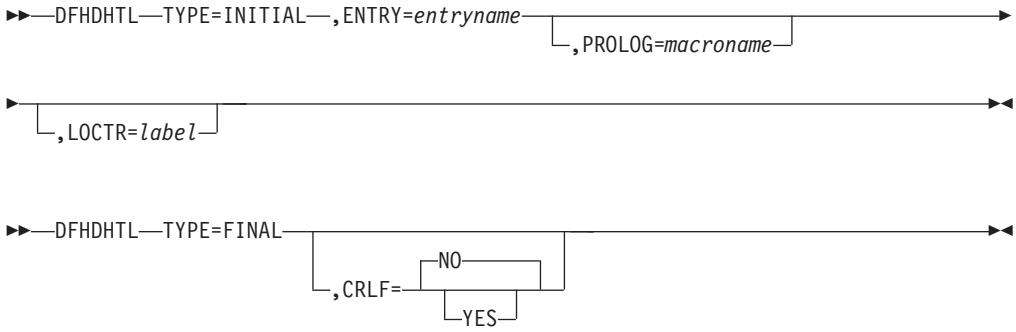
DFHDHTL マクロを使用して、(DOCTEMPLATE リソース定義の属性 PROGRAM と共に) CICS プログラムに含まれる文書テンプレートの prolog および epilog を生成します。

これらのテンプレートは、モジュールのエントリー・ポイントと終わりとの間の、ロード・モジュールの領域を占有するものとして定義されます。ただし、テンプレート・プログラムのソースが 8 バイトの倍数を正確に生成しない場合、リンケージ・エディターまたはバインド・プログラムが、偽のバイナリー文字を挿入する場合があります。このバイナリー文字が Web ブラウザーに送信された場合、予測不

能な結果を引き起こします。DFHDHTL マクロを使用すると、テンプレート・ロード・モジュールが、常に正確に 8 バイトの倍数になります。

TYPE=INITIAL により作成された prolog は、ユーザー指定マクロ (デフォルトは DFHVM) を起動することによりモジュールの目印を生成し、また、テンプレートの本文を追跡する新しいロケーション・カウンター (LOCTR) を生成します。

TYPE=FINAL により作成された epilog は、それまでのテンプレートの長さを計算して、その値が奇数の場合は、単一のブランク文字を生成します。このことにより、エントリー・ポイントが必ずハーフワード境界上になります。epilog は、prolog のオリジナルの LOCTR に戻すことによりモジュール・サイズを切り上げて、モジュールの長さを 8 バイトの倍数にするため、エントリー・ポイントの前に十分な長さの文字を生成します。



TYPE={INITIAL | FINAL}
prolog (INITIAL) または epilog (FINAL) を生成するかどうかを指定します。

ENTRY=*entryname*
モジュールの入り口のラベルを指定します。この地点でテンプレートが開始します。

PROLOG=*macroname*
prolog でモジュール識別データを生成するために起動するマクロを指定します。デフォルトは DFHVM です。

LOCTR=*label*
生成されたテンプレートと prolog コードを分離する内部生成ロケーション・カウンター (LOCTR) ステートメントにラベルを付けます。デフォルトは内部生成名です。

CRLF=NO | YES
epilog コードが終了の CRLF (復帰改行) で終了するかどうかを指定します。デフォルトは NO です。

出口プログラムのテンプレート

テンプレートの内容が動的に変更される場合、または CICS 以外のリソース (例えば、DB2) からテンプレートの内容を取得する場合、出口プログラムが最も適切です。

アプリケーション・プログラムが、出口プログラムで作成されると定義されたテンプレートを要求する場合、CICS は指定されたプログラムを呼び出し、連絡域を渡します。

プログラムの出口パラメーター・リストに指定した場合にのみ、出口プログラムに含まれるテンプレートが CICS によりキャッシュに入れられます。デフォルトでは、テンプレートはキャッシュに入れられません。出口プログラムがキャッシングを指定した場合、キャッシュ内のコピーは、文書テンプレートがアプリケーションにより初めて参照されたときに作成されます。

SET DOCTEMPLATE NEWCOPY コマンドを使用して、出口プログラムの新しいコピーをフェーズできます。このコマンドは、指定された出口プログラムの SET PROGRAM PHASEIN に相当するものです。このコマンドを実行すると、CICS はキャッシュ内の文書テンプレートのコピーを削除して、プログラムの新しいコピーをフェーズし、出口プログラムでキャッシングが指定されている場合は、キャッシュ内に新しい文書テンプレートのコピーを作成します。リフレッシュされた出口プログラムは、キャッシングを行う必要があるかどうかに対して異なる設定を指定でき、CICS は変更を引き継ぎます。

出口プログラムでのテンプレートの連絡域

文書テンプレートを提供する出口プログラムの連絡域は、以下のコピーブックによりマップされます。

- DFHDHTXD (アセンブラー)
- DFHDHTXH (C)
- DFHDHTXL (PL/I)
- DFHDHTXO (COBOL)

連絡域には以下のフィールドが含まれます。

dhtx_template_name_ptr
要求されているテンプレートの名前 (48 文字まで) を指すポインターが含まれます。

dhtx_buffer_ptr
出口プログラムがテンプレートを返す CICS 提供のバッファーのポインターが含まれます。

dhtx_buffer_len
(フルワード・バイナリー。) 出口プログラムがテンプレートを返す CICS 提供のバッファーの長さが含まれます。

dhtx_message_len
(フルワード・バイナリー。) このフィールドを使用して、出口プログラムがテンプレートを戻せないときに出されるメッセージの長さを返します。メッセージがない場合は、値ゼロが返されます。

dhtx_message_ptr
このフィールドを使用して、出口プログラムが失敗した理由を説明したメッセージのポインターを返します。CICS は、CSDH 一時データ宛先にこのメッセージを書き込みます。メッセージがない場合は、値ゼロが返されます。

dhtx_template_len
(フルワード・バイナリー。) このフィールドを使用して、テンプレートの実際の長さを返します。

dhtx_append_crlf
文字「1」(追加する) または「0」(追加しない) を使用して、各行の終わりに復帰文字や改行文字を追加するかどうかを指定します。

dhtx_return_code
(フルワード・バイナリー。) このフィールドを使用して、出口プログラムが正常にテンプレートを返したかどうかを示します。

- 戻りコード 0 は、出口がテンプレートを返したことを示します。
- 戻りコード 8 は、出口がテンプレートを返さなかったことを示します。この場合、CICS はアプリケーション・プログラム内に TEMPLATERR 条件を起こします。

dhtx_cache_response
文字「1」(キャッシュする) または「0」(キャッシュしない) を使用して、出口プログラムからの出力を CICS 文書ハンドラーによりキャッシュに入れる必要があるかどうかを指定します。dhtx_cache_response の値は「0」に初期設定されるので、出口でこの値を変更しなければ、デフォルトのアクションとしては、出口プログラムの応答をキャッシュに入れません。

文書テンプレートがキャッシュに入れられる場合、後続の要求はキャッシュ内のコピーを受け取ります。キャッシュ内のコピーが使用可能な限り、EXEC CICS SET DOCTEMPLATE NEWCOPY コマンドが実行されて出口プログラムおよびキャッシュ内コピーがリフレッシュされるまで、出口プログラムが再び呼び出されることはありません。リフレッシュされた出口プログラムは、dhtx_cache_response に対して異なる値を指定でき、CICS は変更を引き継ぎます。

動的に変更されるテンプレートはキャッシュに入れないようにする必要がありますが、テンプレートが変更されない場合は、キャッシングにより要求のパフォーマンスが向上するので、キャッシングは適切です。

返されるテンプレートが dhtx_buffer_len より長い場合は、テンプレートが長さ dhtx_buffer_len に切り捨てられるため、出口プログラムは dhtx_template_len で必要とされる長さを設定する必要があります。その後、より大きなバッファーを持つ出口プログラムが呼び出されます。

出口プログラムが戻りコード 8 を設定すると、説明メッセージを返すことができます。このメッセージは、CSDH 一時データ宛先に書き込まれます。アドレスとメッセージの長さを、それぞれ dhtx_message_ptr と dhtx_message_len に返します。メッセージが含まれるストレージは、出口プログラムの呼び出し元にアクセスできなければなりません。例えば、出口プログラムは GETMAIN コマンドを実行して、メッセージのストレージを獲得します。CICS は、タスクが終了すると、コマンドに SHARED オプションが指定されていない限り、ストレージを解放します。

文書テンプレートの記号の使用

記号参照または #echo コマンドを、文書テンプレート内の、完成した文書のテキスト内で記号値を置換すべき場所で使用します。テンプレート内で #set コマンドを使用して、記号のデフォルト値を指定できます。

始める前に

記号の名前に使用できるのは、大文字と小文字、数字、およびドル ($)、下線 (_)、ハイフン (-)、番号記号 (#)、ピリオド (.)、アットマーク (@) の特殊文字に限られます。名前には大文字小文字の区別があるので、大文字と小文字は別物と判断されます。

- 記号名の最初にアンパーサンド (&)、最後にセミコロン (;) を使用して指定することにより、文書テンプレートに記号参照を含めることができます。 例えば、記号 ORDER_NUMBER は、テンプレートでは以下のように指定されます。

  ```
  Thank you! Your order number is &ORDER_NUMBER;.
  ```

- また、文書テンプレート内で #echo コマンドを使用して、記号を指定することもできます。 記号名をコマンドで指定しますが、アンパーサンドとセミコロンは使用しないでください。 例えば、記号 USER_NAME は、以下のように指定されます。

  ```
  Welcome to the site,
  <!--#echo var=USER_NAME-->!
  ```

- テンプレート内で #set コマンドを使用して、記号のデフォルト値を指定できます。 例えば、以下のように記号 USER_NAME を組み込んで、デフォルト値を設定できます。

  ```
  <!--#set var=USER_NAME value='New User'-->
  Welcome to the site,
  <!--#echo var=USER_NAME-->!
  ```

  または。以下のとおりです。

  ```
  <!--#set var=USER_NAME value='New User'-->
  Welcome to the site, &USER_NAME;!
  ```

  これらのテンプレートのいずれかが USER_NAME のデフォルト値と共に使用された場合、文書は以下のようになります。

  ```
  Welcome to the site, New User!
  ```

次のタスク

文書テンプレートを使用して文書を作成するアプリケーション・プログラムは、テンプレートを使用するときに、置換される記号の値を定義する必要があります。記号に対してデフォルト値を提供した場合、アプリケーションがその記号の値を定義しないとデフォルト値が使用されます。記号の値は、テンプレートを文書に挿入する前またはそのときに定義する必要があることに注意してください。置換された記号値は、テンプレートの挿入後には変更できません。

テンプレート・コマンドの埋め込み

CICS 文書ハンドラーは、テンプレートに埋め込むことのできる 3 つのコマンドを認識します。サポートされている 3 つのコマンドとは、#set、#echo および #include です。

埋め込みテンプレート・コマンドの構文

埋め込みテンプレート・コマンドは、サーバー・サイド・インクルード・コマンドの構文規則に従っています。サーバー・サイド・インクルード・コマンドは、まず左不等号括弧、感嘆符、ハイフン、ハイフン、番号記号 (ハッシュ) の文字で始まり、その後にコマンドが続きます。終わりは、ハイフン、ハイフン、右不等号括弧の文字になります。例:

```
<!--#command-->
```

サーバー・サイド・インクルードの開始および終了で使用できる文字は、コード・ページ 037 のみです。これ以外の場合、コマンドは無視されます。これらの文字シーケンスの 16 進数表示は、X'4C5A60607B' および X'60606E' です。

#echo コマンド

#echo コマンドは、テンプレートが文書に挿入される場合に置換する必要のある記号を識別します。例えば、この #echo コマンドは、コマンドの場所で文書に置換される記号 ASYM を識別します。

```
This is the <!--#echo var=ASYM--> symbol.
```

テンプレートが使用されると、#echo コマンドを含むストリングは、記号に対して定義された値によって、完全置換されます。その名前で定義された記号定義がない場合は、#echo コマンドは出力データに残されます。記号の値は、アプリケーション・プログラムにより定義されるか、またはテンプレートの #set コマンドによりデフォルト値として定義できます。

#set コマンド

#set コマンドは、記号の値を設定するために使用されます。記号のデフォルト値を設定する場合に役立ちます。例えば、この #set コマンドは、記号 ASYM の「最初」のデフォルト値を指定します。

```
<!--#set var=ASYM value='first'-->
```

#set コマンドは、別の #set コマンドを指定変更できます。同じ記号名に対して複数の #set コマンドをテンプレートに含めた場合、最後のコマンドが使用されます。

テンプレート中の #set コマンドは、適用する記号に既に DOCUMENT SET コマンドを使用して値が与えられている場合は、無視されます。DOCUMENT SET コマンドを使用して割り当てられた記号を変更するには、別の DOCUMENT SET コマンドを実行するしか方法はありません。

#set コマンドは、#echo コマンドと組み合わせて使用できます。また、アンパーサンド (&) で始まり、セミコロン (;) で終わる記号名により指定された記号参照に適用することもできます。

#include コマンド

#include コマンドを使えば、テンプレートを別のテンプレートに組み込むことができます。組み込みは最大で 32 段階まで許可されています。

例:

```
<!--#include template=templatename-->
```

templatename は、DOCTEMPLATE リソース定義で定義されたテンプレートの名前 (48 バイトの名前) です。テンプレート名は、二重引用符で囲むこともできます。

第 52 章 文書および文書テンプレートを使用したプログラミング

このセクションでは、アプリケーション・プログラムでの文書および文書テンプレートの使用方法について説明します。

文書の作成

DOCUMENT CREATE コマンドを使用して、空の文書、またはデータを含む文書のいずれかを作成できます。データとは、文字ストリング、バイナリー・データのブロック、文書テンプレート、またはデータのバッファーです。

このタスクについて

DOCUMENT CREATE コマンドには、16 バイトのデータ域を必要とする、必須の DOCTOKEN パラメーターがあります。文書ハンドラーは DOCTOKEN オペランドを使用してトークンを返します。このトークンは、以後の呼び出しで文書を識別するのに使用されます。

データを含む文書を作成するために、DOCUMENT CREATE コマンドでオプションを指定して、以下のことを実行できます。

- 文字ストリングを含める (TEXT オプション)。
- バイナリー・データのブロックを含める (BINARY オプション)。
- テンプレート名で指定される文書テンプレートを使用する (TEMPLATE オプション)。
- データのバッファーの内容を含める (FROM オプション)。
- 文書テンプレートの記号、または FROM オプションで指定された項目に値を指定する (SYMBOLLIST オプション)。

以下の例に、文書を作成するさまざまな方法を示します。

1. 空の文書を作成してトークンを返すには、DOCTOKEN オプションを指定した EXEC CICS DOCUMENT CREATE コマンドを使用します。 以下の例では、空のを作成し、16 文字の変数 MYDOC にトークンを返します。

   ```
   EXEC CICS DOCUMENT CREATE
           DOCTOKEN(MYDOC)
   ```

2. TEXT オプションを使用して、アプリケーション・プログラムにより指定される文字ストリングを含む文書を作成します。 例えば、DOCTEXT というストリング変数を定義して、*これは文書に追加するテキストの一例です* という意味に初期化する場合は、以下のコマンドを使用して、このテキスト・ストリングで構成される文書を作成します。

   ```
   EXEC CICS DOCUMENT CREATE
           DOCTOKEN(MYDOC1)
           TEXT(DOCTEXT)
           LENGTH(53)
   ```

   このストリングは変更されない文書に追加され、CICS は記号置換を実行しません。

© Copyright IBM Corp. 1989, 2009

3. BINARY オプションを使用してバイナリー・データを含む文書を作成します。この操作では、データが送られるときにコード・ページ変換は実行されません。以下の例では、バイナリー・データとしてのデータ域の内容から構成される文書を作成します。

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(MYDOC2)
          BINARY(DATA-AREA)
```

CICS はこのデータに対して記号置換を行わず、データにマークを付けるので、文書を受け取り側に送ったときに、クライアント・コード・ページに変換されません。

4. TEMPLATE オプションを使用して文書を作成します。この文書は、DOCTEMPLATE リソース定義を使用して CICS に定義した文書テンプレートを使用して作成されます。

   a. TEMPLATENAME などの 48 バイトの変数を定義して、DOCTEMPLATE リソース定義の TEMPLATENAME 属性で指定されているテンプレートの 48 文字の名前の値に初期設定します。

   b. 文書テンプレートに記号が含まれていない場合、または記号のデフォルト値を使用する場合は、SYMBOLLIST オプションを指定しないで DOCUMENT CREATE コマンドを使用できます。 例:

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(MYDOC3)
          TEMPLATE(TEMPLATENAME)
```

   記号置換の値は、文書テンプレートが文書に配置される前、またはそのときにのみ指定できることに注意してください。置換された記号値は、テンプレートの挿入後には変更できません。

   c. 文書テンプレートの記号の値を設定する場合、SYMBOLLIST オプションを指定した DOCUMENT CREATE コマンドを使用します。 例:

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(MYDOC3)
          TEMPLATE(TEMPLATENAME)
          SYMBOLLIST('ORDER_NUMBER=0012345')
          LISTLENGTH(20)
```

5. FROM オプションを使用して、データのバッファーを使用した文書を作成します。 データのバッファーには、文書テンプレートに含まれる記号参照と同じ方法で置換される記号参照を含めることができます。 例:

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(MYDOC4)
          FROM(BUFFER)
          SYMBOLLIST('ORDER_NUMBER=0012345')
          LENGTH(LEN)
```

記号値の定義

DOCUMENT SET コマンドまたは DOCUMENT CREATE コマンドを使用して、アプリケーションは文書テンプレートの記号の値を定義できます。記号値は、DOCUMENT CREATE コマンドまたは DOCUMENT INSERT コマンドのいずれかにより、テンプレートを使用するときに置換されます。

このタスクについて

DOCUMENT SET コマンドで SYMBOL および VALUE オプションを使用して、個々の記号を定義できます。または、DOCUMENT CREATE または DOCUMENT SET コマンドで SYMBOLLIST オプションを使用して、単一コマンドで複数の記号を定義できます。デフォルトではこの記号リストの記号の分離文字はアンパーサンドですが、コマンドの DELIMITER オプションを使用してこの分離文字を指定変更できます。

アプリケーションを設計する際には、以下の点に注意してください。

- 記号の値は、文書テンプレートを文書に挿入する前またはそのときに定義する必要があります。置換された記号値は、テンプレートの挿入後には変更できません。
- 文書テンプレートの各記号を定義する必要があります。文書テンプレートの記号がアプリケーション・プログラムまたは #set コマンドのいずれかにより定義されていない場合、その記号に対する記号置換は実行されず、完成した文書には記号名または記号を指定する #echo コマンドが含まれます。
- DOCUMENT SET コマンドを使用したアプリケーションが提供する記号定義は、#set コマンドによりその記号に対して提供されたデフォルト値を指定変更します。
- DOCUMENT SET コマンドを使用して値を割り当てた記号は、別の DOCUMENT SET コマンドを実行して変更できます。文書テンプレートを文書に挿入した後で記号を変更した場合、新しい値が記号テーブルに書き込まれ、次にテンプレートが文書に挿入されると、その新しい値が使用されます。この変更によって、文書にすでに挿入されている値が影響を受けることはありません。

記号名、記号値の特殊文字とスペース、および記号リストの分離文字について詳しくは、704 ページの『記号および記号リストの指定に関する規則』を参照してください。

1. 個別の記号を定義するには、DOCUMENT SET コマンドに SYMBOL および VALUE オプションを指定して使用します。 SYMBOL オプションは記号の名前を指定し、VALUE オプションはその記号の値を指定します。704 ページの『記号および記号リストの指定に関する規則』に記載されている規則に従います。このトピックでは、UNESCAPED オプションの影響について説明します。
   a. 文書テンプレートを文書に配置する DOCUMENT INSERT コマンドを実行する**前**に、DOCUMENT SET コマンドを実行するようにしてください。
   b. DOCUMENT SET コマンドを使用して記号値を設定する必要がある場合は、DOCUMENT CREATE TEMPLATE コマンドを使用できません。代わりに、まず空の文書を作成してから、DOCUMENT SET コマンドを使用して記号値を定義し、次に DOCUMENT INSERT コマンドを使用して記号参照を含むテンプレートを挿入します。
2. 複数の記号定義を含む記号リストを提供するには、DOCUMENT CREATE または DOCUMENT SET コマンドの SYMBOLLIST オプションを使用します。 DOCUMENT CREATE で TEMPLATE および SYMBOLLIST を一緒に使用した場合、記号リストからの記号は、テンプレートが文書に追加されるときに解決されます。 DOCUMENT SET コマンドを使用している場合、文書テンプレートを

文書に配置する DOCUMENT INSERT コマンドを実行する**前**に、この DOCUMENT SET コマンドを実行するようにしてください。

a. デフォルトの記号の分離文字 **&** (アンパーサンド) が、記号値で使用するのに適切ではない場合は、DELIMITER オプションを使用して代わりの文字を指定します。 『記号および記号リストの指定に関する規則』に記載されている規則に従います。

b. SYMBOLLIST オプションを使用して、記号リストを含むバッファーを指定します。 選択した記号分離文字で分離して、各記号定義を指定します。 684 ページの『記号および記号リスト』 に、記号リストの例を示します。『記号および記号リストの指定に関する規則』に記載されている規則に従います。このトピックでは、UNESCAPED オプションの影響について説明します。

c. LISTLENGTH オプションを使用して、記号リストの長さを指定します。 ご使用のアプリケーションによっては、記号の値を定義するたびに記号リストの正確なリストの長さを指定する代わりに、記号リストの固定リスト長を提供する、LISTLENGTH オプションの永続値を選択する方が便利な場合があります。そのためには、以下点に注意します。

   - 選択する固定リスト長は、提供する予定の記号リストの最大長を含めるのに十分な長さにする必要があります。
   - 修正したリストの長さが、指定した記号の実際の長さより長い場合は、`'&END='` などのダミーの記号を余分に記号リストの最後に含めます。これにより、リストの最後の記号の値の末尾スペースまたは予測不能な文字の可能性が回避されます。テンプレートや文書には、このダミーの記号を組み込まないでください。末尾スペースや予測不能な文字は、ダミーの記号に割り当てられ、文書には表示されません。

タスクの結果

定義した記号は、記号テーブルに書き込まれます。DOCUMENT CREATE コマンドで記号リストを定義している場合、記号置換が即座に実行され、文書テンプレートおよび指定された記号を使用して文書が作成されます。DOCUMENT SET コマンドを使用して個別の記号または記号リストを定義する場合、その文書テンプレートを使用して DOCUMENT CREATE または DOCUMENT INSERT コマンドを実行した場合に、記号テーブルに書き込んだ値が記号置換で使用されます。

記号および記号リストの指定に関する規則

記号にはそれぞれ名前と値があります。個別または記号リスト内のいずれかの記号名の選択、および記号の値の指定については、以下の規則に従います。

記号名

記号の名前に使用できるのは、大文字と小文字、数字、およびドル ($)、下線 (_)、ハイフン (-)、番号記号 (#)、ピリオド (.)、アットマーク (@) の特殊文字に限られます。名前には大文字小文字の区別があるので、大文字と小文字は別物と判断されます。

記号を文書テンプレートに組み込むには、最初にアンパーサンド (&)、最後にセミコロン (;) を使用する記号名である記号参照として指定できます。または、記号名

は、#echo コマンドを使用して指定することもできます。アプリケーション内で記号の名前を指定する場合、アンパーサンドおよびセミコロンは使用しないでください。例えば、テンプレートの記号参照 &mytitle; は、記号リストにある記号名 *mytitle* に対応します。

記号リストの分離文字

DOCUMENT CREATE コマンドおよび DOCUMENT SET コマンドの SYMBOLLIST オプションでは、1 バイトの分離文字を使用した 1 つ以上の定義で構成される文字ストリングを指定します。デフォルトではこの記号の分離文字はアンパーサンドですが、コマンドの DELIMITER オプションを使用してこの分離文字を指定変更できます。記号の分離文字を記号値内で使用する場合、特殊な処理が必要になるため、記号リストの記号値内で使用されない記号の分離文字を選択するようにします。印刷できない文字も使用できます。

許可されない DELIMITER 値がいくつかあります。使用できない値は次のとおりです。

- ヌル (バイナリー X'00')
- シフトイン (バイナリー X'0E')
- シフトアウト (バイナリー X'0F')
- スペース (バイナリー X'40')
- 正符号 (バイナリー X'4E')
- コロン (バイナリー X'7A')
- 等符合 (バイナリー X'7E')
- % 記号 (バイナリー X'6C')
- バックスラッシュ (バイナリー X'E0')

記号値の特殊文字

記号の値には、任意の文字を含めることができます。ただし、記号値に以下の文字を含める必要がある場合には、特別な処理が必要です。

- 正符号 (+)。
- パーセント記号 (%)。
- 等号 (=)。
- 記号リストにおける記号分離文字として使用した文字。この文字に対する特別な処理は、その記号定義が記号リスト内で提供されている場合にのみ必要です。DOCUMENT SET コマンドを使用して、SYMBOL および VALUE オプションで個々の記号値を設定している場合には、特別な処理は適用されません。

記号値では、エスケープ・シーケンスを使用して特殊な意味を持つこれらの文字などを含めることができます。エスケープ・シーケンスは、パーセント記号、および後続の 2 つの 16 進数字 (つまり、0 から 9、a から f、および A から F) から構成されます。記号値を記号テーブルに書き込む場合、値に続く % 記号および 2 つの 16 進数字は、その 2 つの数字で示される単一の ASCII 文字に相当する EBCDIC で置き換えられます。

役に立つ一部の組み合わせを 表 38 に示します。

表 38. 特殊シンボルを表すために使用できるエスケープ・シーケンス

文字	エスケープ・シーケンス
正符号 (+)	%2B
パーセント記号 (%)	%25
等号 =	%3D
アンパーサンド & (デフォルトの記号の分離文字)	%26

パーセント記号に続く文字が 2 つの有効な 16 進数字でない場合は、パーセント記号およびそれに続く文字を、記号リストに表示される通りに記号テーブルに書き込みます。

エスケープ・シーケンスを使用しない場合は、DOCUMENT CREATE コマンドまたは DOCUMENT SET コマンドで UNESCAPED オプションを指定できます。このオプションを指定すると変換は実行されず、記号値は入力したとおりに記号テーブルに書き込まれます。

ただし、UNESCAPED オプションを使用すると、記号リストの記号値内に、記号の分離文字として使用した文字を組み込むことは**できません**。UNESCAPED を使用する場合は、記号の値に使用することのない記号分離文字を選択してください。あるいは、DOCUMENT SET コマンドで SYMBOL および VALUE オプションを指定して、記号の区切り文字として使用した文字を含む記号値を指定することができます。これは、VALUE オプションで使用される記号の区切り文字には、特殊な意味がないためです。

記号値のスペース

記号値にスペースを含める場合、CICS では以下の表記を使用できます。

- スペース文字。
- % 記号および後続の 16 進数字 20 (%20)。
- 正符号。

記号値が記号テーブルに書き込まれると、これらの表記はスペースとして解釈されます。これにより、正符号を使用するコンテンツ・タイプ **application/x-www-form-urlencoded** の HTML 仕様を拡張できます。

ただし、記号リストまたは VALUE オプションに含まれている記号値が CICS によってアンエスケープされないようにするために、UNESCAPED オプションが使用されている場合は、正符号またはエスケープ・シーケンス %20 を使用してスペース文字を示すことは**できません**。UNESCAPED オプションが使用されていると正符号がスペースに変換されないため、このような場合はスペース文字を使用してスペースを示す必要があります。

例: エスケープ・シーケンスを使用しない記号の定義

以下の例に、エスケープ・シーケンスを使用せずに記号値を文書ハンドラーに渡す方法を示します。このリストの記号値には、アンエスケープ処理が実行されない埋め込みの正符号、% 記号、およびアンパーサンドが含まれます。

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(ATOKEN)
          DELIMITER('!')
          SYMBOLLIST('COMPANY=BLOGGS & SON!ORDER=NUTS+BOLTS')
          LISTLENGTH(37)
          UNESCAPED
```

この例では、記号 COMPANY は、「BLOGGS & SON」という値を持ち、記号 ORDER は、「NUTS+BOLTS」という値になっています。

アンパーサンド以外の文字を記号の区切り文字として使用することは、「BLOGGS & SON」でアンパーサンドを使用できることを意味します。この例で使用される記号の区切り文字は「!」です。ただし、記号値に表示されない非印刷文字を使用することを推奨します。

UNESCAPED オプションを使用することによって、「NUTS+BOLTS」の正符号 (+) はスペースに変換されることはありません。UNESCAPED オプションが使用されているため、正符号ではなくスペース文字を使用して、記号値「BLOGGS & SON」でスペースが必要な場所を示す必要があります。これは、データが、コンテンツ・タイプ **application/x-www-form-urlencoded** の仕様には準拠しなくなることを意味します。

文書へのデータの追加

文書の作成の完了後、1 つ以上の DOCUMENT INSERT コマンドを実行することにより内容を拡張できます。テキスト、バイナリー・データ、データのバッファー、文書テンプレート、または記号の値を挿入できます。また、文書にブックマークを挿入して、これらを使用して後の挿入のために位置を示すこともできます。

このタスクについて

DOCUMENT INSERT コマンドでオプションを指定して、以下のことを実行できます。

- 文字ストリングの挿入 (TEXT オプション)。
- バイナリー・データのブロックの挿入 (BINARY オプション)。
- テンプレート名で指定される文書テンプレートの挿入 (TEMPLATE オプション)。
- データのバッファーの内容の挿入 (FROM オプション)。
- 記号テーブルからの名前付き記号の値の挿入 (SYMBOL オプション)。

デフォルトでは、指定したオブジェクトは、文書の終わりに追加されます。文書の途中にデータを挿入するために、1 つ以上のブックマークをセットアップできます。ブックマークによって、アプリケーションはデータ・ブロックを任意のシーケンスで挿入し、また、そのデータのシーケンスを文書の中で制御できます。

ブックマークは、データのブロックの間に配置されるラベルです。データのブロックの途中には配置できません。DOCUMENT INSERT コマンドを使用して、文書の構成中にブックマークを配置し、AT オプションを使用して、後続のオブジェクトを挿入するときにそのブックマークを指定できます。「TOP」という特殊なブックマークがすでに定義されており、このブックマークを使用するとデータを文書の最初に挿入できます。

1. TEXT オプションを使用して、アプリケーション・プログラムにより指定される文字ストリングを挿入します。 例:

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOC)
          TEXT('Sample line 1.')
          LENGTH(15)
```

   このストリングは変更されない文書に追加され、CICS は記号置換を実行しません。

2. BINARY オプションを使用してバイナリー・データのブロックを挿入します。 この操作では、データが送られるときにコード・ページ変換は実行されません。 例:

```
EXEC CICS DOCUMENT INSERT
         DOCTOKEN(MYDOC)
         BINARY(DATA-AREA)
```

   CICS はこのデータに対して記号置換を行わず、データにマークを付けるので、文書を受け取り側に送ったときに、クライアント・コード・ページに変換されません。

3. TEMPLATE オプションを使用して、文書テンプレートを挿入します。 文書テンプレートの記号の値を設定する場合、DOCUMENT SET コマンドを使用して個別の記号または記号リストを指定し、**その後で** DOCUMENT INSERT コマンドを実行します。 702 ページの『記号値の定義』 でこの方法を説明しています。

4. FROM オプションを使用してデータのバッファーを挿入します。 データのバッファーには、文書テンプレートに含まれる記号参照と同じ方法で置換される記号参照を含めることができます。

5. SYMBOL オプションを使用して、記号の値を挿入します。 SYMBOL は、値が記号テーブルに設定されている有効な記号の名前を指定します (DOCUMENT SET または DOCUMENT CREATE コマンドを使用)。文書ハンドラーは、記号に関連付けられた値を文書に挿入します。 記号に関連付けられている値を文書に挿入した後は、構成されている文書でその値を変更することはできないことに注意してください。後で記号に別の値を設定した場合、次に記号が文書に挿入される場合に、その新しい値が使用されます。この変更によって、文書にすでに挿入されている値が影響を受けることはありません。

6. BOOKMARK オプションを使用して、ブックマークを文書に挿入します。 ブックマークは、文書の構成中に配置され、後の段階で取得されるデータをアプリケーションが挿入するための挿入ポイントを示します。 ブックマークの名前の長さは、16 文字にする必要があります。埋め込みスペースを含まないようにする必要があり、16 文字未満の長さの名前を選択する場合、右側にブランクを埋め込む必要があります。 例えば、このコマンドのシーケンスでは、2 ブロックのテキストと 1 つのブックマークを持つ文書を作成します。

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(MYDOCBOOK)
          TEXT('Pre-bookmark text. ')
          LENGTH(19)

EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOCBOOK)
          BOOKMARK('ABookmark       ')
```

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOCBOOK)
          TEXT('Post-bookmark text. ')
          LENGTH(20)
```

7. 他の挿入オプションを指定して AT オプションを使用し、以前挿入されたブックマーク、または特殊な TOP ブックマークにオブジェクトを配置します。 例えば、以下のコマンドではサンプル・ブックマーク ABookmark にテキストを挿入します。

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOCBOOK)
          TEXT('Inserted at a bookmark. ')
          LENGTH(25)
          AT('ABookmark       ')
```

完了した文書は、以下のように表示されます。

```
Pre-bookmark text. Inserted at a bookmark. Post-bookmark text.
```

文書でのデータの置換

文書にブックマークを配置して、後で挿入により置換、または削除できるデータ域の範囲を定めることができます。この手法を使用すると、文書内にテキストまたは他のデータのデフォルト項目を提供できます。これは、置換で使用できるデータがないことをアプリケーションが認識した場合に使用できます。

このタスクについて

文書のデータのデフォルト項目をセットアップして、置換または削除するには、以下のようにします。

1. 文書の最初に使用するデータを指定して、文書を作成します。 この例では、文書はいくつかの初期テキストを使用して作成され、トークンが変数 MYDOCREP で返されます。

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(MYDOCREP)
          TEXT('Initial sample text. ')
          LENGTH(21)
```

2. DOCUMENT INSERT コマンドを使用して、最初のブックマークを指定します。

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOCREP)
          BOOKMARK('BMark1          ')
```

3. DOCUMENT INSERT コマンドを使用して、置換するテキストの項目や他のデータを指定します。

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOCREP)
          TEXT('Text to be replaced. ')
          LENGTH(21)
```

4. DOCUMENT INSERT コマンドを使用して、最後のブックマークを指定します。

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOCREP)
          BOOKMARK('BMark2          ')
```

5. DOCUMENT INSERT コマンドを使用して、文書の最後に使用するデータを追加します。

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(MYDOCREP)
          TEXT('Final sample text. ')
          LENGTH(19)
```

この時点でのこのサンプル文書の論理構造は、以下のとおりです。

```
Initial sample text. <BMark1>Text to be replaced. <BMark2>Final
sample text.
```

ブックマークの名前は文書に表示されません。

6. 2 つのブックマーク BMark1 と BMark2 の間のテキストを置換するには、AT および TO オプションを指定して DOCUMENT INSERT コマンドを使用します。

```
EXEC CICS DOCUMENT INSERT
          DOCTOKEN(ATOKEN)
          TEXT('Replacement Text. ')
          LENGTH(18)
          AT('BMark1          ')
          TO('BMark2          ')
```

ここで、サンプル文書は、以下のように表示されます。

```
Initial sample text. Replacement Text. Final
sample text.
```

7. 2 つのブックマークの間のテキストを削除するには、上記のように AT および TO オプションを指定して DOCUMENT INSERT コマンドを使用しますが、ヌル・ストリングを指定するために、LENGTH をゼロにして、TEXT または BINARY オプションを使用します。

文書の検索、保管、および再利用

アプリケーションが作成した文書は、それが作成された CICS タスクの有効期間の間のみ存在します。文書を再利用するには、アプリケーションでコピーを取得して保管する必要があります。

始める前に

以下のコマンド・シーケンスでは、文書の作成、取得、一時記憶キューへの保管、同一または別のアプリケーションによる文書としての再利用の方法を示しています。このコマンド・シーケンスを使用するには、アプリケーション・プログラムで以下の変数を定義し、初期化する必要があります。

- 文書トークンを保持するための、16 バイトの ATOKEN フィールド
- 検索済み文書を保持するための、20 バイトのバッファー DOCBUF
- 検索データ長を保持するための、FWORDLEN というフルワード 2 進数フィールド
- 一時記憶域の WRITE コマンドを保持するための、HWORDLEN というハーフワード 2 進数フィールド

このタスクについて

1. DOCUMENT CREATE コマンドを使用して、初期文書を作成します。

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(ATOKEN)
          TEXT('A sample document.')
          LENGTH(18)
```

取得した文書を保持するのに必要なバッファー・サイズをアプリケーションに計算させるために、文書のサイズを変更する文書コマンド (DOCUMENT CREATE および DOCUMENT INSERT コマンド) で、DOCSIZE オプションを使用できます。この値は、RETRIEVE コマンドを実行するときに、元のコード・ページの文書のコピー (制御情報を含む) を含めるのに必要なバッファーの最大サイズです。ただし、元の EBCDIC データより多くのバイトを必要とするエンコード方式 (例えば UTF-8) を CHARACTERSET オプションで指定した場合は、この最大サイズでは変換された文書を保管するのに不十分な場合があります。
DOCUMENT RETRIEVE にダミー・バッファーを指定し、MAXLENGTH をゼロにして実行し、LENGERR 条件を処理して返された LENGTH の値を使用することにより、バッファーを割り振る前に実際の文書の長さを判別できます。

2. 同じタスクで、DOCUMENT RETRIEVE コマンドを実行して、アプリケーション自身のバッファー内の文書のコピーを取得します。

```
EXEC CICS DOCUMENT RETRIEVE
          DOCTOKEN(ATOKEN)
          INTO(DOCBUF)
          LENGTH(FWORDLEN)
          MAXLENGTH(20)
```

デフォルトでは、文書を取得する場合、そのアプリケーション・バッファーに引き渡されるデータは、文書の正確なレプリカを再構成するのに必要な制御情報を含む形式で保管されます。CICS は文書の内容にタグを挿入して、ブックマークを識別し、コード・ページ変換を必要としないブロックの範囲を定めます。したがって、検索されたコピーから作成される文書は、元の文書とまったく同じです。元の文書を再作成する必要がない場合、DOCUMENT RETRIEVE コマンドを以下のように変更できます。

   a. 制御情報なしのコピーを要求するには、DATAONLY オプションを指定します。 このオプションを使用すると、CICS はすべての埋め込みタグを除外します。検索された文書には、ブックマークは含まれず、コード・ページ変換を必要としないブロックの範囲を定めるマーキングもありません。
   b. コピー全体を単一のクライアント・コード・ページに変換するには、CHARACTERSET オプションを指定します。

3. 一時記憶キューに文書を保管するには、以下のようにします。

```
EXEC CICS WRITEQ TS
          QUEUE('AQUEUE')
          FROM(DOCBUF)
          LENGTH(HWORDLEN)
```

4. 同一または別のアプリケーションで、保管データをアプリケーションのバッファーに読み込むには、以下のようにします。

```
EXEC CICS READQ TS
          QUEUE('AQUEUE')
          INTO(DOCBUF)
          LENGTH(HWORDLEN)
```

5. FROM オプションを指定して DOCUMENT CREATE コマンドを使用し、データ・バッファーの内容、つまり取得された文書を使用して新しい文書を作成します。

```
EXEC CICS DOCUMENT CREATE
          DOCTOKEN(ATOKEN)
          FROM(DOCBUF)
          LENGTH(FWORDLEN)
```

次のタスク

DOCUMENT RETRIEVE および DOCUMENT INSERT コマンドを使用して、文書全体を既存文書に挿入できます。以下の変数は、アプリケーション・プログラムで最初に定義し、初期化する必要があります。

- 検索する文書の文書トークンを含んでいる 16 バイトの RTOKEN フィールド
- 検索する文書の保持に十分な長さの DOCBUF バッファー
- 検索データ長を保持するための、RETRIEVLEN というフルワード・バイナリー・フィールド
- バッファーが受信可能な最大データを保持するための、MAXLEN という名前のフルワード・バイナリー・フィールド、すなわち、DOCBUF の長さ
- 挿入される文書の文書トークンを含んでいる 16 バイトの ITOKEN フィールド

次のコマンド・シーケンスは、ITOKEN が示している他の文書に挿入される RTOKEN によって示される文書を示しています。

```
EXEC CICS DOCUMENT RETRIEVE
          DOCTOKEN(RTOKEN)
          INTO(DOCBUF)
          LENGTH(RETRIEVLEN)
          MAXLENGTH(MAXLEN)

EXEC CICS DOCUMENT INSERT
          DOCTOKEN(ITOKEN)
          FROM(DOCBUF)
          LENGTH(RETRIEVLEN)
```

検索される文書は、DOCUMENT INSERT コマンドで指定された文書の最後に挿入され、すべての検索文書の制御情報は 2 番目の文書で示されます。DOCUMENT INSERT コマンドの LENGTH パラメーターは、DOCUMENT RETRIEVE コマンドから RETRIEVLEN フィールドに返された値と等しくなっている必要があります。

文書の削除

DOCUMENT DELETE コマンドを使用して、トランザクションの間に必要がなくなった文書を削除できます。コマンドを実行すると、文書に割り振られたストレージは、即座に解放されます。WEB CONVERSE、WEB SEND (クライアント)、および WEB SEND (サーバー) コマンドの DOCSTATUS(DOCDELETE) オプションでも、文書を削除できます。

このタスクについて

DOCUMENT DELETE、WEB CONVERSE、WEB SEND (クライアント)、および WEB SEND (サーバー) は、すべて DOCTOKEN を使用して、文書の 16 バイトのバイナリー・トークンを指定します。文書トークンは、EXEC CICS DOCUMENT API コマンドを使用して文書を作成する場合に返されます。

DOCUMENT DELETE コマンドを使用して文書を削除するには、以下のようにします。

1. 削除する文書の DOCTOKEN を指定します。例を以下に示します。

```
EXEC CICS DOCUMENT DELETE
          DOCTOKEN(MYDOC)
```

2. 文書が文書処理ドメインから削除され、ストレージが即座に解放されます。ACTION(EVENTUAL) がコマンドで指定されている場合、Web ドメインは文書のコピーを保存します。

WEB CONVERSE、WEB SEND (クライアント)、および WEB SEND (サーバー) で DOCSTATUS(DOCDELETE) オプションを指定することにより、文書を削除できます。このオプションにより、アプリケーションが CONVERSE または SEND コマンドを実行したときに、文書が必要なくなったことをアプリケーションが示すことができます。WEB SEND コマンドが完了すると、文書が文書処理ドメインおよび Web ドメインから削除され、ストレージが即座に解放されます。

コマンドで DOCSTATUS(NODOCDELETE) および ACTION(EVENTUAL) を指定して WEB SEND を実行すると、WEB RETRIEVE コマンドを使用して文書を取得できます。DOCSTATUS(DOCDELETE) オプションまたは ACTION(IMMEDIATE) オプションを使用すると、文書が Web ストレージから永続的に削除され、その文書は取得できません。文書を削除した後の文書の取得の制限について詳しくは、「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

第 13 部 名前付きカウンター・サーバー

名前付きカウンター・サーバーを理解して使用します。

© Copyright IBM Corp. 1989, 2009

第 53 章 概要: 名前付きカウンター・サーバー

CICS は、アプリケーション・プログラムが Parallel Sysplex (並列シスプレックス) 環境で使用する、固有のシーケンス番号を生成する機能を提供します。この機能は、名前付きカウンター・サーバーによって制御されます。このサーバーは、数字の各シーケンスを**名前付きカウンター**として保存します。

シーケンス番号が割り当てられるたびに、対応する名前付きカウンターが自動的に増分されます。デフォルトでは増分は 1 で、次の要求が順番に次の数字を取得するようになっています。EXEC CICS GET コマンドを使用して次の番号を要求するときに、増分を変えることができます。

この機能には、文書 (例えば注文書、送り状、発送メモ) に固有の番号を付けたり、カスタマー・ファイルでレコード番号を割り振るのに 1 ブロックの数字を確保するなど、いろいろな使い方があります。

単独の CICS 領域では、固有番号の割り振りを制御するために使用できる方式がいろいろあります。例えば、CICS 共通作業域 (CWA) を使用すれば、アプリケーション・プログラムごとに更新される数値を保管することができます。CWA 方式で問題なのは、CWA は CICS のアドレス・スペースに対して一意的であるため、同じアプリケーションを実行している他の領域で共有することができないという点です。CICS 共用データ・テーブルを使用すれば、このサービスを提供することができますが、CICS 領域はすべて、同じ MVS イメージに常駐しなければなりません。名前付きカウンター機能は、名前付きカウンターをカップリング・ファシリティーで保守し、シスプレックス内の個々の MVS イメージで稼働している名前付きカウンター・サーバー経由で、アクセスを提供することによって、他の方式が提示する共用上の問題をすべて解決します。これにより、並列シスプレックスによる CICS 領域はすべて、同一の名前付きカウンターに対するアクセス権を得ることが確実になります。

名前付きカウンター・サーバーを使用する場合は、(次のカウンター値に割り当てる) 通常の各要求は、単一のカップリング・ファシリティー・アクセスのみを必要とします。これにより、同じ目的でファイルを使用する場合に比較すると、パフォーマンスが大幅に向上します。この観点からすると、名前付きカウンター・サーバーのパフォーマンスは、カップリング・ファシリティー・データ・テーブルより優れています。これは、カップリング・ファシリティー・データ・テーブルを更新するには少なくとも 2 つのカップリング・ファシリティー・アクセスが必要とされるためです。ハードウェア構成によっては、毎秒何千もの名前付きカウンター・サーバー要求を簡単に作成できるはずです。

このセクションでは、以下について説明します。

- 718 ページの『名前付きカウンター・フィールド』
- 718 ページの『名前付きカウンター・プール』
- 721 ページの『第 54 章 名前付きカウンター EXEC インターフェースの使用』
- 723 ページの『第 55 章 名前付きカウンター呼び出しインターフェースの使用』

© Copyright IBM Corp. 1989, 2009

- 737 ページの『第 56 章 名前付きカウンターのリカバリー』

名前付きカウンター・フィールド

名前付きカウンターはそれぞれ、次の要素で構成されています。

カウンター名
この名前には最大 16 バイトまで使用可能であり、A ～ Z までの文字、0 ～ 9 までの数字、$、@、#、および _ で構成されます。名前が 16 バイトに満たない場合は、末尾ブランクを埋め込みます。

現行値 要求するアプリケーション・プログラムに割り当てられる、次の番号。

最小値 カウンターの最小値、および REWIND コマンドに応答して、サーバーがカウンターをリセットするときの数値を指定します。

最大値 カウンターが割り当てることのできる最大数を指定します。その後はカウンターを、REWIND コマンド (または自動的に WRAP オプション) によって、明示的にリセットしなければなりません。

すべての値は 8 バイト (ダブルワード) の 2 進数として、内部に保管されます。EXEC CICS インターフェースによって、値は、フルワードの符号付き 2 進数か、ダブルワードの符号なし 2 進数のいずれかとして使用することができます。これにより、ダブルワード・コマンドを使用して (721 ページの『第 54 章 名前付きカウンター EXEC インターフェースの使用』を参照)、名前付きカウンターを定義し、そのコマンドの符号付きフルワード・バージョンを使用して、サーバーから数字を要求する場合には、オーバーフロー条件を生じさせることができます。

名前付きカウンター・プール

名前付きカウンターは、名前付きカウンター・プールに保管されます。このプールは、カップリング・ファシリティーのリスト構造に常駐しています。各プールは、そのリスト構造がたとえ最小サイズの 256KB で定義されていても、最大で 1,000 の名前付きカウンターを保持することができます。

名前付きカウンター・プールは、プール用にカップリング・ファシリティー・リスト構造を定義し、次にプール用の最初の名前付きカウンター・サーバーを開始することによって、作成します。プール名は容量 1 ～ 8 バイトで、カウンター名と同じ文字セットで構成されています。プール名は許可された文字ならどれを使用して作成してもいいのですが、DFHNC*xxx* 形式の名前をお勧めします。

必要に合わせて別々のプールを作成することができます。実動 CICS 領域が使用する (例えば、DFHNCPRD という名前の) プールを作成し、テストまたは開発領域用には、(DFHNCTST や DFHNCDEV のような名前の) 別のプールを作成するということが可能です。アプリケーション・プログラムで論理プール名を使用できる方法、実行時に実際のプール名に対して論理プール名を決定する方法についての詳細は、719 ページの『名前付きカウンター・オプション・テーブル』を参照してください。

名前付きカウンター・サーバーのリスト構造の定義、および名前付きカウンター・サーバーの開始については、「*CICS System Definition Guide*」に説明されています。

名前付きカウンター・オプション・テーブル

POOL(*name*) パラメーターは、すべての EXEC CICS COUNTER および DCOUNTER コマンドでオプションです (詳細については、 721 ページの『第 54 章 名前付きカウンター EXEC インターフェースの使用』を参照してください)。 POOL パラメーターを指定する場合は、実際のプール名か論理プール名のいずれかを参照することができます。POOL パラメーターを指定しようと省略しようと、CICS は名前付きカウンター・オプション・テーブルを参照して、実際のプール名を決定します。このテーブルは、リンク・リストからロードされます。

名前付きカウンター・オプション・テーブル DFHNCOPT では、名前付きカウンター API コマンドが参照する実際のプール名を判別するためのメソッドがいくつか提供されています。「*CICS System Definition Guide*」には、このすべてについて説明されています。ここではまた、ユーザーが自分用のオプション・テーブルを作成するために使用できる、DFHNCO マクロについても述べています。

デフォルト・オプション・テーブルの POOLSEL パラメーターは、API の POOL(*name*) オプションと連携して動作します。デフォルト・オプション・テーブルは、ソース形式およびオブジェクト形式で提供されます。生成済みバージョンは *hlq*.SDFHLINK にあります。ソース・バージョンは *hlq*.SDFHSAMP ライブラリーで提供され (ここで、*hlq* は、CICS インストール時に設けられたライブラリー名の上位修飾子を表します)、以下の項目を含みます。

```
DFHNCO   POOLSEL=DFHNC*,POOL=YES
DFHNCO   POOL=
END      DFHNCOPT
```

デフォルトのオプション・テーブル項目は、次のような働きをします。

POOLSEL=DFHNC*

このプール選択パラメーターは、DFHNC という文字で始まる汎用論理プール名を定義します。名前付きカウンター API 要求が、この総称名と一致するプール名を指定すると、DFHNCO 項目の POOL= オペランドがそのプール名を判別します。デフォルトのテーブルではこれが POOL=YES になっているので、API コマンドの POOL(*name*) オプションで渡される名前は、実名と受け取られます。したがって、デフォルトのオプション・テーブルでは、DFHNC で始まる論理プール名がすべて実際のプール名であることを、条件として指定しています。

POOL=

デフォルト・テーブルにあるこの項目は、「デフォルト」項目です。 POOLSEL パラメーターが指定されていないので、デフォルトの値は POOLSEL=* です。これは、より明らかな一致が検出されない POOL パラメーター上の、任意の値に一致する値がとられることを意味します。したがって、任意の名前付きカウンター API は以下のことを要求します。

- DFHNC 以外のもので始まる POOL 値を指定する、あるいは
- POOL 名のパラメーターをまったく省略する

デフォルト・プールにマップされます (名前オペランドを省略する、POOL= オプション・テーブル・パラメーターによって示されます)。

NCPLDFT システム初期設定パラメーターを指定することにより、CICS 領域が使用するデフォルト・プール名を指定できます。NCPLDFT が省略される場合は、プール名のデフォルトは DFHNC001 です。

上記から、次の場合、ユーザーは独自のオプション・テーブルを作成する必要がなく、また名前付きカウンター API コマンドは、POOL オプションを指定する必要がないことがわかります。

- DFHNC*xxx* 形式のプール名を使用する、または
- CICS 領域は、NCPLDFT システム初期設定パラメーターによって定義可能な 1 つのプールのみを使用します。

注:

1. DFHNCOPT 名前付きカウンター・オプション・テーブルには接尾部はありません。CICS 領域は、MVS リンク・リストで最初に検出したテーブルをロードします。
2. CICS 領域と同じ MVS イメージ内では、アプリケーション・プログラムが使用する個々の名前付きカウンター・プールに対して、名前付きカウンター・サーバーが稼働していなければなりません。

第 54 章 名前付きカウンター EXEC インターフェースの使用

名前付きカウンター値はすべて、ダブルワード符号なし 2 進数として内部に保留されますが、CICS API は、フルワード (COUNTER) とダブルワード (DCOUNTER) の両方のコマンド・セットを提供します。ただし、この両セットを混在させることはできません。これらの EXEC CICS コマンドによって、名前付きカウンターで次のような操作を実行することができます。

DEFINE
新しい名前付きカウンターを定義し、最小値および最大値を設定し、カウンターを開始する現行番号を指定します。

DELETE
名前付きカウンターを、その名前付きカウンター・プールから削除します。

GET
最大数がまだ割り振られていない場合、名前付きカウンターから現行番号を入手します。

WRAP オプションを使用する: 直前の要求に対して最大数が割り振られている場合は、カウンターは限界状態にあり、WRAP オプションを指定しない限り、要求は失敗します。このオプションで、カウンターが限界状態になれば、定義された最小値に自動的にリセットするように指定します。リセット後は、最小値は現行番号として返され、カウンターは次の要求に備えて更新されます。

INCREMENT オプションを使用する: デフォルトでは、名前付きカウンターは、サーバーが現行番号を GET 要求に割り当てた後は、増分 1 で更新されます。一度に複数の番号が必要であれば、INCREMENT オプションを指定することができます。このオプションは、現行数字から、数字のブロックを効率的に保存するものです。例えば、INCREMENT(50) と指定すると、サーバーは 100 025 を返します。

- アプリケーション・プログラムは 100 025 から 100 074 を使用できます。
- 現行数字 (100 025) を 50 ごとに更新すると、次の要求に備える現行数字は 100 075 となります。

この例では、現行値を INCREMENT(50) オプションで更新しても、最大値を 1 以上超えることはないと想定しています。現行値と最大値プラス 1 の間の数字の範囲が指定された増分に満たなければ、さらに REDUCE オプションをも指定しない限り、要求は失敗します。

REDUCE オプションを使用する: 残りの数字の範囲が小さすぎて INCREMENT 値を満たすことができない (現行の数字が、最大値に近すぎる) という理由で要求が失敗することはないことを確認するためには、REDUCE オプションを指定します。 REDUCE オプションを指定すれば、サーバーは自動的に増分を調整して、残りのすべての数字を割り当て、カウンターを限界状態にすることができます。

WRAP および REDUCE オプションを使用する: 両方のオプションを指定しても、カウンターの状態によって、どちらか一方だけが有効になります。

© Copyright IBM Corp. 1989, 2009

- サーバーが GET 要求を受け取ったときに、カウンターが既に限界に達していれば、REDUCE オプションは効果がなく、WRAP オプションに従うことになります。
- サーバーが GET 要求を受け取ったときに、カウンターがまだ限界に達してはいないが、残っている範囲が増分するには小さすぎる場合は、REDUCE オプションに従うことになり、WRAP オプション効果がありません。

COMPAREMIN および COMPAREMAX オプションを使用する: これらのオプションを使用すれば、名前付きカウンターの GET (および UPDATE) 操作を、現行の数値が指定の範囲にあるか、あるいは指定された比較値の 1 つより大きいか、小さいかの条件をつけて行うことができます。

QUERY
現行値、最小値、最大値を取得するために、名前付きカウンターの照会を行います。複数の名前付きカウンター・コマンドを微細な方法で使用したり、シスプレックスのどこかで他のタスクが変更した QUERY コマンドで返される情報に、頼ったりすることはできません。CICS シスプレックス・ワイド ENQ 機能でさえ、ユーザー用のカウンターをロックすることはできません。なぜなら、名前付きカウンター CALL インターフェースを使用する、バッチ・アプリケーション・プログラムを使えば、名前付きカウンターにアクセスできるためです。ある操作を、現行値が一定の範囲内にあるか、あるいは一定の数字より大きいか、小さいかという条件によって変わるようにしたい場合は、要求に際して COMPAREMIN および COMPAREMAX パラメーターを使ってください。

REWIND
限界状態にある名前付きカウンターを、定義された最小値に巻き戻します。

UPDATE
アップデートするカウンターの現行値を、新規の現行値に更新します。例えば、データベース内の次の空きキーに、現行値を設定することができます。GET コマンドと同様、COMPAREMIN 値および COMPAREMAX 値を指定することによって、これを条件付きにすることができます。

第 55 章 名前付きカウンター呼び出しインターフェースの使用

CICS の名前付きカウンター API に加えて、CICS は呼び出しインターフェースを提供します。このインターフェースをバッチ・アプリケーションから使用して、同じ名前付きカウンターにアクセスすることができます。CICS とバッチ・プログラムの両方を持っていて、両方が同じ名前付きカウンターにアクセスして、指定された範囲から固有の数字を獲得する必要がある場合には、このことは重要です。呼び出しインターフェースは、CICS サービスに依存していません。そのため、どのリリースの CICS で実行されているアプリケーションでも、使用できます。

名前付きカウンター呼び出しインターフェースでは、CICS コマンド・レベル API を使用しません。したがって、システム初期設定パラメーター CMDPROT=YES には影響されません。このインターフェースが、ユーザー・キーで稼働している CICS アプリケーション・プログラムから呼び出された場合は、要求の処理中に CICS キーに切り替えられますが、CICS はプログラムに出力パラメーター・フィールドへの書き込みアクセスがあるかどうかは検査しません。

アプリケーション領域による、特定のプールをアドレッシングする最初の要求は、そのプールのサーバーへの接続を自動的に確立します。この接続は、現行の MVS TCB (CICS の場合、準再入可能 (QR) TCB) に関連付けられており、通常、ジョブ終了時に TCB が終了するまで持続します。この接続は、この接続を確立した TCB からしか使用できません。他の TCB から発行した要求は、サーバーへの別の接続を確立します。

注: 名前付きカウンター・サーバー・インターフェースは、MVS の名前およびトークンのサービスを内部的に使用します。その結果、名前付きカウンター・インターフェースを使用するジョブは、(APAR OW06685 に説明されているように) MVS チェックポイントおよびリスタートの各サービスを使用できません。

アプリケーション・プログラミングに関する考慮事項

このタスクについて

名前付き呼び出し可能インターフェースを使用するには、以下のようにします。

1. アプリケーション・プログラムに、そのアプリケーション・プログラム言語のパラメーター・リスト定義を定義する適切なコピーブックが組み込まれていることを確認します。コピーブックは、機能コードの記号定数を定義してコードを返します。また、高水準言語の呼び出し可能入り口点の定義も行います。サンプル集名は、DFHNC*xxx* という形式になっており、*xxx* は、以下のようにプログラム言語を示します。

   ```
   ASM or EQU for Assembler
   C for C/C++
   COB for COBOL
   PLI for PL/I
   ```

2. アプリケーション・プログラムが呼び出し可能インターフェースのリンケージ・ルーチン DFHNCTR でリンク・エディットされていることを確認します。

© Copyright IBM Corp. 1989, 2009

3. 名前付きカウンター・サーバー・インターフェース・モジュール DFHNCIF、およびオプション・テーブル DFHNCOPT が CICS 領域で使用可能であることを確認します。つまり、これらのオブジェクトは、STEPLIB ライブラリー内、リンク・リスト・ライブラリー内、または LPA 内になければなりません。ユーザー・キーで稼働する CICS アプリケーション・プログラムをサポートするには、DFHNCIF が APF 許可を受けているライブラリーからロードされなければなりません。デフォルト・オプション・テーブルおよび名前付きカウンター・サーバー・インターフェース・モジュールは、CICSTS41.CICS.SDFHLINK に提供されています。

CICS は、サポートされているすべての言語のコピーブックを提供しています。

アセンブラー

標準アセンブラーの名前付きカウンター・インターフェース定義は、サンプル集 DFHNCASM に提供されています。定数 CSECT 域内で COPY DFHNCASM を使用して、これらをアプリケーション・プログラムに組み込みます。記号値は、NC_*name* DC F'*nnn*' の形式で静的フルワード定数として定義されます。例:

```
NC_BROWSE_NEXT DC F'7'
```

サンプル集 DFHNCEQU に、記号等価値として定義の代替セットが提供されています。これらの記号は、静的定数との競合を避けるため、すべて NC_EQU_*name* の形式になっています。これらの等価値は、機能コードまたは戻りコードの比較に使用される際、アドレス定数値として使用される必要があります。これにより、例えば機能コード NC_ASSIGN を =A(NC_EQU_ASSIGN) への参照に置き換えることができます。

名前付きカウンター・インターフェースを呼び出す、アセンブラー・バージョンの構文を次に示します。

```
CALL DFHNCTR,(function,return_code,pool_selector,counter_name,        X
              value_length,current_value,minimum_value,maximum_value,  X
              counter_options,update_value,compare_min,compare_max),VL
```

CALL マクロは、以下の例にあるように、VL オプションを指定して、リストの終わりの指示を設定しなければなりません。

```
CALL DFHNCTR,(NC_ASSIGN,RC,POOL,NAME,CTRLEN,CTR),VL
```

C/C++

C/C++ の名前付きカウンター・インターフェース定義は、ヘッダー・ファイル DFHNCC に提供されています。記号定数名は大文字です。機能名は、小文字で `dfhnctr` です。

COBOL

COBOL の名前付きカウンター・インターフェース定義は、コピーブック DFHNCCOB に提供されています。

COBOL では、名前に下線を入れることはできないため、サンプル集 DFHNCCOB に提供されている記号名は、下線の代わりにハイフンを使用しています (例えば、NC-ASSIGN および NC-COUNTER-AT-LIMIT など)。

RETURN-CODE 特殊レジスターは各呼び出しにより設定されます。これは、プログラムが終了する前に再度明示的に設定されない場合は、プログラム全体の戻りコードに影響します。

PL/I

PL/I の名前付きカウンター・インターフェース定義は、インクルード・ファイル DFHNCPLI に提供されています。

構文規則

名前付きカウンター呼び出しの構文を以下に示します。

```
CALL DFHNCTR(function,return_code,pool_selector,counter_name,
             value_length,current_value,minimum_value,maximum_value,
             counter_options,update_value,compare_min,compare_max);
```

図 134. DFHNCTR 呼び出し構文 -PL/I の説明図

注:

1. 名前付きカウンターを参照するすべての機能には、少なくとも最初の 4 つのパラメーターが必要ですが、残りのパラメーターはオプションです。また、後続の未使用パラメーターは省略可能です。

   組み込みのオプション・パラメーターを使用したくない場合は、デフォルト値を指定するか、または省略されたパラメーターのヌル・アドレスがパラメーター・リストに含まれているか確認します。オプション・パラメーターを省略する呼び出しの例については、732 ページの『ヌル・パラメーターを持つ DFHNCTR 呼び出しの例』を参照してください。

2. NC_FINISH 機能は、最初の 3 つのパラメーターのみを必要とします。

function

以下の記号定数のいずれか 1 つを使用して、実行する機能を 32 ビット整数として指定します。

NC_CREATE

current_value、*minimum_value*、*maximum_value*、*update_value* および *counter_options* の各パラメーターに指定した初期値、範囲の制限、およびデフォルト・オプションを使用し、新規の名前付きカウンターを作成します。

オプションの値パラメーターを省略した場合は、省略された値のデフォルトを使用して新規の名前付きカウンターが作成されます。例えば、すべてのオプション・パラメーターを省略した場合、カウンターは、初期値が 0、最小値が 0、および最大値が高位値 (ダブルワード・フィールドが X'FF' で埋められる) で作成されます。

NC_ASSIGN

名前付きカウンターの現行値を割り当て、次の要求に備えてその値を増分します。割り当てられた数値が、カウンターに指定された最大数と等しい場合、最終的に、最大数より 1 大きい値に増分されます。これにより、名前付きカウンターの後続の NC_ASSIGN 要求は、カウンターが NC_REWIND 機能を使用してリセットされるか、または NC_WRAP カウンター・オプション (*counter_options* パラメーターを参照) によって自動的に巻き返されるまで、(NC_COUNTER_AT_LIMIT で) すべて失敗します。

この操作には、*compare_min* および *compare_max* パラメーターを使用して、名前付きカウンターの現行値に条件テストを組み込むことができます。

呼び出しパラメーター・リストにこれらのフィールドを指定し、要求が成功すると、サーバーは、最小値および最大値を返します。

NC_ASSIGN 要求で *counter_options* パラメーターを使用して、NC_CREATE 要求によって設定されたカウンター・オプションを指定変更できます。

update_value パラメーターを使用して、増分がこの要求では名前付きカウンターにのみ使用されるように指定できます (デフォルトの増分は 1 です)。これにより、ユーザーは、単一の要求で、ある範囲の数値を取得できるようになります。詳細については、*update_value* パラメーターの説明を参照してください。

名前付きカウンターは、現行値が割り当てられた後、*update_value* の分ずつ増分されるので、注意してください。例:

現行値が 109 で、*update_value* が 25 を指定する場合

名前付きカウンター・サーバーは 109 を返して、次の NC_ASSIGN 要求に備えて現行値を 134 に設定し、109 から 133 までの範囲で数値を効果的に割り当てます。増分は、名前付きカウンター用に設定された最小値および最大値によって決定される、ゼロから限界までの間の任意の値が可能です。したがって、増分限界は ((*maximum_value* + 1) − *minimum value*) です。増分がゼロの場合、NC_ASSIGN が、比較オプションを除き、NC_INQUIRE と同じ動作をする原因となります。

増分が 1 より大きく、名前付きカウンターが上限に近い場合、サーバーは、現行の数値を、指定された増分の全体値ずつ増分できない場合があります。この状態は、カウンターを増分すると最大値に 1 を加えた値を超える場合に発生します。ユーザーは *counter_options* NC_NOREDUCE | NC_REDUCE、および NC_NOWRAP | NC_WRAP を使用し、名前付きカウンター・サーバーがこの状態で実行するアクションを制御します。これらのオプションの操作について詳細については、*counter_options* パラメーターを参照してください。

NC_BROWSE_FIRST
指定された名前よりカウントが大きいか等しい名前の、最初の名前付きカウンターの詳細を返し、それに応じて *counter_name* フィールドを更新します。

NC_BROWSE_NEXT
指定された名前よりカウントが大きい名前の、次の名前付きカウンターの詳細を返し、それに応じて *counter_name* フィールドを更新します。

NC_DELETE
指定した名前付きカウンターを削除します。パラメーター・リストにこれらのフィールドを指定し、要求が成功すると、サーバーは *current_value*、*minimum_value*、*maximum_value*、および *counter_options* を返します。

NC_FINISH

現在の MVS タスク (TCB) と、指定されたプールの名前付きカウンター・サーバーとの間の接続を終了します。同一のプールに対して、後続の要求が行われた場合は、新規の接続が確立されます。

この機能は、特定のカウンターには適用されません。したがって、必要なパラメーターは、機能、戻りコード、およびプール名だけです。

この機能は、接続を終了する特別な理由 (例えば、サーバーのシャットダウンを許可するなど) がある場合にのみ使用します。

NC_INQUIRE

名前付きオプションの詳細 (*current_value*、*minimum_value*、*maximum_value* および *counter_options*) を変更せずに返します。現行値は、次の NC_ASSIGN 呼び出しで返される値です。名前付きカウンターの最大値が既に割り当てられている場合は、サーバーは最大値より 1 大きい現行値を返します。

NC_REWIND

名前付きカウンターを、その最小値にリセットします。この機能は、最大値によって許可されている最後の数値が割り当てられた場合にのみ有効で、カウンターを NC_COUNTER_AT_LIMIT 条件のままにします。NC_ASSIGN 呼び出しによって、サーバーが名前付きカウンターに最終値を代入した場合は、NC_REWIND 機能を使用してカウンターをリセットします。

この操作には、compare_min および compare_max パラメーターを使用して、名前付きカウンターの現行値に条件テストを組み込むことができます。

パラメーター・リストにこれらのフィールドを指定し、要求が成功すると、サーバーは、新規の現行値、最小値、および最大値を返します。

いずれかのオプション・パラメーターまたは *update_value* パラメーターが、名前付きカウンターが限界に達していたために失敗した NC_ASSIGN 要求に指定されていた場合は、同じパラメーター値を NC_REWIND 要求にも指定しなければなりません。これにより、元の NC_ASSIGN がまだ失敗するかどうかを検査できます。NC_REWIND 要求は、対応する NC_ASSIGN 要求が成功する場合は常に、戻りコード 102 (NC_COUNTER_NOT_AT_LIMIT) で抑制されます。

NC_WRAP オプションが有効な場合、または *update_value* パラメーターがゼロの場合は、これらの条件で NC_ASSIGN が常に成功するため、NC_REWIND は抑制されます。NC_WRAP オプションについて詳細については、*counter_options* パラメーターを参照してください。

NC_UPDATE

名前付きカウンターを新規の値に設定します。この操作には、*compare_min* および *compare_max* パラメーターを使用して、名前付きカウンターの現行値に条件テストを組み込むことができます。

ユーザーは、新規の値を *update_value* パラメーターに指定します。新規の値を指定しない場合は、名前付きカウンターは未変更のままとなります。

この機能を使用して、有効な *counter_options* 指定変更パラメーター (またはヌル・アドレス) を指定できますが、カウンター・オプションは有効ではありません。ヌル・アドレスか NC_NONE のいずれかを *counter_options* パラメーターとして指定します。

return_code
戻りコードを受信するための 32 ビットの整数フィールドを指定します。レジスター 15 にも同じ情報が返されます。レジスター 15 は、COBOL の呼び出し側の場合、RETURN-CODE 特殊レジスターに保管されています。

各戻りコードには、対応する記号定数があります。これらについて詳細については、733 ページの『戻りコード』を参照してください。

pool_selector
8 文字のプール選択パラメーターを指定します。このパラメーターは、名前付きカウンターがあるプールの識別に使用します。

このパラメーターはオプションです。プール選択パラメーターを省略した場合は、8 つの空白 (X'40') 文字のストリングとみなされます。

pool_selector に使用できる文字は、A ～ Z、0 ～ 9、$ @ # および _ (下線) ですが、先頭文字を数字または下線にすることはできません。必要に応じて、8 文字を埋めるために、パラメーターには末尾スペースを埋め込まなければなりません。現行領域にデフォルトのプールを使用するために、パラメーターをすべてスペースにすることができます。このようにすると、オプション・テーブルによって有効な非ブランクの名前付きカウンター名にマップされます。

使用中の名前付きカウンター・オプション・テーブルに応じて、プール選択パラメーターを実際のプール名として使用するか、またはオプション・テーブルによって実際のプール名にマップされる論理プール名として使用することが可能です。デフォルト・オプション・テーブルは、下記を想定しています。

- DFHNC で始まるすべてのプール選択パラメーター (テーブル項目を POOLSEL=DFHNC* とマッチングする) は、実際のプール名である。
- 他のすべてのプール選択パラメーター (すべてのブランクを含む) は、デフォルト・プール名にマップする。

注: 呼び出しインターフェース用のデフォルト・プール名は、DFHNC001 です。EXEC CICS API 用のデフォルト・プール名は、システム初期設定パラメーター NCPLDFT によって定義されます。
DFHNCOPT オプション・テーブル内のプール選択パラメーターについて詳細については、719 ページの『名前付きカウンター・オプション・テーブル』を参照してください。

counter_name
名前付きカウンターの名前を収容する 16 バイトのフィールドを指定します。必要に応じて末尾スペースが埋め込まれます。

名前に使用できる文字は、A ～ Z、0 ～ 9、$ @ # および _ (下線) ですが、先頭文字を数字または下線にすることはできません。必要に応じて、16 文字を埋めるために、パラメーターには末尾スペースを埋め込まなければなりません。

他のアプリケーションとの競合を避けるため、8 バイトまでの共通接頭部を持つ名前を使用することをお勧めします。 CICS によって内部的に使用される名前付きカウンターの名前は、すべて DFH で始まります。

NC_BROWSE_FIRST および NC_BROWSE_NEXT 機能の場合、実際の名前は、このフィールドに返されます。この名前は、この目的に応じた変数でなければなりません。他のすべての機能の場合、これは定数の場合があります。

value_length
名前付きカウンターの各値フィールドの長さを示す 32 ビットの整数フィールドを指定します。値は、符号なし形式 (高位ビットは値の一部)、または正符号付き形式 (高位ビットはゼロ符合ビットのために予約済み) のいずれであってもかまいません。符号なし形式を使用するには、8 ビットから 64 ビットに対応する 1 ～ 8 までの範囲のバイト単位で長さを指定します。符号付き形式で値を使用するには、7 ビットから 63 ビットに対応する -1 から -8 までの範囲の負の数値でバイトの長さを指定します。名前付きカウンター EXEC インターフェースとの互換性のために、フルワードの符号付きバイナリー値 (COUNTER) として処理されるカウンターではこの長さを -4 に設定し、ダブルワードの符号なしバイナリー値 (DCOUNTER) として処理されるカウンターでは 8 に設定する必要があります。

呼び出しに値パラメーターが使用されていない場合は、値の長さを含むすべての末尾の未使用パラメーターを完全に省略するか、*value_length* を 0 に指定することができます。

入力値は、8 バイトより短い場合、高位 0 バイトを使用して内部で使用されている完全な 8 バイトに拡張されます。出力値が短い値フィールドに返される場合、指定した下位バイトの数値が返され、すべての高位バイトは無視されます。ただし、値が長すぎてフィールド内で正しく表すことができない場合は、 732 ページの『結果オーバーフローの検査』で説明されているように警告戻りコードを設定することができます。

current_value
以下に使用される変数を指定します。

- 名前付きカウンターの初期シーケンス番号の設定
- 名前付きカウンターからの現行シーケンス番号の受信

NC_CREATE 機能の場合、このパラメーターは入力 (送信側) フィールドとなり、定数としての定義が可能です。デフォルト値は低い値 (2 進ゼロ) です。この値には、下記の最小値および最大値で指定した範囲内の値か、または、最大値より 1 大きい値のいずれかが可能です。最大値より 1 大きい値の場合は、この値を使用する前に NC_REWIND 機能を使用してカウンターをリセットする必要があります。このフィールドでは符号検査は行われませんが、符号ビットが設定された値は、通常、カウンターの限界と矛盾するとして、サーバーによって拒否されます。すべての符号付きの番号で構成される範囲を持つカウンターでは、しきい値にあるカウンターには符号ビットが設定されており、このカウンターを有効な入力値として使用することができます。

他のすべてのカウンター機能では、このパラメーターは出力 (受信側) フィールドとなり、変数で定義されなければなりません。

minimum_value
以下に使用される変数を指定します。

- 名前付きカウンターの最小値の設定
- 名前付きカウンターからの指定された最小値の受信

NC_CREATE 機能の場合、このパラメーターは入力 (送信側) フィールドとなり、定数としての定義が可能です。デフォルト値は低い値 (2 進ゼロ) です。

他のすべての機能では、このパラメーターは出力 (受信側) フィールドとなり、変数で定義されなければなりません。

maximum_value

以下に使用される変数を指定します。

- 名前付きカウンターの最大値の設定
- 名前付きカウンターからの指定された最大値の受信

NC_CREATE 機能の場合、このパラメーターは入力 (送信側) フィールドとなり、定数としての定義が可能です。非ゼロの *value_length* パラメーターを指定し、その一方で *maximum_value* を省略する場合は、指定された長さの高い値 (または、符号付き変数では、正の最大値) が *maximum_value* のデフォルトになります。*value_length* パラメーターを省略したか、ゼロと指定した場合は、高い値の 8 バイトが *maximum_value* のデフォルトになります。しかし、最小値がすべて低い値で、最大値が高い値の 8 バイトの場合は、この最大値は削減され、サーバーが内部使用に予約値を使用できるようにします。

他のすべての機能では、このパラメーターは出力 (受信側) フィールドとなり、変数で定義されなければなりません。

counter_options

オプションのフルワード・フィールドを指定して、折り返しの制御と、削減の増分を行う名前付きカウンター・オプションを示します。有効なオプションは、シンボリック値 NC_WRAP または NC_NOWRAP と NC_REDUCE または NC_NOREDUCE によって表されます。デフォルト・オプションは NC_NOWRAP と NC_NOREDUCE です。

NC_NOWRAP

サーバーは、NC_COUNTER_AT_LIMIT 条件で失敗する NC_ASSIGN 要求に応答して、名前付きカウンターを自動的に最小値に巻き戻すことはありません。NC_NOWRAP が有効になっていて、かつ名前付きカウンターが NC_COUNTER_AT_LIMIT 状態の場合、カウンターが NC_REWIND 要求 (または NC_WRAP にリセットするカウンター・オプション) によってリセットされるまで、NC_ASSIGN 機能は操作不能です。

NC_WRAP

サーバーは、NC_COUNTER_AT_LIMIT 条件にあるカウンターの NC_ASSIGN 要求に応答して、NC_REWIND を自動的に実行します。サーバーは、名前付きカウンターの現行値を最小値と等しい値に設定して、呼び出し側にこの新しい現行値を返し、次に名前付きカウンターを増分します。

NC_NOREDUCE

割り当てられる残りの数値の範囲 (現行値と、最大値に 1 を加えた値の差) が、*update_value* パラメーターで指定された増分より小さい場合、その割り当ては失敗します (NC_WRAP が有効な場合を除く)。NC_NOREDUCE は、NC_NOWRAP と共に使用した場合、NC_ASSIGN 要求が NC_COUNTER_AT_LIMIT 条件で失敗することを意味します。

例えば、現行値が 199 990 であり、カウンター最大値が 199 990 に定義されている場合に、要求が更新値を 15 に指定すると、増分により現行値が 200 000 を超えるため、NC_REQUEST は失敗します。

NC_REDUCE
割り当てられる残りの数値の範囲 (現行値と、最大値に 1 を加えた値の差) が、*update_value* パラメーターで指定された増分より小さい場合、その増分は削減されて、割り当ては成功します。この場合、NC_ASSIGN 要求には、*update_value* によって指定された数値より小さい数値の範囲が割り当てられており、名前付きカウンターは、NC_COUNTER_AT_LIMIT 条件のままになっています。後続の NC_ASSIGN 要求は、名前付きカウンターが NC_REWIND 要求でリセットされるまで失敗します。

NC_CREATE で指定されたオプションは、名前付きカウンターと共に保管され、他の名前付きカウンター機能のデフォルトとして使用されます。NC_ASSIGN、NC_REWIND、または NC_UPDATE 要求のオプションは、指定変更できます。DFHNCTR 呼び出しで *counter_options* を指定しない場合は、記号定数 NC_NONE (ゼロに等しい) を入力パラメーターとして指定します (または、ヌル・アドレスを指定します)。

NC_CREATE、NC_ASSIGN、NC_REWIND、および NC_UPDATE 機能では、このパラメーターは入力フィールドとなります。

NC_DELETE、NC_INQUIRE、および NC_BROWSE 機能では、このパラメーターは出力フィールドとなり、NC_CREATE で指定されたオプションを返します。

update_value
カウンターの更新に使用される値を指定します。NC_ASSIGN の場合、これは、(現行の数値が割り当てられた後) 現行カウンター値に追加される増分です。1 以外の増分を指定した場合に割り当て操作にどのように影響するかについての情報は、*function* パラメーターの NC_ASSIGN オプションを参照してください。

NC_UPDATE では、これは、名前付きカウンターの新しい現行値です。

compare_min
名前付きカウンターの現行値と比較される値を指定します。値を指定すると、このパラメーターは、NC_ASSIGN、NC_REWIND、または NC_UPDATE 操作を、指定された値より大きいか、または指定された値と等しい名前つきカウンターの現行値の条件にします。比較が正しく実行されない場合は、この操作は、範囲外カウンターの戻りコード (RC 103) で拒否されます。

ヌル・アドレスを指定してこのパラメーターを省略した場合は、サーバーが比較を実行しません。

compare_max
名前付きカウンターの現行値と比較される値を指定します。値を指定すると、このパラメーターは、NC_ASSIGN、NC_REWIND、または NC_UPDATE 操作を、指定された値より小さいか、または指定された値と等しい名前つきカウンターの現行値の条件にします。比較が正しく実行されない場合は、この操作は、範囲外カウンターの戻りコード (RC 103) で拒否されます。

このパラメーターに高位の値 (X'FF') を指定すると、サーバーは比較を実行しません。*value_length* パラメーターによって指定されたすべてのバイトに (X'FF') を指定する必要があります。

compare_max 値が compare_min 値より小さい場合、値の範囲は折り返すことになります。この場合、どちらか一方の比較が正常に実行されると、現行値は、範囲内にあると見なされ、そうでなければ、両方の比較が正常に行われなければなりません。

結果オーバーフローの検査

呼び出しインターフェースは、結果フィールドの指定サイズに適合しない、または符号付き変数を使用すると符号ビットにオーバーフローする結果を検査します。

value_length を負の値として指定して、結果フィールド (*counter_value*、*minimum_value*、または *maximum_value*) を符号付き変数として定義している場合は、呼び出しインターフェースが、符号ビットにオーバーフローする結果を検査します。この場合、操作は正常に完了しますが、戻りコード NC_RESULT_OVERFLOW が設定されます。特殊な場合には、戻りコードの不要な設定を避けるために、しきい値にあるカウンターの結果値は、この形式のオーバーフローでは検査されません。つまり、限界にあり最大値が正の最大値であるカウンターに対して照会が行われる場合は、この戻りコードを使用せずに現行のカウンター値として負の数値が返される可能性があります。

結果フィールド (*counter_value*、*minimum_value*、または *maximum_value*) が短すぎて、結果の非ゼロの部分全体を収めることができない場合、操作は正常に完了しますが、以下の戻りコードのいずれかが設定されます。

- NC_RESULT_CARRY、先行部分が 1 である場合。
- NC_RESULT_TRUNCATED、先行部分が 1 より大きい場合。

最大値を持つ 4 バイトの符号なしカウンターがしきい値に到達した場合は、戻りコード NC_RESULT_CARRY が設定され、カウンター値は 0 になります。

ヌル・パラメーターを持つ DFHNCTR 呼び出しの例

DFHNCTR 呼び出しのオプション・パラメーターを省略する場合は、そのパラメーター・リストが欠落しているパラメーターのヌル・アドレスで作成されていることを確認してください。下記の例は、ヌル・アドレスに設定されているいずれかのパラメーターを持つ NC_CREATE 要求を COBOL プログラムから発行する方法について例証しています。

省略されたパラメーターのヌル・アドレスを持つ DFHNCTR 呼び出し: この例では、呼び出しに使用されているパラメーターは、以下のように、WORKING-STORAGE SECTION で定義されています。

呼び出しパラメーター	COBOL 変数	フィールド定義
function	01 FUNCTION	PIC S9(8) COMP VALUE +1.
return_code	01 NC-RETURN-CODE.	PIC S9(8) COMP VALUE +0.
pool_selector	01 NC-POOL-SELECTOR	PIC X(8).
counter_name	01 NC-COUNTER-NAME	PIC X(16).

呼び出しパラメーター	COBOL 変数	フィールド定義
value_length	01 NC_VALUE-LENGTH	PIC S9(8) COMP VALUE +4.
current_value	01 NC-CURRENT-VALUE	PIC S9(8) VALUE +0.
minimum_value	01 NC-MIN-VALUE	PIC S9(8) VALUE +0.
maximum_value	01 NC-MAX-VALUE	PIC S9(8) VALUE -1.
counter_options	01 NC-OPTIONS	PIC S9(8) COMP VALUE +0.
update_value	01 NC-UPDATE-VALUE	PIC S9(8) VALUE +1.
compare_min	01 NC-COMP-MIN	PIC S9(8) VALUE +0.
compare_max	01 NC-COMP-MAX	PIC S9(8) VALUE +0.

ヌル・アドレスに使用されている変数は、以下のように、LINKAGE SECTION で定義されています。

```
LINKAGE SECTION.
  01  NULL-PTR           USAGE IS POINTER.
```

以下は、上記にあるように WORKING-STORAGE SECTION で指定されているデータ名と、LINKAGE SECTION に記載されているように NULL-PTR 名を使用して、*value_length*、*current_value*、*minimum_value*、および *counter_options* のみが指定されているオプション・パラメーターである名前付きカウンター・サーバーへの呼び出しを例証しています。その他は、デフォルトが許可されているか、または、末尾のオプション・パラメーターの場合は、まとめて省略されます。

```
 NAMED-COUNTER SECTION.
*
     SET ADDRESS OF NULL-PTR TO NULLS.
*
     MOVE   1     TO  FUNCTION.
     MOVE 100     TO  NC-MIN-VALUE  NC-CURRENT-VALUE.
     MOVE NC-WRAP TO  NC-OPTIONS.
     MOVE "DFHNC001" TO NC-POOL-SELECTOR.
     MOVE "CUSTOMER_NUMBER" TO NC-COUNTER-NAME.
     CALL 'DFHNCTR' USING  FUNCTION NC-RETURN-CODE NC-POOL-SELECTOR
                   NC-COUNTER-NAME  NC-VALUE-LENGTH  NC-CURRENT-VALUE
                   NC-MIN-VALUE  NULL-PTR  NC-OPTIONS.
```

戻りコード

名前付きカウンターの呼び出しインターフェースには 3 つの警告戻りコード (1 から 3) があり、これらは要求の結果がオーバーフローしたことを示します。そうでなければ、正常に完了していました。複数の警告戻りコードが同じ要求に適用される場合は、最も適切な警告戻りコードが設定されます。

残りの戻りコードは、その重大度に応じた範囲 (100、200、300、および 400®) に分割されています。非ゼロの戻りコードの各範囲は、戻りコードのカテゴリーを示すダミーの戻りコードで始まります。これによって、シンボル名を使用している各範囲内の値の検査が容易になります。

以下のリストでは、数値の戻りコードの後にそのシンボル名が続きます。

0 (NC_OK)
要求は正常に完了しました。

1 (NC_RESULT_OVERFLOW)
結果値が符号ビットにオーバーフローしました。

2 (NC_RESULT_CARRY)
結果値がオーバーフローし、先行部分は 1 でした。

3 (NC_RESULT_TRUNCATED)
結果値がオーバーフローし、先行部分は 1 より大きい値でした。

100 (NC_COND)
この範囲の戻りコードは、条件が満たされなかったために条件機能が成功しなかったことを示します。

101 (NC_COUNTER_AT_LIMIT)
NC_ASSIGN 機能は、この名前付きカウンターの直前の要求が最大値を取得し、カウンターが現在限界にあるため、拒否されます。NC_REWIND 機能呼び出しが実行されてカウンターをリセットするまで、新規のカウンター値を割り当てることはできません。

102 (NC_COUNTER_NOT_AT_LIMIT)
名前付きカウンターがそのしきい値ではないため、NC_REWIND FUNCTION は拒否されます。これは、別のタスクが NC_REWIND を持つカウンターのリセットに既に成功した場合に最も発生しやすい戻りコードです。

103 (NC_COUNTER_OUT_OF_RANGE)
名前付きカウンターの現行値が、*compare_min* および *compare_max* パラメーターで指定された範囲内にありません。

200 (NC_EXCEPTION)
この範囲の戻りコードは、アプリケーション・プログラムが処理できなければならない例外条件を示します。

201 (NC_COUNTER_NOT_FOUND)
名前付きカウンターが見つかりません。

202 (NC_DUPLICATE_COUNTER_NAME)
NC_CREATE 機能は、指定された名前の名前付きカウンターが既に存在するため、拒否されます。

203 (NC_SERVER_NOT_CONNECTED)
NC_FINISH 機能は、選択されたプールのアクティブな接続が存在しないため、拒否されます。

300 (NC_ENVIRONMENT_ERROR)
この範囲の戻りコードは、環境エラーを示します。これらは、重大なエラーで、通常、プログラムが処理できない可能性のある外部要因によって発生します。

301 (NC_UNKNOWN_ERROR)
サーバーが、インターフェースが理解しないエラー・コードを報告しました。一般的に、インターフェース・ロード・モジュール DFHNCIF の保守またはリリース・レベルが、サーバーのものより低くなければ、このようなことは発生しません。

302 (NC_NO_SPACE_IN_POOL)
名前付きカウンター・プール内のスペースが不足しているため、新規カウンターを作成できません。

303 (NC_CF_ACCESS_ERROR)
構成上の障害または接続の切断などの予期しないエラーが、カップリング・ファシリティーへのアクセスに使用されるマクロで発生しました。詳細については、CICS ジョブ・ログのメッセージ DFHNC0441 を参照してください。

304 (NC_NO_SERVER_SELECTED)
プログラム内に指定されているプール選択パラメーターは、現在のオプション・テーブルを使用して有効なサーバー名に解決されません。

305 (NC_SERVER_NOT_AVAILABLE)
インターフェースで、該当する名前付きカウンター・プールのサーバーへの接続を確立できません。詳細については、CICS ジョブ・ログ内の AXM サービス・メッセージに記載されています。

306 (NC_SERVER_REQUEST_FAILED)
要求のサーバー処理中に異常終了が発生しました。詳細については、CICS ジョブ・ログおよびサーバー・ジョブ・ログのメッセージを参照してください。

307 (NC_NAME_TOKEN_ERROR)
名前付きカウンター・インターフェース・モジュール内の IEANT*xx* 名前/トークン・サービス呼び出しで、予期しない戻りコードが提供されました。

308 (NC_OPTION_TABLE_NOT_FOUND)
プール名の解決に必要な DFHNCOPT オプション・テーブル・モジュールをロードできませんでした。

309 (NC_OPTION_TABLE_INVALID)
オプション・テーブルの処理中に、名前付きカウンター・インターフェースで不明な項目形式が検出されました。オプション・テーブルが正しく生成されていないか、または DFHNCIF インターフェース・ロード・モジュールがオプション・テーブルと同じリリース・レベルではありません。

310 (NC_USER_EXIT_NOT_FOUND)
指定されたプール名と一致するオプション・テーブル項目でユーザー出口プログラムが指定されましたが、そのユーザー出口プログラムはオプション・テーブルを使用してリンク・エディットされていないため、ロードできません。

311 (NC_STRUCTURE_UNAVAILABLE)
名前付きカウンター・サーバーのリスト構造は、一時的に使用できません。その理由の一例として、z/OS システムによって管理されている再作成が進行中であることなどが挙げられます。

注: 名前付きカウンターへの EXEC CICS インターフェースは、内部で CALL インターフェースを使用しますが、1 秒間待って要求を再試行することで、この戻りコードをアプリケーション・プログラムから隠します。EXEC CICS インターフェースは、成功するまでこの待機と再試行を続けます。その結果、アプリケーション・プログラムは、時間的遅延のみを体験

し、エラー応答は得ません。CALL インターフェースを使用するアプリケーション・プログラムでも同じ技法を使用できます。

400 (NC_PARAMETER_ERROR)
この範囲の戻りコードは、パラメーター・エラーを示しており、一般に、呼び出し側のプログラムのコーディング・エラーが原因です。

401 (NC_INVALID_PARAMETER_LIST)
以下のいずれか 1 つの理由により、パラメーター・リストが無効です。

- 指定されたパラメーターが少なすぎる (4 つより少ない、または NC_FINISH 機能の場合 3 つより少ない)
- 指定されたパラメーターが多すぎる (8 つより多い)
- パラメーター・アドレスがゼロになっている
- リストの終わりのマーカーが欠落している

402 (NC_INVALID_FUNCTION)
機能コード・パラメーターが、サポートされている範囲内にありません。

403 (NC_INVALID_POOL_NAME)
プール選択パラメーターに、許可されていない文字、または埋め込みスペースが含まれています。

404 (NC_INVALID_COUNTER_NAME)
counter_name パラメーターに、許可されていない文字、または組み込みスペースが含まれています。

405 (NC_INVALID_VALUE_LENGTH)
値の長さパラメーターが 0 ～ 8 の範囲内にありません。

406 (NC_INVALID_COUNTER_VALUE)
指定されたカウンター値または増分値が、カウンターの最小限度または最大限度と矛盾しています。

NC_CREATE 機能の *current_value* パラメーター、または、NC_UPDATE 機能の *update_value* パラメーターで指定されたカウンター値は、指定された最小値より小さくても、(最大値 + 1) より大きくてもなりません。

NC_ASSIGN または NC_REWIND 機能の *update_value* パラメーターで指定されている増分値は、カウンターの合計範囲 ((最大値 - 最小値) + 1) より大きくてはなりません。

407 (NC_INVALID_COUNTER_LIMIT)
最大値が、最小値より小さい値になっています。

408 (NC_INVALID_OPTIONS)
counter_options パラメーターの値が無効です。値が、定義されているどのオプションにも対応しないか、または、相互に排他的なオプションを表しています。

第 56 章 名前付きカウンターのリカバリー

名前付きカウンターは、カップリング・ファシリティーにのみ保管されます。よって、名前付きカウンターを使用しているアプリケーションは、場合によってはカップリング・ファシリティーの障害によって被る可能性のある影響に対処するためにリカバリー・ロジックをインプリメントする必要があります。

カップリング・ファシリティーまたは名前付きカウンター・プールのリスト構造に障害が起こり、別のファシリティーが使用可能な場合、通常はその使用可能なファシリティーで名前付きカウンター・プールのリスト構造を非常に迅速に再作成することが可能です。サーバーの元のインスタンスは、問題を検出するとすぐに終了し、通常は、ARM が使用可能でインストール・ポリシーが再始動を許可すれば、ARM によって新規インスタンスが即時に開始されます。プールのリスト構造に障害が起こったことが明らかな場合、新規サーバーは、新規インスタンスを即時に割り振ることができなくてはなりません。また、プールは数秒以内に再度使用可能にならなくてはなりません。

ただし、状況によっては (カップリング・ファシリティーに電源障害が起こった場合など)、MVS は最初にその状況を接続の切断と認識し、元のファシリティーが再始動するまでリスト構造に障害が起こったことを判別することができません。このような状況の場合は、既存の構造を強制削除するオペレーター・コマンドを発行して、新規インスタンスの即時に割り振りを許可することでリカバリーの速度を上げることができます。

新規構造が作成されるまでサーバーは使用不能なので、カウンター値取得の試みはリジェクトされます。つまり、このような要求を出すアプリケーションが数値割り当ての代替方法をもたない限り、新規構造の作成中、これらのアプリケーションは使用不能です。

障害が原因で名前付きカウンター・プールのリスト構造が再作成された場合、そのリスト構造は空であり、アプリケーションは名前付きカウンターがもう存在しないことを即時に発見します。この状況における標準のリカバリー技法は、次のとおりです。

1. アプリケーションに、リソースに対してカウンター名に基づきエンキュー・コマンド (ENQ) を発行させます。これにより、各名前付きカウンターの修復のために必ず 1 つのタスクのみが試行されます。
2. 別のタスクが既に名前付きカウンターを再作成したかどうかをチェックします。
3. 名前付きカウンターが再作成されていない場合は、以下で説明する方法を用いて、他の情報から再構成した適切な値で名前付きカウンターを再作成します。
4. 名前付きカウンターを解放するために、リソースに対してデキュー・コマンド (DEQ) を発行します。

複数のタスクによる時間の無駄およびリソースによる同一処理の重複を避けるために、エンキューおよびデキュー・プロセスが使用されます。ただし、名前付きカウンターの再作成に用いるプロセスが単純であれば、エンキューおよびデキュー・プロセスを省略することができます。別のタスクが既に名前付きカウンターを修復済

© Copyright IBM Corp. 1989, 2009

みの場合、後続のカウンター再作成の全試行は、カウンター名が重複しているというメッセージと共にリジェクトされます。

名前付きカウンターが、構造障害発生中に保存する必要のない一時情報 (例えば、構造障害時には保持されない可能性のある、アクティブ・プロセスの固有 ID など) にのみ使用されている場合、リカバリー技法は最小限で済みます (例えば、カウンターを標準初期値で再作成するなど)。

ただし、永続情報 (オーダー番号など) に名前付きカウンターが使用されている場合、カウンターの再作成には、カウンター値を再構成するための特有のアプリケーション・ロジックが必要となる場合があります。例えば、アプリケーションに、オーダー・ファイル内で現在使用されている最高位キーを検出させることができます。アクティブ・トランザクションが既にカウンターから新規数値を獲得済みであり、まだそれらの数値が使用されていない場合は、リカバリー・プロセスでこのことを考慮する必要があります。割り当て済みで、まだ記録されていない値を考慮に入れる方法は、2 つあります。

1. 最後に使用された値に安全マージンを追加します。この際、アプリケーションがカウンターを、既に割り当てられている可能性のある値に設定しないように、十分に大きいマージンを選んでください。
2. すべてのカウンター値を一時的な値として扱います。カウンターを次の明らかに未使用の値に復元し、そのカウンターを使用するアプリケーションに、カウンター値が既に割り当て済みの状況に適用されるするロジックを組み込みます。するとアプリケーションはそのロジックの使用を試みます。重複値は、その値が (データベースまたはファイル・キーとして) 使用された時に重複キー例外によって検出されます。その時点で、アプリケーションは新規のカウンター値を取得して、再試行できます。重複値を使用しようとした最初の試みの結果、副次作用が生じないように気を付けてください。

使用された最高位のカウンター値の検出および次の値の一時的割り当ての技法は、名前付きカウンター・サーバーが使用不能な場合の数値割り当てのバックアップ方法としても利用できます。ただし、その場合、重複キーを処理するロジックが通常はとても異常なリカバリー状態でのみ実行されるため、検証とテストに特に注意を払う必要があります。

既存のデータ・リポジトリーからカウンター値を再作成することが困難な場合、もう 1 つの可能な手段として、極めて短い間隔で (例えば、100 または 1000 ごとに) カウンター値がファイル内のレコードにログ記録されるという方法があります。リカバリー・ロジックは、ファイルにログ記録された数値を取り出し、安全マージンを追加する (値がログ記録される間隔を 2 倍にするなど) ことによって、名前付きカウンターに適した値を再作成することができます。

z/OS リリース 3 以上が実行されているシステムの場合、システム管理の二重化を使用して、異なるカップリング・ファシリティー内の名前付きカウンターの二重コピーを保持することができます。これにより、カウンターをアクセスできなくなるリスクは大幅に減りますが、パフォーマンスとリソースには多少の悪い影響を与えます。操作上のエラーまたはソフトウェアの問題が原因で、この構造が失われる理論上のリスクも依然として存在し、またカップリング・ファシリティー内のどのデータも永続的と見なすことはできないので、カウンター値の再構成方法も依然として必要とされる場合があります。

第 14 部 印刷とスプール・ファイル

印刷制御機構およびスプール・ファイルのための CICS 機能の理解および使用。

© Copyright IBM Corp. 1989, 2009

第 57 章 印刷の CICS サポート

CICS は印刷用として特別なコマンドを提供しませんが、BMS コマンドおよび端末制御コマンドにはプリンターだけに適用されるオプションがあり、さらに一時データまたは SPOOL コマンドを使用する一部のプリンター用のオプションがあります。次のセクションでは、API および選択項目を判別する要因について説明します。

他のタイプのエンド・ユーザー・コミュニケーションでは通常は起こらない、印刷と関連した次の 2 つの論点があります。

1. 特に 3270 プリンターの場合に、形式設定に関する追加の考慮事項がある。
2. 印刷の必要があるタスクは、プリンターに直接アクセスする必要がない。

さらに、プリンターには、異なるアプリケーション・プログラミング・インターフェースを持つ、次の 2 つの異なるカテゴリーがあります。

CICS プリンター
端末として CICS に定義され、CICS が直接管理するプリンター。通常、これはエンド・ユーザーの近くに設置される低速装置で、比較的短い文書の即時印刷に適しています。3289 および 3262 が通常 CICS プリンターとして接続されます。

非 CICS プリンター
オペレーティング・システムまたは別のアプリケーションが管理するプリンター。これらのプリンターは、中央処理設置場所に設置される通常高速な装置で、即時に使用可能になる必要がない大量印刷に適しています。また、これは拡張機能または特殊な接続、管理、または共用を必要とする他のプリンターである場合もあります。

CICS プリンターの書式設定

プリンター端末への書き込みに関するアプリケーション・プログラミング・インターフェースは、基本的に、ディスプレイへの書き込みの場合と同じです。(このセクションでは、タスクがそのプリンシパル装置としてプリンターを持っている場合の問題については説明しません。この点の説明は 752 ページの『CICS プリンターの使用』にあります。)

端末制御コマンド (SEND) はすべての CICS プリンターに対して使用可能であり、その大部分は BMS (SEND MAP、SEND TEXT、および SEND CONTROL) でもサポートされています。 775 ページの『BMS サポート・レベル』 には、BMS でサポートされているデバイスがリストされています。外部コントローラーまたは LU タイプ 4 のコンポーネントであるプリンターの場合には、端末管理および BMS の他にバッチ・データ交換 (BDI) コマンドも使用することができます。BDI コマンドの説明は、 617 ページの『第 42 章 バッチ・データ交換の使用』にあります。

BMS と端末管理を使用する間の選択項目はディスプレイ端末の場合と同じ考慮事項を基礎にしています。ディスプレイと同様に、プリンターは機能面およびその機能

© Copyright IBM Corp. 1989, 2009

のインプリメンテーション面の両方で大いに違いがあり、その違いはデータ・ストリームと受け入れる装置制御の面に反映されます。

端末制御コマンドを使用する場合には、アプリケーション・コードは、プリンターが必要としている方法で出力を形式設定しなければなりません。ライン・プリンターおよび類似装置の場合には、形式設定はプログラミングにわずかな影響しか与えません。しかし、高機能プリンターの場合には、データ・ストリームがしばしば非常に複雑になります。形式設定には有効なアプリケーション・コードが必要であり、装置に依存する要素がプログラム論理に入り込みます。

これらの端末の一部にとっては、BMS を使用することによって、コーディングの労力が著しく削減されます。この BMS は、プログラマーが装置データ・ストリームを作成すること、あるいはそれを理解することすら軽減します。また、変更なしで、同一のプログラムが、多くのタイプのプリンター、あるいはプリンターとディスプレイの混合をサポートできるように、BMS はほとんどのデータ・ストリーム依存関係をアプリケーション・コードから除去します。BMS はすべての装置依存関係を除去するわけではなく、形式設定に関するいくつかの制約事項が課せられます。また、余分なパス長もともないます。その量は、個別の BMS 要求の数、要求の複雑さ、およびユーザー独自のプログラムで避けられるパス長によって異なります。

印刷出力の要求

CICS 印刷要求は、CICS に要求画面の内容を、同一制御装置で最初に利用可能なプリンターにコピーすることを要求します。ともなうオーバーヘッドはプリンターが利用可能かどうか、および要求端末が CICS に対してリモートかローカルかによって異なります。

プリンターが使用不能で、要求がリモート装置あるいはローカル装置からである場合、次のように処理されます。

- CICS はディスプレイ端末のバッファーを読み取ります。これには、ヌルを含む画面上の**すべての** 位置の伝送がともないます。

  ローカル装置からの要求の場合、READ BUFFER コマンドはチャネル速度で行われるので、入力メッセージが大きくても、応答に時間がかかり過ぎたり、回線を占有することはありません。
- 端末エラー・プログラムがメッセージを処置することができるように、エラー・タスクが生成されます。プリンターが使用可能で、要求がローカル装置からのものである場合には、このステップは不要です。
- プリンターが使用可能になった時点で、3270 印刷タスク (CSPP) が生成されて、バッファー全体がそのプリンターに書き出されます。

プリンターが使用可能で、要求がリモート装置から出ているのであれば、CICS は、要求を出した装置のバッファーのコピーを出力装置バッファーに送ることを要求する、非常に短いデータ・ストリームを制御装置に送ります。

CICS 3270 プリンター

BMS が提供するプリンターに関する追加の形式制御機構のほとんどは、特定タイプの CICS プリンター、すなわち、3270 プリンターを対象とするものです。**3270 プリンター**とは、3270 データ・ストリームを受け入れる任意のプリンターであり、3270 ディスプレイに相当するハードコピー装置です。このプリンターは、3270 ディスプレイのディスプレイ装置バッファーにあたるページ・バッファーを持っています。 (3270 データ・ストリームの紹介については、 623 ページの『3270 バッファー』を参照してください。) 最初に 3270 プリンターについて説明し、もっと単純な非 3270 プリンターについては、 748 ページの『非 3270 CICS プリンター』まで説明を据え置きます。

3270 プリンターは、2 つの異なるタイプの形式設定指示、すなわち、**バッファー制御オーダー**および**印刷形式設定オーダー**を受け入れます。バッファー制御オーダーは、制御装置が受信したときに実行され、バッファーを埋め込む方法を制御します。このオーダーは、3270 表示画面を形式設定するために使用するオーダーと同じです。重要なオーダーの一部については既に 629 ページの『データ・ストリーム中のオーダー』で説明しました。例えば、SBA (バッファー・アドレス設定) は、後続のデータを入れるバッファー内の場所を制御装置に指示し、SF (フィールド開始) は属性バイトおよびフィールド・データのシグナルを送る、などです。完全なリストは、「*IBM 3270 Information Display System Data Stream Programmer's Reference*」に記載されています。

対照的に、印刷形式設定オーダーは、受信時には実行されませんが、その代わりにデータとともにバッファー内に保管されます。これらのオーダー (NL (改行)、FF (用紙送り) など) は印刷操作の間のみ解釈され、この時点で印刷出力の形式が制御されます。(これはディスプレイでは、バッファー位置を占めるだけで、何の効果もありません。これは画面上ではブランクのように見えます。)

3270 プリンターに書き込み中の場合には、バッファー制御オーダーまたは印刷形式設定オーダー、あるいはその両方の混合を使用して形式を設定することができます。 633 ページの『アウトバウンド・データ・ストリームの例』には、バッファー制御オーダーを使用した形式設定の例が示されています。この同一のデータ・ストリームを 3270 プリンターに送信すると、プリンターは 633 ページの図 126 で示される画面のイメージを印刷します。同一データ・ストリームをディスプレイとプリンターに送信することができるように、バッファー制御オーダーを使用して印刷出力を形式設定することを選択することができます。

他方、同一ストリームを 3270 プリンターと非 3270 プリンターに送信することができるように、印刷形式設定オーダーを使用して形式設定することを選択することもできます (印刷形式設定オーダーは多くの非 3270 プリンターに対する形式設定制御とおなじです)。この選択項目について詳しくは、 746 ページの『NLEOM オプション』を参照してください。

次の表は、バッファー制御オーダーを使用した 633 ページの『アウトバウンド・データ・ストリームの例』 のデータ・ストリームと同じ*印刷* 出力を作成する、印刷形式設定オーダーを使用したデータ・ストリームを示しています。

表 39. 印刷制御オーダーを使用するデータ・ストリームの例

バイト	内容	ノーツ
1	X'FF'	プリンターを新規ページに進める「用紙送り」(FF) オーダー。
2-23	ブランク	1 行目の 1 ～ 22 桁目を占める 22 個のブランク。
24-33	Car Record	印刷するテキスト。これは、1 行目の次に使用可能な桁 (23 ～ 32) に表示されます。
34	X'1515'	プリンターを 3 行目の先頭に位置決めする、2 つの連続した「改行」(NL) オーダー。
35-80	Employee No: _____ Tag _____ State: __	3 行目の 1 桁目から印刷が始まるテキスト。
81	X'19'	印刷を停止する「メッセージ終結」(EM) 印刷オーダー。

印刷形式設定オーダーを使用した場合には、フィールド構造が失われることに注意してください。プリンターを入力用には使用しないので、このことは普通は問題ではありません。しかし、たとえ印刷形式設定オーダーを使用して形式設定したとしても、さらに、バッファー制御オーダーを使用して、テキスト領域にカラーまたは下線などの属性を割り当てることが必要になることがあります。

CICS 3270 プリンターのオプション

BMS の場合には、3270 プリンターに適用される個別制御機構は以下のコマンド・オプションの形式をとります。

- PRINT
- ERASE
- L40、L64、L80、および HONEOM
- NLEOM
- FORMFEED
- PRINTERCOMP

端末制御コマンドでは、ERASE もオプションとして表されますが、他の制御はデータ・ストリームで直接表されます。「*IBM 3270 Data Stream Device Guide*」および「*3270 Data Stream Programmer's Reference*」にはエンコード方法が示され、続いてその実行内容について説明されています。

PRINT オプションおよび印刷制御ビット

3270 ディスプレイまたはプリンターへの書き込みによって、装置バッファーが更新されます。表示の際に、結果はバッファーから駆動される画面上に即時に反映されます。ただし、プリンターの場合は、「書き込み制御文字」の該当ビットをオンにするまで印刷が開始されないため、目に見える効果がないことがあります。 (WCC は 3270 データ・ストリームの一部です。 625 ページの『書き込み制御文字』を参照してください。) BMS の場合には、SEND MAP、SEND TEXT、または SEND CONTROL コマンドの PRINT オプションを指定するか、または SEND MAP で使

用されたマップの PRINT オプションを指定して、印刷ビットをオンにします。端末管理 SEND を使用している場合には、CTLCHAR オプションを使用して印刷ビットをオンにしなければなりません。

すべての端末管理 SEND で端末書き込みが発生し、すべての SEND MAP、SEND TEXT、または SEND CONTROL で、ACCUM オプションまたは PAGING オプションを使用していない限り、端末書き込みが発生します。ACCUM は、ページがいっぱいになるか、あるいは論理メッセージが終了するまで書き込みを遅らせます。ACCUM を使用する場合には、同一ページに関するすべての SEND コマンドに同一オプションを使用する必要があります。PAGING は別のタスクへの端末書き込みを据え置きますが、それは PAGING を使用しない場合と同じ方法で生成されます。

印刷ビットがオンになるまで印刷は起こらないという事実によって、複数の書き込み段階で印刷バッファーを構築し、既にバッファー内にあるデータまたは属性バイトを変更することができます。すなわち、ハードウェアを使用して、BMS の ACCUM オプションによって得られる効果の一部を達成することができます。しかし、NLEOM オプションはこの機能に影響を与えます。次の説明を参照してください。

ERASE オプション

3270 ディスプレイ・バッファーと同様、3270 プリンター・バッファーがクリアされるのは消去する書き込みコマンドを使用した場合だけです。これは、BMS および端末管理 SEND の両方の場合に ERASE オプションを指定して行います。プリンターが代替画面サイズ機能を持っている場合には、バッファー・サイズは、ディスプレイの場合と同様に、消去の時点で設定されます。したがって、トランザクションでの最初の書き込みには消去を組み込んで、バッファーをトランザクションに必要なサイズに設定し、直前のトランザクションから残っているすべてのバッファー内容を消去する必要があります。

線幅オプション: L40、L64、L80、および HONEOM

印刷ビットに加えて、書き込み制御文字には、印刷時の行の長さを制御するビットの対が含まれています。端末制御コマンドを使用している場合には、CTLCHAR オプションを使用して、これらのビットを設定します。BMS の場合、「メッセージ終結処理」を表す HONEOM オプションによってデフォルトが作成されます。この設定値によって、プリンターはバッファー制御オーダーおよび印刷形式設定オーダーのみに従って形式設定し、印刷を、バッファー内の最初の EM (メッセージ終結) で停止します。装置の最大幅 (プラテン幅) を超えて印刷しようとした場合だけ、プリンターはそれ自身の新しい行に移動します。

しかし、行の長さが 40、64、または 80 文字 (それぞれ、L40、L64、および L80 オプション) の固定長であることを指定することもできます。これを指定した場合には、プリンターは一定の印刷形式設定オーダーを無視し、指定の行サイズに達した時に新しい行に移動し、バッファー全体を印刷します。無視される印刷形式設定オーダーは NL (改行)、CR (復帰)、および EM (メッセージ終結) です。その代わりに、図形のように印刷されます。

BMS のもとで L40、L64、または L80 を使用する場合には、端末定義のページ幅と対応する値のみを使用する必要があります (751 ページの『CICS プリンターの特

性の判別』を参照してください)。その理由は、BMS はページ・サイズを基礎にしてバッファー・アドレスを計算し、異なったページ幅を使用した場合には、これらのアドレスが誤りになるためです。

NLEOM オプション

通常 BMS は、SEND TEXT または SEND MAP のいずれを使用しているにしても、印刷形式設定オーダーではなくバッファー制御オーダーを使用して、3270 プリンターを形式設定します。しかし、NLEOM オプションを指定して、印刷形式設定オーダーのみを使用することを BMS に指示することができます。この指示をした場合には、BMS はブランクおよび NL (改行) 文字を使用してデータ全体を形式設定し、データの後ろに EM (メッセージ終結) を挿入します。NLEOM は HONEOM を暗黙指定します。 (NLEOM サポートには**標準** BMS が必要で、最小 BMS では利用不能です。)

SCS プリンターとの互換性を保つためにこれを行いたい場合があります (印刷形式設定オーダーは対応する SCS 制御文字と互換性があります)。以下で説明する操作上の相違によって、NLEOM を選択するか、もしくは選択しないことになる場合もあります。

ブランク行

3270 プリンターは印刷時にヌル行を抑制します。すなわち、データ・フィールドを持たず、表示画面上にブランクが表示される行は、同一マップをプリンターに送信するときに省略されます。BMS のもとでは、画面のすべての行に少なくとも 1 つのフィールドを配置することによって、印刷形式を強制的に、表示画面と同じように見えるようにすることができます。そうではなく、空にしたい行には単一ブランクを含むフィールドを使用してください。また、BMS は行にデータがあるかどうかにかかわらず、すべての行に改行文字を使用するので、NLEOM を指定するとこの効果があります。

複数の送信

NLEOM の場合、連続した書き込みからのデータは位置決め情報を含んでいないので、それらのデータはバッファー内にスタックされます。しかし、BMS は、ACCUM オプションが使用されていない限り、NLEOM を指定した各 SEND のデータの終わりに EM (メッセージ終結) 文字を追加します。印刷を行う時は、NLEOM を指定した最初の SEND からのデータ (およびその時点までに未消去のバッファー内のすべてのデータ) のみを印刷するように、最初の EM 文字が印刷を停止します。最終的な効果は、ACCUM オプションを指定しない限り、NLEOM を指定した複数の SEND コマンドによって埋め込まれたバッファーは印刷することができないという点です。

ページ幅

BMS は常にその時点で、サイズがページ上の文字桁数になっている内部バッファーを使用して出力のページを構築します。(BMS がページ・サイズを判別する方法の説明については、 751 ページの『CICS プリンターの特性の判別』を参照してください。) バッファー制御オーダーを使用している場合には、端末定義では、ページ幅に 40、64、80、または装置の最大幅 (プラテン・サイズ) を指定しなければなりません。そうでない場合には、出力を正しく形式設定することができません。NLEOM

を使用している場合には、これに反して、端末定義では最大プラテン・サイズまでの任意のページ幅を指定することができます。

合計ページ・サイズ

バッファー制御オーダーを使用している場合には、バッファーがページのイメージとして使用されるので、生成される行数とページ幅がバッファー・サイズを超えてはいけません。各行の右方の未使用位置はヌル文字で表されます。しかし、NLEOM を使用した場合には、BMS はページ・サイズをバッファー容量に制限しません。BMS は端末用に定義されているページ・サイズに従ってページを構築してから、可能な場所で改行文字を使用してストリームを圧縮します。結果のストリームがバッファー容量を超える場合には、BMS は端末への複数の書き込みを使用して、それを送信します。

FORMFEED

FORMFEED オプションにより、BMS は、用紙の先端まで送る機能がプリンターにあるとして (関連する TYPETERM 定義の FORMFEED オプションによって) 定義されていれば、用紙送り印刷形式設定オーダー (X'0C') をバッファーの先頭に入れます。CICS は、この機能なしで定義されているプリンターの用紙送り要求を無視します。

画面の位置 (1,1) を使用するマップを使用して SEND MAP を実行した場合には、オーダーが上書きされて用紙送りは失われます。これは、ユーザーが NLEOM を使用していても、使用していなくても起こります。

SEND CONTROL コマンドで FORMFEED と ERASE を一緒に使用した場合には、NLEOM が存在しているかどうかによって結果が異なります。NLEOM が指定されていないと、SEND CONTROL FORMFEED ERASE が用紙送り文字とそれに続くページ全体のヌル行を送信します。プリンターは、これらのヌル行を単一ブランク行に置き換えることによって抑制します。NLEOM が指定されている場合には、非 3270 プリンターでの場合と同様に、全部ブランクのページになるように、同一コマンドで、用紙送り文字とそれに続く改行文字 (ページ上の各行に 1 個) が送信されます。

PRINTERCOMP オプション

プリンターに対して SEND TEXT を実行する場合には、ページ・サイズに影響するもう 1 つのオプションがあります。これは、個別の SEND TEXT コマンドではなく、実行中のトランザクションと関連した PROFILE に指定される PRINTERCOMP オプションです。(CICS が提供するデフォルト・プロファイルでは、PRINTERCOMP 値は NO です。)

PRINTERCOMP(NO) のもとでは、BMS は 3270 ディスプレイに送信される内容と変わらない印刷出力を生成します。ディスプレイの場合には、BMS は、各 SEND TEXT コマンドからのテキストの前に属性バイトを付け、各行を属性バイトによって始めます。これらの属性バイトは、画面上のスペースを取るため、PRINTERCOMP が NO の場合、BMS はプリンター用にそれらをブランクで置き換えます。PRINTERCOMP が YES の場合には、BMS はこれらのブランクを抑制し、ユーザーがプリンターの幅とバッファーの位置をすべて使用できるようにしま

す。テキストに組み込んだ改行文字は、PRINTERCOMP(YES) が指定された場合でも、PRINTERCOMP(NO) が指定されたときと同様に処理されます。

使用可能の線幅が 1 桁分減ることになっても、ディスプレイ装置との互換性のため、およびアプリケーションが異なるプリンター・タイプを使用する場合にも結果が変わらないようにするために、可能であれば、PRINTERCOMP(NO) を使用してください。

非 3270 CICS プリンター

非 3270 プリンターは、3270 データ・ストリームを受け入れない、SNA 文字セット (SCS) プリンターなどの、任意のプリンターです。非 3270 プリンターは 3270 ファミリー装置とすることができ、3287 および 3262 のような多くの装置は、制御装置での定義方法によって、3270 プリンターまたは SCS (非 3270) プリンターのどちらにもできます。

非 3270 プリンターはページ・バッファーを持っていないので、バッファー制御オーダーを認識しません。形式設定はすべて印刷制御オーダーによって行われます。3270 プリンターとの互換性のために、BMS はメモリー内にページのイメージを構成してその形式設定を行い、常に一度にページ全部を印刷します。しかし、関連するハードウェア・バッファーがないので、プラテン幅を超えていなければ、任意のサイズのページを定義することができます。NLEOM オプションを指定した 3270 プリンターの場合とまったく同様に、BMS はページを印刷するために必要な回数だけ伝送します。

BMS は、ブランクおよび NL (改行) 文字を使用してこれらのプリンター用の形式設定を行います。ユーザー端末の定義が用紙送りをサポートすることを示している場合には、さらに用紙送り (FF) 文字が使用されます。

また、端末定義に HORIZFORM オプションがあり、マップに HTAB 指定が含まれている場合には、BMS は形式設定のために水平タブも使用します。同様に、端末定義に VERTICALFORM が指定され、マップに VTAB が含まれている場合には、垂直タブも使用します。タブ文字によって、データ・ストリームをかなり短くすることができます。タブを使用する場合には、BMS は、現行タスクまたはそれ以前のタスクが既にプリンターにタブを設定してあることを前提とします。SCS プリンターでは、「*IBM 3270 Data Stream Device Guide*」に説明されているように、端末管理 SEND コマンドでタブを設定します。その他の非 3270 プリンターの場合には、該当する装置の手引きを調べる必要があります。

SCS プリンターに対する SEND TEXT では、BMS は、改行 (X'15') および属性設定 (X'28') を除く入力データ・ストリームで非 3270 制御コードを認識しません。その他の文字はすべて、表示文字であることが前提とされています。特に、BMS の下で透過性制御オーダー (X'35') を使用しようとすると、データ・ストリームが影響を受ける可能性があります。この制御オーダーでは、通常、その後に続くデータが無視されます (データの長さを含む次のバイトが無視されます)。しかし、BMS は X'35' 制御オーダーを認識しないため、透過性制御オーダーの次のデータを、データ・ストリームの通常の部分であるかのように処理します。

このデータが正しく処理されない場合、BMS はそのデータをデータ・ストリームから除去する可能性があります。例えば、透過シーケンスで検出された X'28' 文字

は、誤って属性設定制御オーダーであると認識されます。この場合、X'28' 文字の後に続く 2 バイトは、誤って属性の記述であると認識され、すべての 3 バイトがデータ・ストリームから除去されることがあります。また X'0C' 文字 (用紙送り) も、データ・ストリームから除去されがちです。BMS によって認識されると変更される可能性のある文字を含んでいる透過シーケンスなどのデータ・ストリームを送信する場合は、BMS ではなく、端末管理 SEND コマンドを使用することをお勧めします。

SCS 入力

SCS プリンターにおいても、「プログラム・アテンション」キーの形式では、入力の能力に制限があります。ただし、これらのキーは 636 ページの『アテンション・キー』 で説明されている PA キーと類似していません。代わりに、「APAK nn」の文字で構成された不定形式データ・ストリームを送信します。この場合、「nn」は 2 桁の PA キー番号 (例えば PA キー 1 の場合は「APAK 01」) です。

「APAK」(SCS 入力は他の PA キー入力とは異なるため、APAK はトランザクション ID であり、TASKREQ 属性値ではありません) という名前のトランザクションを定義することによって、このような入力を取り込むことができます。このトランザクションで起動されるプログラムによって、RECEIVE と入力の数値位置を実行することでどの PA キーを押したかが判別されます。

第 58 章 CICS におけるプリンターの使用

CICS プリンターおよび非 CICS プリンターに関するプログラミング。

CICS プリンターの特性の判別

複数のタイプの CICS プリンターをサポートするプログラムを作成している場合には、特定のプリンターの特性を判別する必要がある場合があります。端末との接続に関して一般的に説明したように、この目的には、ASSIGN および INQUIRE TERMINAL コマンドを使用できます。 603 ページの表 32 には、プリンターに特有のいくつかのオプションを含む、端末に適用される ASSIGN の各オプションがリストされています。

端末定義で、特にプリンターに適用される INQUIRE TERMINAL オプション、およびそれに対応するパラメーターが表 40 に示されています。

表 40. プリンター用の INQUIRE TERMINAL オプション

INQUIRE オプション	**TERMINAL または TYPETERM 定義の中のソース**	**説明**
PAGEHT	PAGESIZE(x,y) の x	ページ当たり行数 (代替画面サイズ端末の場合には、デフォルトのサイズを示す)
PAGEWD	PAGESIZE(x,y) の y	行当たり文字数 (代替画面サイズ端末の場合には、デフォルトのサイズを示す)
DEFPAGEHT	PAGESIZE(x,y) の x	デフォルト・モードのページ当たり行数 (代替画面サイズ端末専用)
DEFPAGEWD	PAGESIZE(x,y) の y	デフォルト・モードの行当たり文字数 (代替画面サイズ端末専用)
ALTPAGEHT	ALTPAGE(x,y) の x	代替モードのページ当たり行数 (代替画面サイズ端末専用)
ALTPAGEWD	ALTPAGE(x,y) の y	代替モードの行当たり文字数 (代替画面サイズ端末専用)
DEVICE	DEVICE	装置タイプ (使用可能な値については「*CICS System Programming Reference*」を参照)
TERMMODEL	TERMMODEL	端末の型式番号 (1 または 2 のいずれか一方)

BMS ページ・サイズ、3270 プリンター

BMS は端末定義および実行中のトランザクションのプロファイルの両方を使用して、CICS プリンターのページ・サイズを判別します。端末が代替画面サイズ機能を持っている場合には、プロファイルを使用して、デフォルトのサイズを使用するか、あるいは代替サイズを使用するかを判別します。(CICS のデフォルト・プロファイルには、画面の「デフォルト」サイズが指定されています。) 752 ページの表 41 には、使用される値がリストされています。

© Copyright IBM Corp. 1989, 2009

表 41. BMS ページ・サイズを定義するパラメーターの優先順位： BMS は、端末定義に指定されている該当列の最初の値を使用します。

代替サイズを使用する代替画面サイズを持つ端末	デフォルトのサイズを使用する代替画面サイズを持つ端末	代替画面サイズ機能を持たない端末
ALTPAGE	PAGESIZE	PAGESIZE
ALTSCREEN	DEFSCREEN	TERMMODEL
DEFSCREEN	TERMMODEL	(12,80)
TERMMODEL	(12,80)	
(12,80)		

「ページ」の定義は BMS に対して固有です。端末管理 SEND コマンドを使用して印刷している場合には、印刷形式設定によって、装置の物理的限界内でどのようなページを構成するかを定義します。バッファー・サイズを確認し、一度に送信可能なデータ量を判別する必要がある場合には、ASSIGN コマンドによって返される SCRNHT 値および SCRNWD 値からこれを判別できます。

複数のプリンター・タイプのサポート

異なったページ・サイズを持つプリンターをサポートするプログラムを作成している場合には、プログラム外部のページ・サイズのような装置依存関係を常に保持できるわけではありません。しかし、BMS は次の 2 つの方法で、この問題を援助します。

1. 一般的にはマップを参照し、BMS にタスクと関連した端末用に設計されたマップを選択させることができる。
2. SEND TEXT を使用している場合には、受信端末のページ・サイズを基にして、BMS がテキストをワード境界で行に分割する。また、各ページのヘッダーおよびトレーラー・テキストを要求することもできます。

関連概念

797 ページの『装置依存マップ』

CICS プリンターの使用

このタスクについて

示したように、印刷において頻繁に発生する 2 番目の問題は、プリンターの所有権に関することです。印刷の要求はしばしばディスプレイ端末のユーザーから送信されます。要求を処理し、印刷出力を生成するタスクはユーザーの端末と関連しているので、出力をプリンターに直接送信することはできません。

タスクが、使用したいプリンターを所有していない場合には、プリンターを所有しているタスクを作成して処理を行う必要があります。これを行うには、以下のようないくつかの方法があります。

1. START コマンドを使用してタスクを作成する。
2. タスクのトリガーとなる区画内一時データ・キューに書き込む。
3. BMS ROUTE コマンドで、出力をプリンターに向ける。
4. 画面コピーだけが必要な場合には、ISSUE PRINT コマンドを使用する。

START コマンドによる印刷

このタスクについて

印刷タスクを作成するための最初の手法は、印刷が必要なタスクで START コマンドを実行することです。コマンドでは、プリンターを TERMID オプションで START されたタスクによって要求された端末として名前を指定し、印刷するデータまたはそれを検索する指示を FROM オプションに渡します。START により、CICS は端末が使用可能である場合に、そのタスクのプリンシパル装置が指定された端末であるタスクを作成します。

START されたタスクによって実行される (ユーザーが提供する必要がある) プログラムは (RETRIEVE コマンドを使用して) 印刷するデータを取得してから、SEND、SEND MAP、または SEND TEXT を使用してそのデータを端末 (プリンター) に書き込みます。例:

```
(build output in OUTAREA, formatted as expected by the STARTed task)
   EXEC CICS START TRANSID(PRNT) FROM(OUTAREA) TERMID(PRT1)
             LENGTH(OUTLNG) END-EXEC.
```

図 135. 印刷するタスク (プリンター PRT1)

```
   EXEC CICS RETRIEVE INTO(INAREA) LENGTH(INLNG) END-EXEC.
 (do any further data retrieval and any formatting required)
   EXEC CICS SEND TEXT FROM(INAREA) LENGTH(INLNG) ERASE PRINT END-EXEC.
 (repeat from the RETRIEVE statement until a NODATA condition arises)
```

図 136. START されたタスク (トランザクション PRNT の実行)

現行印刷が完了する前に、別のタスクがデータを同一プリンターに送信した場合には、そのプリンターと関連したタスクは、送信されたすべてのデータを処理し終えるまでループします。これを行うと、CICS は、直前のタスクがまだ印刷中である間に到着した出力に対して、新規タスクを作成するオーバーヘッドを節約できます。未処理の START 要求がある限り、CICS がプリンター用に新規タスクを作成する場合の、最終的に得られる印刷内容に変化はありません。

一時データでの印刷

このタスクについて

印刷タスクを作成するための 2 番目の方法は、一時データを使うものです。CICS 区画内一時データ・キューは、「トリガー」と呼ばれるプロパティーを持つように定義できます。トリガーを持つキュー上の項目数がトリガー値に達した場合には、CICS がそのキューを処理するトランザクションを作成します。キュー定義によって、CICS は、このタスクがどのトランザクションを実行しているか、また、もしあれば、プリンシパル装置としてどの端末を必要としているかがわかります。

このメカニズムを使用して、プリンターを所有するタスクに対するデータを生成するタスクから、印刷データを入手することができます。一時データ・キューは、この方法で出力を指示する各プリンター用に定義されます。印刷するタスクは、(WRITEQ TD コマンドを使用して) その出力を必要なプリンターに関連付けられたキューに入れます。十分な項目がキューにあり、プリンターが使用可能な場合には、CICS がそのキューを処理するタスクを作成します。 (この目的のために、「十

分な」トリガー・レベルは通常は 1 項目だけとして定義されます。) トリガーされたタスクは (READQ TD コマンドを使用して) キューから出力を取り出し、SEND、SEND MAP、または SEND TEXT コマンドを使用して、それをプリンシパル装置 (プリンター) に書き込みます。

START されたプリンター・タスクの場合のように、トリガー処理されるタスクによって実行されるプログラムを提供する必要があります。CICS に付属して配布されるサンプル・プログラムには、「order queue print sample program」と呼ばれる、このようなプログラムの完全な例が含まれています。「*CICS 4.1 Sample Applications Guide*」にはこのプログラムの詳細が説明されていますが、要点を以下に示します。

印刷するタスク (プリンター PRT1):

```
(do any formatting or other processing required)
  EXEC CICS WRITEQ TD QUEUE('PRT1') FROM(OUTAREA)
            LENGTH(OUTLNG) END-EXEC.
```

トリガーされるタスク:

```
   EXEC CICS ASSIGN QNAME(QID) END-EXEC.
   EXEC CICS READQ TD QUEUE(QID) INTO(INAREA) LENGTH(INLNG)
             RESP(RESPONSE) END-EXEC.
   IF RESPONSE = DFHRESP(QZERO) GO TO END-TASK.
 (do any error checking, further data retrieval and formatting required)
   EXEC CICS SEND FROM(INAREA) LENGTH(INLNG) END-EXEC.
 (repeat from READQ command)
```

印刷タスクは、すべてのキュー (プリンター) に対して同一のコードが使用できるように、ハードコーディングされた値ではなく、ASSIGN コマンドを使用してそのキューの名前を判別します。

その START の対応コマンドと同様に、このタスクはキューが空であることを示す QZERO 条件を検出するまで、その読み取りおよび送信シーケンスでループします。これが START されたタスクによる効率上の問題になる限り、一時データにとって重要です。そうでない場合には、未処理のキュー項目が一定の条件のもとで累積されます。(トリガーによって一時データ・キューを処理するタスクの作成の詳細については、675 ページの『自動トランザクション開始 (ATI)』を参照してください。)

この技法を使用する場合には、単一の単位として印刷される出力が単一項目として、あるいは連続項目としてキューに現れるようにする必要があります。キュー項目と印刷出力の間に固定的な関係はありません。パッケージ配置はキューに書き込んでいるプログラムとキューから読み取っているプログラムの間には厳密な関係があります。しかし、タスクが一緒に印刷する必要がある複数の項目を書き込んだ場合には、それが終了する前に、他のタスクがそのキューに書き込まないようにしなければなりません。そうでない場合には、複数のタスクからの印刷出力がインターリーブすることがあります。

TD キューがリカバリー可能として定義されている場合には、CICS がインターリービングを防ぎます。タスクがリカバリー可能キューに書き込むと、最初のタスクがコミットするか、あるいは書き込んだ内容を除去する (SYNCPOINT またはタスクの終わり) まで、CICS は書き込む必要がある他のすべてのタスクを遅らせます。キューがリカバリー可能でない場合には、この機能を自分で実行する必要があります。1 つの方法は、最初のキュー項目を書き込む前に ENQUEUE を実行し、最後の

キュー項目を書き込んだ後に DEQUEUE を実行することです。(一時データ・キューの説明については、 673 ページの『第 48 章 一時データ管理』を参照してください。)

BMS ルーティングによる印刷

このタスクについて

タスクは、BMS ルーティングを使用して、そのプリンシパル装置以外のプリンターへの出力を入手することもできます。この技法は BMS 論理メッセージだけに適用され (ACCUM オプションまたは PAGING オプション)、したがって、論理メッセージを既に構築している場合には、大部分が該当します。

ルーティングしたメッセージを完了した場合には、CICS が経路リストに指定された各端末用にタスクを作成します。このタスクはプリンシパル装置として端末を持ち、ページを表示するための CICS 提供のトランザクション CSPG を使用して、出力をプリンターに送達します。ルーティングは START コマンドを使用する場合の効果と類似していますが、CICS では印刷するプログラムを提供しています。(ルーティングの詳細については、 853 ページの『第 68 章 メッセージ・ルーティング』を参照してください。)

非 CICS プリンターの使用

このタスクについて

CICS の外側で管理されるプリンターを使用するステップは、以下のとおりです。

1. 使用したいプリンターを制御するアプリケーションまたはサブシステムが必要とする方法で、出力を形式設定する。
2. 出力をプリンターを制御するアプリケーションまたはサブシステムに、そのアプリケーションが必要とする形式で送達する。
3. 必要な場合には、出力が印刷可能になっていることをそのアプリケーションに通知する。

非 CICS プリンターの書式設定

CICS の外部で管理される一部のプリンターの場合には、 756 ページの『非 CICS プリンターに関するプログラミング』 に説明されているように、BMS を使用して出力を形式設定できます。しかし、ほとんどのプリンターの場合には、そのプリンターを駆動するアプリケーションの形式設定要件を満たす必要があります。これはアプリケーションが指示した装置形式または中間形式となる場合があります。従来のライン・プリンターの場合には、形式設定は、行イメージを生成し、場合によっては紙送り制御文字を追加することです。

非 CICS プリンター: データの送達

通常、印刷データは、CICS とアプリケーションの両方にアクセス可能な中間ファイルにデータを入れることによって、CICS の外側のアプリケーションに送達されます。ファイル内の形式と同様、ファイルのタイプは、受信アプリケーションによって指示されます。通常、これは 756 ページの表 42 の 1 列目にリストされているものの 1 つです。表の 2 列目には、このようなデータを作成するためにどのグルー

プの CICS コマンドを使用することができるかが示されています。

表 42. 非 CICS プリンターに印刷データを転送するための中間ファイル

ファイル・タイプ	データを書き込む方式
スプール・ファイル	CICS スプール・コマンド (SPOOLOPEN、SPOOLWRITE など)、一時データ・コマンド (WRITEQ TD)、端末管理および BMS コマンド (SEND、SEND MAP など)
BSAM	CICS スプール・コマンド (SPOOLOPEN、SPOOLWRITE など)一時データ・コマンド (WRITEQ TD)
VSAM	CICS ファイル制御コマンド (WRITE)
DB2	EXEC SQL コマンド
IMS	EXEC DLI コマンドまたは CALL DLI ステートメント

非 CICS プリンターに関するプログラミング

VSAM、DB2、または IMS を使用している場合には、使用することができる CICS アプリケーション・プログラミング・コマンドは、使用しているファイルのタイプによって決まります。

しかし BSAM およびスプール・ファイルの場合には、選択項目があります。ファイルの CICS 定義 (またはそれがないこと) によって、使用するコマンドがどれかが決まります。このファイルは以下のいずれかにすることができます。

- 区画外の一時データ・キュー (一時データ・キューの詳細については、 673 ページの『第 48 章 一時データ管理』を参照してください)。
- 順次端末の出力側 (611 ページの『第 40 章 順次端末サポートの使用』および 794 ページの『3270 以外の端末のサポート』を参照してください)。
- スプール・ファイル (761 ページの『第 59 章 JES に対する CICS インターフェース』を参照してください)。

一時データ・キュー定義と順次端末定義の両方とも、CICS 開始 JCL の中の関連データ定義 (DD) ステートメントを指し、それが、ファイルが BSAM ファイルかスプール・ファイルかを判別するこの DD ステートメントです。CICS スプール・コマンドによって作成されるファイルは、使用前の定義を必要とせず、定義によってスプール・ファイルとなります。

印刷アプリケーションが BSAM またはスプール・ファイル入力を受け入れる場合には、ファイルを CICS に対して定義する方法を決める時に考慮する要因がいくつかあります。

システム定義

SPOOLOPEN コマンドによって作成されるファイルは CICS またはオペレーティング・システムに対して定義する必要はありません。これに対して、一時データ・キューおよび順次端末は使用する前に両方に対して定義しなければなりません。

タスク間の共用

一時データ・キューとして定義されるファイルはすべてのタスク間で共有されます。これによって、複数のタスクで印刷ファイルを作成することができますが、ユーザー・タスクが一緒に印刷する必要がある複数のレコード (例

えば、単一レポートに印刷する行など) をキューに書き込む場合には、エンキュー論理を組み込んで、他のタスクがレコードをキューの中のレコードの間に書き込めないようにする必要があります。この要件は、 753 ページの『一時データでの印刷』の区画内キューの場合と同じです。しかし、区画外一時データの場合には、CICS はリカバリーのためのソリューションを提供しないので、ユーザー・プログラムが自分自身でレコードが散在するのを防ぐ必要があります。

これに対して、SPOOLOPEN によって作成されたファイルはこれを作成したタスクによってのみ書き込みが可能です。これにより、出力のインターリービングの危険性はなくなりますが、タスク間のファイルの共用もできなくなります。

順次端末と関連したスプール・ファイルは、一度に 1 つのタスク (プリンシパル装置としての端末を持つタスク) にしか書き込むことができません。また、これはインターリービングも防ぎますが、タスクはファイルを順次に共用することができます。

印刷の解放

BSAM およびスプール・ファイルの両方とも、オペレーティング・システムのためにクローズして、それを CICS から受信アプリケーションに渡す必要があり、したがって関連のファイルがクローズされるまで印刷は始まりません。SPOOLOPEN によって作成されるファイルは、SPOOLCLOSE コマンドによって既にクローズされていない限り、タスク終了時に自動的にクローズされます。これに対して、区画外一時データ・キューは、あるタスクが SET コマンドによって明示的にクローズするまでオープンされたままです。(後から使用する場合には、別の SET によって再オープンしなければなりません。) したがって、一時データは、追加のプログラミングを行うことで、処理のためにファイルを解放してさらに多くの制御を与えます。

順次端末の出力を表すファイルは、CICS がシャットダウンされるまで、自動的にはクローズされず (したがって印刷用に解放されず)、CICS はそれを早目にクローズする機能を提供しません。順次端末を使用してデータを CICS の外側で制御されるプリンターに渡す場合には、BMS を使用するために行うことがあるときに、この制限に注意する必要があります。

形式設定

ファイルを順次端末として定義する場合には、BMS を使用して、出力を形式設定することができます。この機能により、CICS の外部で管理されるプリンター (例えば、MVS ジョブ入力サブシステム (JES) によって管理されるライン・プリンターなど) に、CICS ディスプレイおよびプリンター端末に使用するマップと同一のマップを使用できます。

このオプションを使用する場合には、BMS は常に端末定義の中のページ・サイズを使用して出力のページを一度に送信すること、および順次端末からの出力を表すデータ・セットは CICS がシャットダウンされるまで解放されないことを忘れないでください。

スプール・ファイルの限界

オペレーティング・システムは、順次番号を割り当ててスプール・ファイルを識別します。この番号には上限があります。番号は後から再利用されます。限界は非常に大きいのが普通ですが、非常に長時間 (CICS で可能な限り) 実行され、非常に多く (CICS のもとでアプリケーションが可能な限り)

のスプール・ファイルを作成するジョブの場合に、この限界を超えることがあります。非常に多くのスプール・ファイルを生成するアプリケーションを作成している場合は、システム・プログラマーに相談して、システムの限界内であることを確認してください。新規スプール・ファイルは、各 SPOOLOPEN ステートメントおよびスプール・ファイルに定義された一時データ・キューの各オープンで作成されます。

印刷アプリケーションの通知

データを CICS の外側の印刷アプリケーションに送達する場合には、データを処理可能にしたことをアプリケーションに通知する必要がある場合があります。アプリケーションが自動的に実行され、データを探すことがわかっている場合には、これを行う必要はありません。例えば、MVS ジョブ入力システム (JES) によって所有されたプリンターで印刷をするために必要なことは、適切なルーティング情報によってスプール・ファイルを作成することだけです。JES が後のことをします。

しかし、処理を実行するためにジョブを実行依頼することが必要になる場合があり、そうでない場合には、それに関する処理があることを実行アプリケーションにシグナルを送ります。

CICS タスクからバッチ・ジョブを実行依頼するためには、そのジョブの JCL が入っているスプール・ファイルを作成する必要があり、このファイルを JES 内部読み取りプログラムに送る必要があります。順次端末を使用する場合には、以前に示したように、CICS がシャットダウンされるまでそのジョブは実行されませんが、ファイルは、756 ページの表 42 でスプール・ファイル用にリストされている 3 つの方法のいずれかで作成できます。スプール・コマンドによって書き込まれるファイルの場合には、ファイルを JES 内部読み取りプログラムにルーティングする情報は SPOOLOPEN コマンドに指定します。一時データ・キューおよび順次端末の場合には、ルーティング情報は、「JOB カード」ファイルの最初のレコード上に現れます。

印刷する出力はバッチ・ジョブに (その入力として) 組み込むか、あるいはジョブが受け入れるデータ・ストレージの任意の形式を通じて別個に渡すことができます。

表示画面の印刷

印刷要件が表示画面をプリンターにコピーすることである場合には、既に説明したものに追加の選択項目があります。これらの項目には、端末ハードウェアから提供されるものと、CICS から提供されるものがあります。CICS サポートの一部もハードウェア機能によって異なり、そのためにユーザー・オプションは関連する端末のタイプによって異なり、端末を CICS に定義する方法によって異なる場合もあります。コピーについて詳しくは、「*CICS Resource Definition Guide*」を参照してください。

CICS 印刷キー

このような最初のオプションは **CICS 印刷キー** (**ローカル・コピー・キー**とも呼ばれます) です。これにより、端末が 3270 ディスプレイまたは 3270 互換モードのディスプレイとすると、ユーザーはプログラム・アテンション・キーを押すことによって画面の印刷コピーを要求することができます。印刷キー・サポートは CICS

ではオプションです。システム・プログラマーがそれを組み込むかどうか、およびどのキーを割り当てるかを決めます。デフォルト・キーは PA1 です。 (「*CICS System Definition Guide*」の PRINT オプションを参照してください。)

印刷キーは、適格として定義されているものの中で最初に使用可能なプリンターに、表示画面をコピーします。どのプリンターが適格かは、要求を送信するディスプレイ端末の定義によって以下のように異なります。

- 「プリンター・アダプター」機能なしと定義された VTAM 3270 ディスプレイの場合には、端末定義の PRINTER オプションおよび ALTPRINTER オプションに指定されているプリンターが適格になる。PRINTER が使用されるのは、使用可能な場合で、ALTPRINTER が 2 番目の選択項目です。両方とも利用不能の場合には、PRINTER が使用可能になったときに実行するために、要求はキューに入れられます。
- 3790 および 3650 ホスト会話型 (3270) 論理装置の 3270 互換モードの場合には、同一の選択項目が適用される。
- プリンター・アダプター機能を持つものと定義されている VTAM 3270 ディスプレイの場合には、コピーは、ディスプレイと同じ制御装置上のプリンターに限定される。制御装置内のプリンター許可マトリックスはプリンターの適格性を判別します。
- プリンター・アダプター機能を持つ 3790 の 3270 互換モード論理装置の場合には、3790 が適格性を判別し、プリンターをコピーのために割り振る。
- プリンター・アダプター機能を持つ 3275 の場合には、印刷キーは現在 3275 ディスプレイ・バッファーに入っているデータをディスプレイに接続されている 3284 に印刷します。

CICS がプリンターを明示的に選択する場合には、上記の最初の 3 項に該当するように、プリンターをサービス中にして、タスクに接続しないで、CICS 印刷キー要求に対して「使用可能」にする必要があります。制御装置またはサブシステムが割り当てを行う場合には、可用性および状況はサブシステムによって判別されます。装置のブラケット状態は通常、それが使用可能か、あるいはそうでないかを判別します。

ISSUE PRINT および ISSUE COPY

アプリケーションは、ユーザーと同様に、ISSUE PRINT コマンドおよび ISSUE COPY コマンドを使用して、プリンターへの画面のコピーを開始できます。ISSUE PRINT は、ユーザーが CICS 印刷キーを押す操作をシミュレートし、プリンターの適格性および可用性は CICS 印刷キー要求の場合と同じです。

また、コピーする端末を所有するタスクに対立するものとして、プリンターを所有するタスクで画面をコピーするために使用することができるコマンドもあります。これが ISSUE COPY コマンドです。これは TERMID オプションに指定されている端末のバッファーを、タスクを発行したプリンシパル装置のバッファーにコピーします。コピーが行われてからのコピーの方法および印刷の開始は、ISSUE COPY コマンドの CTLCHAR オプションに定義される「コピー制御文字」によって制御されます。この制御文字のビット設定については、「*IBM 3270 Data Stream Device*

Guide」を参照してください。バッファーがコピーされる端末、およびプリンターは、両方とも 3270 論理装置である必要があり、同一の制御装置上にある必要があります。

ハードウェア印刷キー

一部の 3270 端末は**ハードウェア印刷キー**も持っています。このキーを押すと、ディスプレイと同じ制御装置上で、最初に使用可能で適格なプリンターに画面がコピーされます。この機能は制御装置によって全面的に実行されます。この制御装置の構成および端末状況情報が適格性と可用性を判別します。どのプリンターも使用可能でない場合には、要求が失敗します。ユーザーには、画面の左下隅に記号によって通知され、要求は後からやり直す必要があります。

BMS 画面コピー

CICS およびハードウェア印刷キーの両方とも適格プリンターの事前定義セットへの画面コピーに限定され、複数のプリンターが適格になっている場合には、選択は他のタスクによるプリンターの使用によって異なります。BMS 論理メッセージの一部として作成される画面の場合には、さらに一般的な画面コピー機能が使用可能です。ユーザーは、論理メッセージを表示する CICS 提供のトランザクション CSPG の「ページ・コピー」オプションを使用して、このような画面のすべてを印刷できます。ページ・コピーによって、特定のプリンターの名前を指定して出力を受信します。またそのプリンターはディスプレイと同じ制御装置上にある必要はありません。CSPG の説明については、「*CICS Supplied Transactions*」のCSPG - ページの取得を参照してください。

第 59 章 JES に対する CICS インターフェース

CICS は、CICS アプリケーションが**スプール**・ファイルを作成したり取り出したりできるように、**JES** (MVS のジョブ入力サブシステム・コンポーネント) に対するプログラミング・インターフェースを提供します。スプール・ファイルは、JES によって管理され、スプール・ファイルを作成するジョブと装置による実際の処理との間で、低速**周辺**装置 (プリンター、穿孔装置、プロッター) に向けられた出力をバッファーするために使用されます。カード読み取り装置からの入力ファイルもまたスプール・ファイルで、装置とデータを使用するジョブとの間のバッファーとして提供されます。

インターフェースは以下の 5 つのコマンドで構成されています。

- SPOOLOPEN INPUT。入力用のファイルをオープンします。
- SPOOLOPEN OUTPUT。出力用のファイルをオープンします。
- SPOOLREAD。入力ファイルから次のレコードを取り出します。
- SPOOLWRITE。出力ファイルに 1 つのレコードを追加します。
- SPOOLCLOSE。ファイルをクローズし、JES による後続の処理のために解放します。

ここでいう「入力」および「出力」は、CICS の観点からの用語です。つまり、あるジョブにとってのスプール出力は、もう一方のジョブまたは JES プログラムにとっては常にスプール入力であるということです。

これらのコマンドは、JES の JES2 または JES3 のいずれかの形式とともに使用することができます。ただし、一部の制約事項は、それぞれに適用されます (763 ページの『スプール・インターフェースに関する制約事項』を参照)。JES という用語は両方を意味します。その他の製品の要件をサポートして、JES 遠隔スプーリング通信サブシステム (RSCS) ネットワークを介して接続された他のシステムとファイルを交換することができます。

スプール・コマンドを使って、以下のタイプのことを行うことができます。

- 印刷または JES による他の処理のための (出力) ファイルの作成。JES は、高速プリンターおよびカード読み取り装置など、オペレーティング・システムの「ユニット・レコード」装置のほとんどを管理します。これらの機能を使用するために、処理するデータをスプール・ファイルで JES に渡します。 762 ページの図 137を参照してください。

© Copyright IBM Corp. 1989, 2009

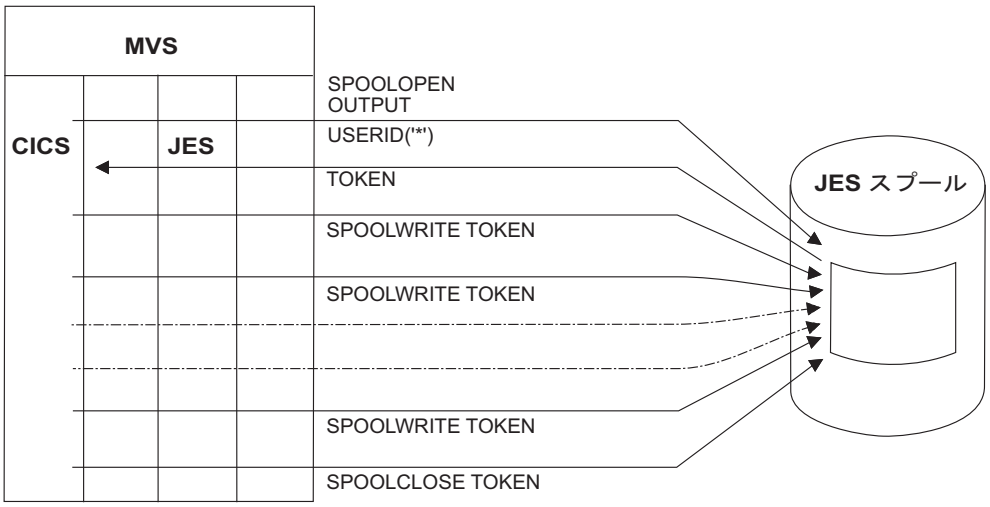


図 137. ファイルを作成して JES スプールに書き込む

- MVS に対してバッチ・ジョブを実行依頼する。JES「内部読み取りプログラム」に送信されたスプール・ファイルは、完全なジョブとして扱われ、実行されます。
- MVS のもとで実行する (CICS の外側の) 他のジョブにデータを渡すための (出力) ファイルの作成。
- そのようなジョブから渡されたデータの取り出し。図 138を参照してください。

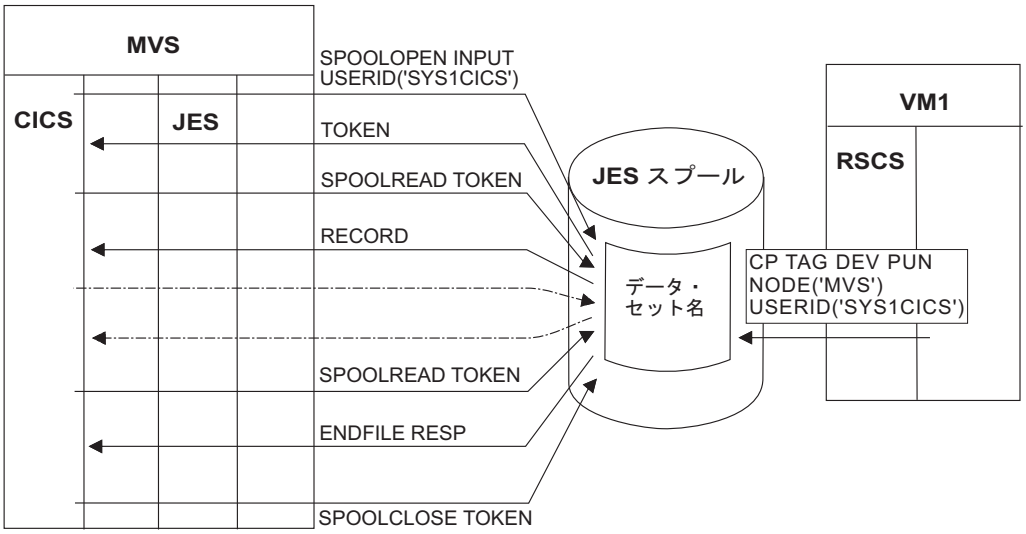


図 138. JES スプールからのデータの取得

- CICS が実行中であるオペレーティング・システム以外の VM、VSE/ESA™、または MVS システムなどのオペレーティング・システムにデータを渡すためのファイルの作成。 763 ページの図 139を参照してください。

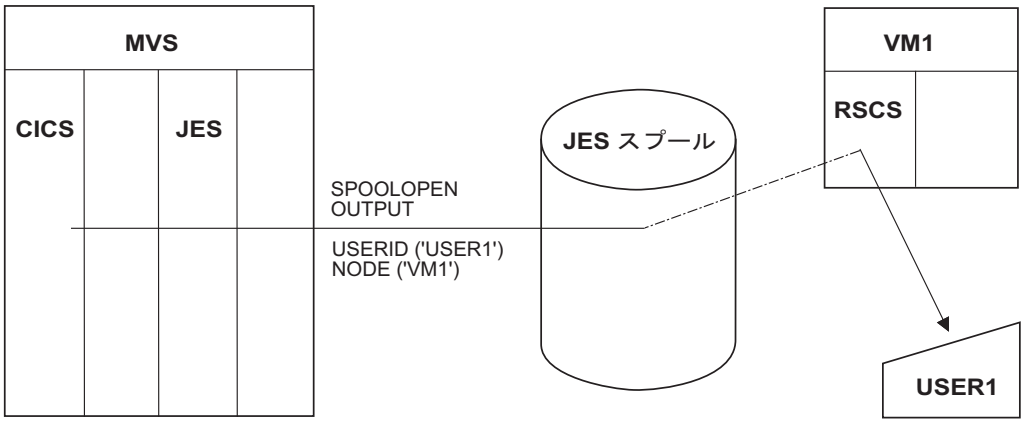


図 139. リモート宛先への書き込み済みファイルの送信

JES に対する CICS インターフェースの使用

このタスクについて

JES に CICS インターフェースを使用するには、CICS 始動 JCL に DFHSIT SPOOL=YES システム初期設定パラメーターを定義する必要があります。

EXEC CICS SPOOLCLOSE、SPOOLOPEN、SPOOLREAD、および SPOOLWRITE の各コマンドに、RESP または NOHANDLE を指定する必要があります。RESP により、HANDLE CONDITION と 1 対 1 で対応するようになります。RESP をコーディングしないと、プログラムが異常終了します。RESP2 オプションをコーディングすることもできます。

レコードが 1000 より多い SYSOUT データ・セットを処理するトランザクションは、INPUT または OUTPUT のいずれの場合にも、残りの CICS のパフォーマンスに影響を及ぼす可能性があります。そのようなトランザクションを避けられない場合は、システム・パフォーマンス全般を慎重に評価する必要があります。残りの CICS が受ける影響を受け入れられない場合は、ペーシング機構を導入する必要があります。

JES スプール・ファイルへのアクセスは、すべて 1 つの作業論理単位内で完了する必要があります。EXEC CICS SYNCPOINT コマンドを暗黙的に実行すると、オープンしている任意のレポートに SPOOLCLOSE コマンドが出されます。

スプール・インターフェースに関する制約事項

JES には、アプリケーションを設計する際に考慮すべき内部限度があります。JES2 に適用するもの、JES3 に適用するもの、両方に適用するものがあります。以下の点に特に注意してください。

- JES2 には、単一のジョブ (CICS など) が作成することができるスプール・ファイルの合計数に上限があります。実行中に CICS がこの限度を超えると、後続の SPOOLOPEN OUTPUT コマンドは ALLOCERR 状態で失敗します。

- JES3 にはそのような明示的な限度はありませんが、JES2 と JES3 の両方に対して、作成された各ファイルの制御情報の中には、CICS の実行全体を通して存続するものがあります。この理由によって、非常にたくさんのスプール・ファイルを作成すると JES リソースを圧迫します。そのようなアプリケーションを設計する前に、システム・プログラマーに相談してください。
- スプール・ファイルは、処理されるまでは他のリソース (バッファー、キュー・エレメント、ディスク・スペース) を必要とします。非常に大きいファイルやその宛先での処理のために長い時間待機する可能性のあるファイルを作る場合、システム担当者に相談する必要があります。
- ローカル・スプール・ファイルを指定し、OUTDESCR オペランドで NODE オペランドと USERID オペランドを指定変更する場合は、NODE('*') および USERID('*') をコーディングします。NODE('*') とその他のユーザー ID を同時に使用しないでください。NODE オペランドと USERID オペランドで明示の ID を指定する場合は、OUTDESCR オペランドはこれを指定変更することができません。
- CICS によって作成されたデータ・セットが JES で HELD 状況にならないようにシステムを定義します。EXEC CICS SPOOLOPEN INPUT コマンドが発行されると、CICS は、HELD 状況でデータ・セットを検索しません。

出力スプール・ファイルの作成

このタスクについて

出力スプール・ファイルを作成するには、SPOOLOPEN OUTPUT コマンドを実行してプログラムを開始し、出力データ・セットを割り振ります。JES は、コマンドの NODE オプションおよび USERID オプションによって、完了時にファイルで何を実行すればいいかがわかります。また、必要に応じて形式設定および他の処理を JES に送信するその他のオプションもあります。SPOOLOPEN は、固有のトークンを TOKEN フィールドに返します。このトークンは、書き込み中のファイルを識別するために、後続のすべての SPOOLWRITE コマンドおよび SPOOLCLOSE コマンドで使用する必要があります。

その後、SPOOLOPEN OUTPUT コマンドで返されたトークン値を指定する SPOOLWRITE コマンドによって、タスクはファイルにデータを入れます。スプール・ファイルは順次であり、それぞれの SPOOLWRITE ごとに 1 つのレコードがファイルに追加されます。ファイルが完了すると、タスクは、ファイルを識別するトークンを使用して SPOOLCLOSE を実行することによって、送達または処理のためにファイルを JES に解放します。

タスクは複数の出力スプール・ファイルを作成することができ、一度に複数のファイルをオープンすることができます。異なるファイル上の操作は、トークンによって別々に維持されます。しかし、スプール・ファイルはタスク間で、あるいは同じタスク内の作業論理単位にまたがって共有されることはできません。スプール・ファイルに書き込むことができるのは、そのファイルをオープンしたタスクのみです。SYNCPOINT コマンドやタスクの終了前にそのタスクがファイルのクローズに失敗すると、CICS がこれらの時点で自動的にファイルをクローズします。

ノードがリモート・システムである場合、データ・セットは宛先ユーザー ID に対する JES スプール上でキューに入れられます。この宛先ユーザーの ID は、

SPOOLOPEN OUTPUT USERID パラメーターで指定されたものです。ノードがリモート VM システムである場合、データは同じ USERID パラメーターに指定された ID の VM RDR キューに入れられます。

注: 実行依頼したジョブをできるだけすぐに実行したい場合は、最初の 5 文字に /*EOF を含むレコードでスプール・ファイルを終了してください。このステートメントによって、JES は、現行のバッファーを埋めてから解放するために他のレコードを待つことなく、処理のためにファイルを解放します。

MVS 内部読み取りプログラムの使用

USERID パラメーターを使用すると、出力を MVS 内部読み取りプログラムに書き込むように指定することができます。この目的で CICS SPOOL コマンドを使用するには USERID(『INTRDR』) を指定して、明示ノード名も使用します。NODE('*') は使用しないでください。 INTRDR は、内部読み取りプログラムを識別する IBM の予約名です。 USERID(『INTRDR』) を指定する場合、SPOOLWRITE コマンドで書き込まれる出力レコードは JOB ステートメントで始まる JCL ステートメントにする必要があります。必ず SPOOLOPEN コマンド上で NOCC オプションも指定しておいてください。システムは、内部読み取りプログラム用の出力レコードを、アドレス・スペースのバッファーに入れます。このバッファーがいっぱいのときは、JES はスプールに内容を入れ、あとでスプールからジョブを取り出します。(スプール・ファイルの命名についての詳細は、767 ページの『スプール・ファイルの識別』を参照してください。)

入力スプール・ファイルの読み取り

このタスクについて

スプール・ファイル読み取りのコマンド・シーケンスは、作成の場合のシーケンスと同様です。ファイルを選択する SPOOLOPEN INPUT コマンドから開始します。次に SPOOLREAD コマンドを使用して各レコードを取り出します。ファイルをすべて読み取るか、または必要な量の読み取りを完了したら、SPOOLCLOSE コマンドを使用して処理を終了します。 CICS は、ユーザーが特定のファイルをオープンすると、出力ファイルをオープンしたときのように、そのファイルを識別するトークンを提供するので、ユーザーはファイルに対するすべての後続のコマンド上でこのトークンを使用します。

出力スプール・ファイルと同様に、入力スプール・ファイルもこれをオープンしたタスクに対して排他的です。最初のタスクがこれをクローズするまでは、他のタスクはこれを使用することができません。ファイルは、それをオープンしたのと同一の作業論理単位で読み取りをされなければならず、タスクがそれをクローズしなかった場合には、SYNCPOINT コマンドまたはタスク終了時に、CICS が自動的にこれをクローズします。ただし、ユーザーのタスク (あるいは他のタスク) が、もう一度最初からファイルを読み取ることのできるような方法でユーザーがファイルをクローズすることもできます。

出力ファイルと異なり、入力ファイルの場合はタスクは一度に 1 つのスプール・ファイルしかオープンできません。さらに、どのようなときでも、入力のためにはた

った *1* つ の CICS タスクしかファイルをオープンすることはできません。この入力スプール・ファイルの単一スレッドは、プログラミング上でいくつかの意味のあることです。

- スプール・ファイルを読み取るタスクをオープンにしておくのはできるだけ短時間でなければならず、タスク終了の処理の一部としてファイルを CICS にクローズさせるのではなく、明示的にクローズすべきです。個々のレコードの処理が長い場合、他の形式のストレージにファイルを転送する必要があるかもしれません。
- 他のタスクがスプール・ファイルを読み取り中である場合、SPOOLOPEN INPUT コマンドは SPOLBUSY 状態で失敗します。これはエラーではありません。しばらく待ってもう一度試みてください。
- 複数の入力ファイルを読み取る場合、入力スレッドを独占したり、それを必要とする他のタスクをロックアウトしないように、クローズから次のオープンまでにタスクを少し遅らせる必要があります。

リモート・アプリケーションは、CICS トランザクション用のファイルを、CICS が常駐するシステムの特定のユーザー名にルーティングする必要があります。これを実行する場合は、VM システムで使用される CP コマンドの例について 762 ページの図 138を参照してください。図には、データの検索に使用する EXEC CICS SPOOL コマンドも示されています。

CICS トランザクションは SPOOLOPEN コマンドを実行し、USERID パラメーターに書き出しプログラム名と、オプションで書き出しプログラム名内の出力のクラスを指定します。通常の応答は、以下のとおりです。

1. この外部書き出しプログラムへの入力はありません。
2. 単一スレッドが使用中です。
3. ファイルは検索のためユーザーに割り振られ、CICS によって返される「トークン」によって識別されます。トークンは、データ・セットの検索のため、各 SPOOL コマンドに組み込まれている必要があります。

(1) と (2) の場合、トランザクションは、適切なインターバルの後でそれ自身を再始動して SPOOLOPEN を再試行します。

(3) の場合、トランザクションは SPOOLREAD コマンドによってファイルを取得し、できるだけすぐに SPOOLCLOSE を実行してその他のユーザーのためにパスを解放します。入力パスが**単一スレッド化**されているため、これは JES からの**入力**の場合に特に重要です。インターフェースを使用しているトランザクションが複数ある場合、それらのファイルは、単一の書き出しプログラム内で異なる書き出しプログラムや異なるクラスを使用することによって区別されます。さらに、トランザクションが終了するか、または SPOOLCLOSE と後続の SPOOLOPEN の間の短い時間待機することを確認してください。これを行わないと、1 つのトランザクションによって別のトランザクションのインターフェースの使用が妨げられます。

JES 出口

JES2 と JES3 は両方とも、着信ファイルのスクリーニング方法を提供します。JES2 の場合は、TSO/E 対話式データ伝送機能スクリーニングおよび通知出口が使用されます。JES3 の場合は、着信ネットデータ・ファイル検証出口です。

インストールが使用するこれらの出口を検討して、JES への CICS インターフェースを使って読み取られるファイルが正しく処理されることを確認する必要があります。

スプール・ファイルの識別

入力スプール・ファイルは、SPOOLOPEN INPUT コマンド上の USERID および CLASS オプションによって識別されます。

入力では USERID は JES の外部書き出しプログラムの名前です。外部書き出しプログラムは、同じ宛先または処理を持つスプール・ファイルのグループを表す、JES 始動時に JES に定義された名前です。JES が自己処理をするファイルでは、通常外部書き出しプログラムは、例えばプリンターなどの特定のハードウェア装置に関連づけられています。これらの書き込みプログラムの名前は、JES の使用のために予約されてます。

アプリケーション間でファイルを転送するためには、CICS タスクがスプール・ファイルを読み取るときと同様に、唯一の命名要件は、送信側が使用した名前を受信側 (CICS タスク) が知っているということと、受信側のオペレーティング・システム内の他のどのアプリケーションも同じ名前を他の目的で使用しないということです。 CICS タスクが対象外のスプール・ファイルを読み取らないようにするため、CICS では、ユーザーが指定した外部書き出しプログラムの名前の最初の 4 文字が、自身の VTAM アプリケーション ID と一致していることが必要になります。その結果、ファイルを CICS に向けるジョブまたはシステムは、CICS のアプリケーション ID の最初の 4 文字で始まる外部書き出しプログラム名にファイルを送らなければなりません。

JES は外部書き出しプログラムのために、1 文字の CLASS 値によってファイルをカテゴリー化します。 SPOOLOPEN INPUT コマンドでクラスを指定すると、ユーザーが名前を指定した外部書き出しプログラム用のそのクラスの中で最初の (一番古い) ファイルを入手することになります。クラスを省略すると、その書き出しプログラム用のすべてのクラスの中で最も古いファイルを入手することになります。送信側はクラスを割り当てます。「A」は、送信側がクラスを指定しない場合に使用されます。

出力では、NODE 値および USERID 値の両方で SPOOL ファイルの宛先を識別します。NODE は、(MVS、VM のような) オペレーティング・システムの名前です。システムは、この名前で、CICS が実行されている MVS システム内の VTAM に認識されています。

USERID の意味は、オペレーティング・システムによって異なります。VM では特定ユーザーです。MVS では JES 外部書き出しプログラムまたは他の JES 宛先、TSO ユーザー、あるいはそのシステムで実行中の他のジョブであるかもしれません。そのような宛先の 1 つが JES 内部読み取りプログラムで、通常 INTRDR という予約名です。MVS システムにジョブの実行依頼をしたい場合、スプール・ファイルをその内部読み取りプログラムに書き込みます。このファイルは、カード読み取り装置または TSO を通して実行依頼されたジョブと同じ形式および同じシーケンスで、ジョブの実行に必要なすべての JCL ステートメントを含んでいなければなりません。

次のものは、内部読み取りプログラム用の SPOOLOPEN を使用する COBOL プログラムの例です。この例では、(紙送り制御用の先頭文字を使用を防ぐために) NOCC オプションを指定し、かつ JCL レコード形式を使用しなければなりません。

```
DATA DIVISION.
WORKING-STORAGE SECTION.
  01  OUTPUT-FIELDS.
      03  OUTPUT-TOKEN    PIC X(8)  VALUE LOW-VALUES.
      03  OUTPUT-NODE     PIC X(8)  VALUE 'MVSESA31'.
      03  OUTPUT-USERID   PIC X(8)  VALUE 'INTRDR  '.
      03  OUTPUT-CLASS    PIC X     VALUE 'A'.
PROCEDURE DIVISION.
   EXEC CICS SPOOLOPEN OUTPUT
      TOKEN(OUTPUT-TOKEN)
      USERID(OUTPUT-USERID)
      NODE(OUTPUT-NODE)
      CLASS(OUTPUT-CLASS)
      NOCC
      PRINT
      NOHANDLE
   END-EXEC.
```

図 140. 内部読み取りプログラムに SPOOL コマンドを使用する COBOL プログラムの例

OUTDESCR は、MVS JCL の OUTPUT ステートメントに対するパラメーターのストリングのアドレスを含む、フィールドのアドレスに設定するポインター変数を指定します。

次のものは、OUTDESCR オペランドを使用する COBOL プログラムの例です。

```
WORKING-STORAGE SECTION.
01 F.
02 W-POINTER USAGE POINTER.
02 W-POINTER1 REDEFINES W-POINTER PIC 9(9) COMP.
01 RESP1 PIC 9(8) COMP.
01 TOKENWRITE PIC X(8).
01 ....
01 W-OUTDIS.
02 F PIC 9(9) COMP VALUE 43.
02 F PIC X(14) VALUE 'DEST(A20JES2 )'.
02 F PIC X VALUE ' '.
02 F PIC X(16) VALUE 'WRITER(A03CUBI)'.
02 F PIC X VALUE ' '.
02 F PIC X'11' VALUE 'FORMS(BILL)'.
LINKAGE SECTION.
01 DFHCOMMAREA PIC X.
01 L-FILLER.
02 L-ADDRESS PIC 9(9) COMP.
02 L-OUTDIS PIC X(1020).
PROCEDURE DIVISION.
     EXEC CICS GETMAIN SET(W-POINTER) LENGTH(1024)
          END-EXEC.
     SET ADDRESS OF L-FILLER TO W-POINTER.
     MOVE W-POINTER1 TO L-ADDRESS.
     ADD 4 TO L-ADDRESS.
     MOVE W-OUTDIS TO L-OUTDIS.
     EXEC CICS SPOOLOPEN
                OUTPUT
                PRINT
                RECORDLENGTH(1000)
                NODE('*')
                USERID('*')
                OUTDESCR(W-POINTER)
                TOKEN(TOKENWRITE)
                RESP(RESP1)
                NOHANDLE
     END-EXEC.
     EXEC CICS SPOOLWRITE
              .
              .
              .
```

注:

1. GETMAIN コマンドは、必ずコーディングしてください。
2. L-FILLER は呼び出し側プログラムから渡されるパラメーターではありません。L-FILLER 用の BLL は SET ADDRESS に置換されます。それから、GETMAIN 域のアドレスは、L-ADDRESS になる L-FILLER が指す最初の語 (つまり自分自身を指すようになる) に移動します。それから L-ADDRESS に 4 を足して変更し、そのアドレスのすぐ後ろにある領域 (L-OUTDIS) を指すようにします。すると、L-OUTDIS は OUTDESCRIPTOR DATA で埋められます。こうして、W-POINTER は、OUTDESCR データを指すポインターを持つ領域を指すようになります。

SPOOL コマンドの例

COBOL

```
   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01  RESP             PIC 9(8) BINARY.
   01  RESP2            PIC 9(8) BINARY.
   01  TOKEN            PIC X(8).
   01  OUTLEN           PIC S9(8) BINARY VALUE +80.
   77  OUTPRT           PIC X(80) VALUE
       'SPOOLOPEN FUNCTIONING'.
   01  PARMSPTR              POINTER.
   01  PARMS-AREA.
       03  PARMSLEN      PIC S9(8) BINARY VALUE 14.
       03  PARMSINF      PIC X(14) VALUE
            'WRITER(MYPROG)'.
       03  PARMADDR                POINTER.
   PROCEDURE DIVISION.
       SET PARMSPTR TO ADDRESS OF PARMS-AREA
       SET PARMADDR TO PARMSPTR
       SET PARMSPTR TO ADDRESS OF PARMADDR
       EXEC CICS SPOOLOPEN OUTPUT
                 NODE ('*')
                 USERID ('*')
                 RESP(RESP) RESP2(RESP2)
                 OUTDESCR(PARMSPTR)
                 TOKEN(TOKEN)
                 END-EXEC
       EXEC CICS SPOOLWRITE
                 FROM(OUTPRT)
                 RESP(RESP) RESP2(RESP2)
                 FLENGTH(OUTLEN)
                 TOKEN(TOKEN)
                 END-EXEC
       EXEC CICS SPOOLCLOSE
                 TOKEN(TOKEN)
                 RESP(RESP) RESP2(RESP2)
                 END-EXEC.
```

PL/I

```
DCL
  RESP FIXED BIN(31),
  RESP2 FIXED BIN(31),
  TOKEN CHAR(8),
  OUTLEN FIXED BIN(31) INIT(80),
  OUTPRT CHAR(80) INIT('SPOOLOPEN FUNCTIONING'),
  PARMADDR POINTER,
  PARMSPTR POINTER;
DCL
  1 PARMS,
    2 PARMSLEN FIXED BIN(31) INIT(14),
    2 PARMSINF CHAR(14) INIT('WRITER(MYPROG)')
      ALIGNED;
PARMADDR=ADDR(PARMS);
PARMSPTR=ADDR(PARMADDR);
EXEC CICS SPOOLOPEN OUTPUT NODE('*') USERID('*')
     TOKEN(TOKEN) OUTDESCR(PARMSPTR) RESP(RESP)
     RESP2(RESP2);
EXEC CICS SPOOLWRITE FROM(OUTPRT) FLENGTH(OUTLEN)
     RESP(RESP) RESP2(RESP2) TOKEN(TOKEN);
EXEC CICS SPOOLCLOSE TOKEN(TOKEN) RESP(RESP)
     RESP2(RESP2);
```

C

```
#define PARMS struct _parms
PARMS
{
  int   parms_length;
  char  parms_info[200];
  PARMS * pArea;
};
PARMS ** parms_ptr;
PARMS  parms_area;
char userid[8]= "*";
char node[8]= "*";
char token[8];
long rcode1, rcode2;
/* These lines will initialize the outdescr area and
set up the addressing */
parms_area.parms_info[0]= '\0';
parms_area.pArea = &parms_area;
parms_ptr = &parms_area.pArea;
/* And here is the command with ansi carriage controls
specified and no class*/
EXEC CICS SPOOLOPEN OUTPUT
       NODE ( node )
     USERID ( userid )
   OUTDESCR ( parms_ptr )
      TOKEN ( token )
        ASA
       RESP ( rcode1 )
      RESP2 ( rcode2 );
```

アセンブラー

```
OUTPRT   DC    CL80'SPOOLOPEN FUNCTIONING'
PARMSPTR EQU   6
RESP     DC    F'0'
RESP2    DC    F'0'
TOKEN    DS    2F
OUTPTR   DC    A(PARMSLEN)
PARMSLEN DC    F'14'
PARMSINF DC    C'WRITER(MYPROG)'
         LA    PARMSPTR,OUTPTR
         EXEC CICS SPOOLOPEN OUTPUT OUTDESCR(PARMSPTR)
               NODE('*') USERID('*') RESP(RESP)
               RESP2(RESP2) TOKEN(TOKEN)
         EXEC CICS SPOOLWRITE FROM(OUTPRT)
               TOKEN(TOKEN) RESP(RESP) RESP2(RESP2)
         EXEC CICS SPOOLCLOSE TOKEN(TOKEN) RESP(RESP)
               RESP2(RESP2)
```

第 15 部 基本マッピング・サポート (BMS)

基本マッピング・サポートを使用して端末装置を制御します。

© Copyright IBM Corp. 1989, 2009

第 60 章 基本マッピング・サポート

基本マッピング・サポート (BMS) は、CICS プログラムと端末装置との間のアプリケーション・プログラミング・インターフェースです。BMS はこの目的のための 2 セットのコマンドのうちの 1 つです。もう 1 つの端末制御については、 587 ページの『第 8 部 端末管理』に説明してあります。

BMS には、多くのアプリケーションに共通して便利な点がいくつかあります。第一に、BMS はアプリケーション・プログラムから装置依存の要素を取り除きます。BMS は**装置独立**の出力コマンドを解釈し、特定の端末用の**装置依存**のデータ・ストリームを生成します。また、装置依存の着信データを装置独立の形式に変換します。これらの機能により、複雑な装置のデータ・ストリームについて理解する必要がなくなります。またこれらの機能により、同一プログラムをいろいろな装置で使用することができます。BMS は、装置情報をアプリケーション・プログラムからではなく、端末定義から判別するためです。

第二に、BMS はアプリケーション論理から形式の設計および準備を分離して、相互に与える影響を少なくします。この 2 つの機能により、新規プログラムの作成および既存コードの保守がより簡単になります。

この章では、以下のことについて説明します。

- 『BMS サポート・レベル』
- 777 ページの『BMS の出力例』

関連タスク

801 ページの『第 62 章 BMS マップ出力の送信』

関連情報

781 ページの『第 61 章 マップの作成』
BMS では、マップ定義のために 3 つのアセンブラー言語マクロ命令 (**マクロ**) が提供されています。ここでは、現在でも幅広く使用されているこのマップ定義の方法について説明します。

807 ページの『第 63 章 SEND MAP コマンドの使用』

817 ページの『第 64 章 マップ・データの受信』
形式設定画面は出力に対してと同様、入力に対しても重要です。データ入力アプリケーションがそのいい例ですが、他のほとんどのアプリケーションでも形式設定入力を、少なくとも部分的に使用します。入力では、BMS は出力の場合と大体逆のことを行います。つまり装置制御文字をデータ・ストリームから除去し、入力フィールドをデータ構造に移動するため、それらを名前で識別できます。

BMS サポート・レベル

BMS のサポートには、最小、標準、完全の 3 つのレベルがあります。ほとんどのインストール・システムでは完全 BMS を使用します。その場合は、BMS のすべての機能を使用することができます。ご使用のシステムが最小または標準の BMS を使用している場合、使用中のシステムに必要なレベルを超える機能に注意してくだ

© Copyright IBM Corp. 1989, 2009

さい。ここではこれらについて要約し、また最小 BMS にない機能の説明がある個所では、これらについての注意があります。

最小機能 BMS

最小 BMS は、3270 端末用のすべての基本機能をサポートします。これには、ここに示す例、および簡単なマップ出力とマップ入力についての解説で述べられるすべてが含まれます。

注: 最小 BMS は、標準 BMS または完全 BMS よりパス長が大幅に短くなります。これはより大きなバージョンに組み込まれ、提供される機能を超える機能を必要としないコマンド上の一種の「高速パス」として呼び出されます。具体的には、プリンシパル装置が外部形式設定を定義に組み込んでいない 3270 ディスプレイかプリンターであるときに、
ACCUM、PAGING、SET、OUTPARTN、ACTPARTN、LDC、MSR、または REQID の各オプションを伴わない SEND MAP コマンドおよび SEND CONTROL コマンド、または RECEIVE MAP コマンドのために使用されます。CICS トレース・テーブルを調べることで、特定の BMS 要求が高速パスを使用したかどうかを知ることができます。高速パスが使用されたときは、トレース・テーブルに BMS 項目および終了コードのための重複項目が含まれます。

標準機能 BMS

標準 BMS では以下のものが追加されます。

- 3270 以外の端末のサポート
- テキスト出力コマンド
- 特殊なハードウェア機能のサポート (分割、論理装置コード、磁気スロット読み取り装置、外部形式設定など)
- SEND コマンド上の追加オプションの NLEOM および FMHPARM

標準 BMS は以下の端末をサポートします。

- 順次端末 (カード読み取り装置、ライン・プリンター、テープまたはディスクで構成)
- TCAM 端末 (589 ページの『第 33 章 端末アクセス方式のサポート』 を参照)。

  注: CICS Transaction Server for z/OS バージョン 4 リリース 1 では、ローカル TCAM 端末はサポートされません。サポートされている TCAM 端末は、TCAM の DCB インターフェース (ACB ではない) によって CICS TS 3.1 以前の端末専有領域に接続されているリモート端末のみです。
- TWX 33/35 型
- 1050
- 2740-1 (受信バッファーなし)、2740-2、2741
- 2770
- 2780
- 2980 1 型、2 型、および 4 型
- 3270
- 3600 (3601) LU

- 3650 (3653 および 3270 ホスト会話型 LU)
- 3650 インタープリター LU
- 3767/3770 対話式 LU
- 3770 バッチ LU
- 3780
- LU タイプ 4

全機能 BMS

完全 BMS は以下のために必要です。

- ユーザー独自の端末に直接的に送信されない BMS 出力の送信 (SET オプション、PAGING オプション、および BMS ルーティング)
- 複数の BMS SEND コマンドを使用して、累積して作成するメッセージ (ACCUM オプションおよび PAGING オプション)

CICS プラットフォームには、BMS の機能を部分的にしかサポートしないものがあります。BMS は CICS Transaction Server for z/OS および CICS Transaction Server for VSE で全面的にサポートされています。しかし、同一レベルであってもプラットフォームによって差があります。この差は通常、ランタイム環境の違いによるものです。これらの機能ごとの詳細は、「*CICS Family: API Structure*」に記載されています。アプリケーションが最終的に他のプラットフォームに移動する場合、またはアプリケーションのエンド・ユーザー・インターフェース部分が他のプラットフォームに配布されるようなことがある場合は、この解説書を参照してください。

BMS の出力例

形式設定画面を作成するために、BMS はプログラムからデータ項目のリストを取り出し、それらを事前定義された形式に従って画面 (または印刷ページ) に表示します。プログラムが提供する変数データと、形式内の定数データ (表題、変数フィールドのラベル、それらのフィールドのデフォルト値) とを組み合わせます。書き込みを行っている端末のためのデータ・ストリームが作成され、この組み合わせデータを指定された画面の位置に、正しい属性 (カラー、強調表示など) で表示します。データ・ストリームについて理解する必要はありません。また、必要な CICS コマンドを書くための形式について、多くの知識は必要はありません。

注: わかりやすさのため、このセクションでは主に表示画面を取り上げて説明しますが、その説明の大部分はプリンターにも該当します。 741 ページの『第 57 章 印刷の CICS サポート』では、表示画面とプリンターの違いについて説明し、印刷に適用される追加の考慮事項についても説明します。さらに、BMS は 3270 の機能をサポートするように設計されているため、ここに示す例および説明は標準の 3270 端末を想定しています。その他の端末については 794 ページの『3270 以外の端末のサポート』に説明してあります。

マップと呼ばれる形式を、マップを使用するプログラムからは分離して定義します。マップにより、フィールドの再配置、フィールド属性の変更、およびプログラムの修正を伴わない定数テキストの変更ができます。変数データを追加または除去する場合は、影響を受けるフィールドを使用するプログラムの変更が当然必要になります。

作動の仕方の基本は、実際より簡単な例で説明してあります。実際には、必要な条件は常により複雑になりますが、ここでは混乱するような細部の説明を避け、本質的なことについて述べます。より実際的で完全な BMS の例は、CICS サンプル・アプリケーションにあります。これらのプログラムは CICS 配布テープにソース形式で組み込まれています。詳しくは、「*Sample Applications Guide*」を参照してください。

この例では、百貨店で使用されるトランザクションのコードを書く必要があると想定します。この百貨店では請求金額売買が完了する前に顧客の取引残高をチェックします。このトランザクションは、顧客の取引がオープンであるか、および現在の購入が受け入れ可能であるかを、その取引の状態に基づいてチェックするのみであるため、「クイック・チェック」と呼ばれます。このトランザクションの*出力* 部分用のプログラムは、顧客番号を入力として取得し、応答で、図 141 に示される次のような画面を作成します。

```
QCK                     Quick Customer Account Check
Account:    0000005
Name:       Thompson        Chris
Max charge: $500.00
```

図 141. 通常の「クイック・チェック」出力画面

プログラムは、入力された取引番号を使用して会計ファイルからその顧客の記録を取り出します。この記録の情報から、プログラムは取引番号と顧客名をマップに表示し、信用限度が許す最大請求金額、未解決の取引残高、および最後の請求期間の後に追加された購入を計算します。合計がマイナスになった場合は、値ゼロを表示し、説明のメッセージを追加します。また、請求金額カードが紛失、盗難、または取り消しと記載されている場合は、図 142 に示すメッセージで事務担当者に警告する必要があります。

```
QCK                     Quick Customer Account Check
Account:    0000005
Name:       Thompson        Chris
Max charge:    $0.00
STOLEN CARD - SECURITY NOTIFIED
```

図 142. 警告メッセージが出ている「クイック・チェック」出力画面

このメッセージは、事務担当者の注意を引くため強調表示されます。

最初にしなければならないことは、画面の定義です。この特定のマップを定義する方法については、 781 ページの『第 61 章 マップの作成』で説明します。しかしここでは、このプロセスの出力の 1 つが 779 ページの図 143 にあるようなデータ構造であると想定します(ここに示す例をコーディングするのに COBOL を使用するため、ここではこの構造の COBOL コーディング・バージョンを示します。しかし、BMS は構造を CICS がサポートするすべての言語で生成します)。マップ作成プロセスは、このソース・コードをライブラリーに格納し、このライブラリーからプログラムにコピーできます。

```
01  QCKMAPO.
    02  FILLER PIC X(12).
    02  FILLER PICTURE X(2).
    02  ACCTNOA    PICTURE X.
    02  ACCTNOO  PIC X(7).
    02  FILLER PICTURE X(2).
    02  SURNAMEA    PICTURE X.
    02  SURNAMEO  PIC X(15).
    02  FILLER PICTURE X(2).
    02  FNAMEA    PICTURE X.
    02  FNAMEO  PIC X(10).
    02  FILLER PICTURE X(2).
    02  CHGA    PICTURE X.
    02  CHGO  PIC $,$$0.00
    02  FILLER PICTURE X(2).
    02  MSGA    PICTURE X.
    02  MSGO  PIC X(30).
```

図 143. 「クイック・チェック」のシンボリック・マップ

この構造でのデータ名はマップ定義から取ります。フィールドに名前を割り当てますが、プログラムがそれを変更する必要があることも考えられます。ここに示す例では、このカテゴリーには取引番号、名字、名前、最大請求金額、および説明のメッセージを表示するフィールドが含まれます。ここには、「Quick Customer Account Check (クイック顧客取引チェック)」および「Account (取引)」などの、変更されないフィールド・ラベルまたは画面表題はいずれも含まれません。

画面で指定する各フィールドは、データ構造内でいくつかのフィールドを生成します。これらのフィールドは、マップに割り当てられた名前に追加された 1 文字の接尾部によって区別されます。ここには 2 つの例があります。「A」という接尾部はフィールド属性バイトを示し、「O」という接尾部は出力データを示します。カラーおよび強調表示などの特殊な装置機構を使用するマップを作成する場合、またはこのマップを出力だけでなく入力にも使用する場合には、これらの数はもっと多くなります。これらの他のフィールドについては、804 ページの『表示特性の設定』および 817 ページの『第 64 章 マップ・データの受信』で説明しています。

この特定の例題で重要となるフィールドは、「O」の接尾部がついたフィールドです。これは、画面に表示したいデータを入力するフィールドです。データの表示方法を変更する場合は、「A」サブフィールドを使用します。ここに示す例では、顧客が疑わしいカードを使用している場合は、MSGA を使用してメッセージを強調表示します。

ここに、この例に必要なコードの概要を示します。マップ作成によって生成されたデータ構造 (図 143) にコピーする必要があり、3 行目の COPY QCKSET ステートメントがこれを行います。(通常は、取引記録形式にもコピー・ステートメントを使用します。内容が見られるように、ここでは部分的に拡大して示します。)

```
   WORKING-STORAGE SECTION.
C COPY IN SYMBOLIC MAP STRUCTURE.
  01  COPY QCKSET.
  01  ACCTFILE-RECORD.
      02  ACCTFILE-ACCTNO                 PIC S9(7).
      02  ACCTFILE-SURNAME                PIC X(15).
      02  ACCTFILE-FNAME                  PIC X(10).
      02  ACCTFILE-CREDIT-LIM             PIC S9(7) COMP-3.
      02  ACCTFILE-UNPAID-BAL             PIC S9(7) COMP-3.
      02  ACCTFILE-CUR-CHGS               PIC S9(7) COMP-3.
      02  ACCTFILE-WARNCODE               PIC X.

   PROCEDURE DIVISION.

      EXEC CICS READ FILE (ACCT) INTO (ACCTFILE-RECORD) RIDFLD (CKNO)
              ... END-EXEC.
      MOVE ACCTFILE-ACCTNO TO ACCTNOO.
      MOVE ACCTFILE-SURNAME TO SURNAMEO.
      MOVE ACCTFILE-FNAME TO FNAMEO.
      COMPUTE CHGO = ACCTFILE-CREDIT-LIM - ACCTFILE-UNPAID-BAL
              - ACCTFILE-CUR-CHGS.
      IF CHGO < ZERO, MOVE ZERO TO CHGO
          MOVE 'OVER CHARGE LIMIT' TO MSGO.
      IF ACCTFILE-WARNCODE = 'S', MOVE DFHBMBRY TO MSGA
          MOVE 'STOLEN CARD - SECURITY NOTIFIED' TO MSGO
          EXEC CICS LINK PROGRAM('NTFYCOPS') END-EXEC.
      EXEC CICS SEND MAP ('QCKMAP') MAPSET ('QCKSET') END-EXEC.
      EXEC CICS RETURN END-EXEC.
```

図 144. BMS の出力例

第 61 章 マップの作成

BMS では、マップ定義のために 3 つのアセンブラー言語マクロ命令 (**マクロ**) が提供されています。ここでは、現在でも幅広く使用されているこのマップ定義の方法について説明します。

ただし、他にもマップ作成のための製品があり、これらにおいてはディスプレイ端末の機能が利用され、マップ作成プロセスがより簡単になっています。これらの製品では BMS マクロと同一の出力を生成しますが、一般にプログラマーの労力が少なくてすみます。

これらの製品の 1 つが 表示画面定義機能 II (SDF II) です。SDF II では、画面を表示ステーションから直接作成でき、作業しながら表示や使用可能度をテストできます。SDF II について詳しくは、「*Screen Definition Facility II General Introduction Part 1*」および「*Screen Definition Facility II General Introduction Part 2*」を参照してください。

BMS マップ定義に使用されるアセンブラー・マクロは次の 3 つです。

DFHMDF
画面上またはページ上で個別のフィールドを定義します。

DFHMDI
単一のマップをフィールドの集合体として定義します。

DFHMSD
一つ一つのマップをマップ・セットにグループ化します。

このプロセスの解説は、個別フィールドの定義の仕方についての説明から始まります。次に、フィールドから入手して完全なマップを作成する方法や単一のマップからマップ・セット (アセンブル単位) にグループ化する方法について説明します。BMS は、基本的に 3270 型端末用に設計されていますが、ほとんどすべてのタイプをサポートできます。 3270 型端末について詳しくは、 623 ページの『第 44 章 3270 ファミリーの端末』を参照してください。

関連概念

775 ページの『第 60 章 基本マッピング・サポート』

マップ・フィールドの定義: DFHMDF

このタスクについて

マクロのコーディングを始める前に、画面のレイアウトを設計してください。それが終了した後で、各フィールドを画面 (ページ) 上に、DFHMDF マクロを使用して定義します。ここでは以下のことを示します。

- 画面上でのフィールドの位置
- フィールドの長さ
- デフォルトの内容 (プログラムに常に内容を提供する場合以外)

© Copyright IBM Corp. 1989, 2009

- **フィールド**表示属性。オペレーターがフィールドに入力できるかどうか、何を入力できるか、カーソルがフィールドで停止するか、また文字の輝度、および修正データ・タグの初期状態を管理します。
- カラー、下線、強調表示などの**拡張**表示属性 (使用する端末による)
- プログラムでフィールドを参照する名前 (フィールドの内容または属性を修正する場合)

アプリケーションが参照するフィールドは、割り振られたフィールド名でなければなりません。フィールド名を作成するために使用できるフィールド名の長さおよび文字は、以下の規則に準拠していなければなりません。(これらの規則は並行してサポートされるコンパイラーとアセンブラーに適用されることに注意してください。)

使用する文字は、アセンブラー通常記号の名前に使用できるものでなければなりません。この文字セットは、A〜Z の英字 (大文字または小文字)、$、#、@、0〜9 の数字、およびアンダースコアー (_) 文字から成ります。

この規則には、1 つだけ例外があります。以下の場合に限り、ハイフン (-) 文字を使用することができます。

- マップ・セットが COBOL で書かれたアプリケーション・プログラムだけで使用される場合。
- マップ・セットが高水準アセンブラーを使用して生成されている場合。

フィールド名の先頭の文字は英字でなければなりませんが、その他の文字は、上記文字セットの任意の文字にすることができます。

さらに、フィールド名に使用する文字は、マップを使用するアプリケーションのプログラミング言語によってサポートされる文字セットに準拠している必要があります。例えば、アプリケーション言語が COBOL である場合は、@ の文字およびアンダースコアーのいずれも使用できません。これら文字セットの詳細については、適切な言語解説書を参照してください。

DFHMDF マクロを使用すれば、フィールド名の長さを 1 文字から 最大 30 文字にすることができます。DFHMDF は、定義された名前にいくつかの追加文字のうち 1 つを加えることによって、追加の変数名を導き出し、シンボリック記述マップを生成します。したがって、これらの導き出された名前の長さは、最大 31 文字になります。アセンブラー PL/I および C 言語は、すべて最低 31 文字の変数名をサポートします。しかし、COBOL 言語で許可されるのは 30 文字までです。これは、COBOL アプリケーションの場合、マップで使用するフィールド名が 29 文字を超えてはいけないことを意味します。例えば、次のフィールド定義は、COBOL 以外のすべての言語で有効です。

```
 ThisIsAnExtremelyLongFieldName DFHMDF LENGTH=10,POS=(2,1)
```

また、下記のフィールド定義は、COBOL でのみ有効です。

```
Must-Not-Exceed-29-Characters DFHMDF LENGTH=10,POS=(2,1) "
```

フィールド定義のすべてのオプションについてここで説明しているわけではありません。これ以外のオプションについては、「*CICS* アプリケーション・プログラミング・リファレンス」で説明しています。

図 145 は、 778 ページの図 142 で見たマップのフィールド定義を示しています。

```
        DFHMDF POS=(1,1),LENGTH=3,ATTRB=(ASKIP,BRT),INITIAL='QCK'
        DFHMDF POS=(1,26),LENGTH=28,ATTRB=(ASKIP,NORM),                X
              INITIAL='Quick Customer Account Check'
        DFHMDF POS=(3,1),LENGTH=8,ATTRB=(ASKIP,NORM),INITIAL='Account:'
ACCTNO  DFHMDF POS=(3,13),LENGTH=7,ATTRB=(ASKIP,NORM)
        DFHMDF POS=(4,1),LENGTH=5,ATTRB=(ASKIP,NORM),INITIAL='Name:'
SURNAME DFHMDF POS=(4,13),LENGTH=15,ATTRB=(ASKIP,NORM)
FNAME   DFHMDF POS=(4,30),LENGTH=10,ATTRB=(ASKIP,NORM)
        DFHMDF POS=(5,1),LENGTH=11,ATTRB=(ASKIP,NORM),INITIAL='Max charge:'
CHG     DFHMDF POS=(5,13),ATTRB=(ASKIP,NORM),PICOUT='$,$$0.00'
MSG     DFHMDF LENGTH=20,POS=(7,1),ATTRB=(ASKIP,NORM)
```

図 145. BMS マップ定義

1. **POS** (位置) パラメーターはフィールドの行および桁の位置を示します。これはマップの左上隅の (1,1) を起点とします。これは必ずなくてはなりません。各フィールドはフィールド属性バイトで始まることを覚えておいてください。POS はこのバイトの位置を定義します。フィールドの内容がすぐあとの右側に続きます。
2. **LENGTH** オプションは、フィールドの長さが何文字かを示します。この長さには属性バイトは含まれない ため、各フィールドは LENGTH 値より 1 桁多くなります。例えばこのマップの最初のフィールドの場合、属性バイトは 1 行目 1 桁目にあり、表示データは 2 ～ 4 桁目にあります。フィールドは 256 文字までの長さにすることができ、ある行から別の行に折り返すことができます (マップが画面より小さい場合は、フィールドの折り返しに注意してください。詳細は 812 ページの『マップの外側』を参照してください)。
3. **ATTRB** (属性) オプションは、フィールドの**フィールド属性**を設定します。これについては 626 ページの『3270 フィールド属性』で説明しています。これは必須ではありません。省略した場合、BMS はデフォルト値 (ASKIP、NORM) (自動スキップ保護、通常輝度、変更データ・タグ・オフ) を使用します。拡張属性にはそれぞれ他にもオプションがありますが、このマップでは使用されていません。これらについては 804 ページの『表示特性の設定』に説明してあります。
4. フィールドの **INITIAL** 値も必須ではありません。これは、「QCK」 などの定数値を持つラベル・フィールドおよび表題フィールドに使用したり、フィールドにデフォルト値を割り当てる場合に使用したりできるので、プログラムで常に値を提供する必要はありません。
5. CHG フィールドの定義上にある **PICOUT** オプションは、どの種類の PICTURE (ピクチャー) 文節をフィールドに生成するかを BMS に指示します。これにより、データをマップに移動するときに COBOL または PL/I の編集機能を直接使用することができます。 PICOUT を省略し、さらに数値 (NUM) 属性も省略すると、BMS は文字データとみなします。 779 ページの図 143 には、CHG およびその他のフィールドに対する PICOUT オプションの影響、およびそれが省略された場合の影響が示されています。BMS はピクチャーから長さを判断するため、PICOUT を使用する場合は LENGTH オプションを省略できます。
6. **GRPNAME** オプションおよび **OCCURS** オプションは、より複雑な問題に対処するためのものなので、ここに挙げた簡単な例には出てきません。GRPNAME を使用すると、処理のためにプログラム内でマップ・フィールドを再分割できます。OCCURS は、マップ・フィールドを隣接して定義し、複数のもののように

できるため、マップ・フィールドをプログラムで 1 つの配列として扱うことができます。これらのオプションについては、マップについてさらに説明した後に、 791 ページの『複合フィールドの使用』で説明されます。

マップの定義: DFHMDI

このタスクについて

マップ上のすべてのフィールドの定義を完了したら、DFHMDI マクロを先に使用して、それらのフィールドが 1 つのマップを形成することを BMS に指示します。このマクロは以下のものを BMS に示します。

- マップの名前
- 行と桁によるマップのサイズ
- 画面上で表示される場所 (1 つの画面に複数のマップを表示できます)
- 3270 拡張表示属性を使用するかどうか。使用する場合、どの属性表示か。
- DFHMDF マクロで特定の値を割り当てなかったフィールドに対する、これらの拡張属性のデフォルト
- マップの送信に関連した装置制御 (アラームを鳴らすかどうか、キーボードをアンロックするかなど)
- マップがサポートする装置のタイプ (タイプの異なる装置のためにマップの複数のバージョンを作成する場合) (797 ページの『装置依存マップ』を参照してください)

マップ名およびマップのサイズは DFHMDI マクロ上の重要な情報ですが、文書化する目的のためには、他のオプションをデフォルトにするのではなく、それらを明示的に指定する必要があります。ここに示す例での DFHMDI マクロは次のようになります。

```
QCKMAP   DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=ALARM
```

ここではマップを QCKMAP と命名しました。これは、SEND MAP コマンドで使用する ID です。長さは 24 行で幅は 80 桁あり、ディスプレイの 1 行目 1 桁目から開始します。また、マップが表示されるときにアラームが鳴るように指示しました。

マップ・セットの定義: DFHMSD

このタスクについて

マップの作成にはさらに、マップ・セットを定義する DFHMSD というマクロが必要です。マップはマップ・セットと呼ばれるグループでアセンブルされます。通常は、単一トランザクションまたは複数の関連するトランザクションで使用するすべてのマップをグループ化します (マップをグループ化する理由については、 789 ページの『マップ・セットへのマップのグループ化』でさらに説明します)。また、マップ・セットは複数のマップを含む必要はなく、ここの簡単な例では、マップ・セットは「クイック・チェック」マップのみで構成されています。

1 つの DFHMSD マクロはマップ・セットにあるすべてのマップ定義の前に置かれ、以下のものを示します。

- マップ・セットの名前
- マップを出力、入力、またはその両方に使用するかどうか
- DFHMDI マクロで個別マップについて指定しなかったマップ特性のデフォルト
- フィールドまたはマップ定義のいずれかで指定しなかった拡張属性のデフォルト
- 現行アセンブルで物理マップまたはシンボリック・マップを作成するかどうか
  (788 ページの『物理的およびシンボリックのマップ・セット』を参照)
- マップを使用するプログラムのプログラミング言語
- マップの作成に使用するストレージについての情報

次に、先の例の最初に必要な DFHMSD マクロを示します。

```
QCKSET   DFHMSD TYPE=MAP,STORAGE=AUTO,MODE=OUT,LANG=COBOL,TIOAPFX=YES
```

このマップ・セット定義は、この中のマップが出力にのみ使用されること、およびこれらのマップを使用するプログラムが COBOL で書かれていることを BMS に示します。QCKSET という名前がマップ・セットに割り当てられます。TIOAPFX=YES と設定すると、12 バイトの「接頭部」フィールドが各シンボリック・マップの先頭に組み込まれます (この効果は 779 ページの図 143 の 2 行目に示されています)。コマンド言語プログラムでは常にこの充てん文字が必要です。デフォルトでは省略されることがあるため、明示的に指定してください。MAP および STORAGE については、 801 ページの『第 62 章 BMS マップ出力の送信』に説明があります。

マップ定義の最後には、マップ・セット内の最後のマップが終了したことをアセンブラーに示すために別の DFHMSD マクロが必要になります。

```
         DFHMSD TYPE=FINAL
```

BMS マクロの作成

BMS マクロはアセンブラー言語ステートメントなので、アセンブラー構文規則に従わなくてはなりません。ここでは、これらの規則の詳細については説明しません。詳しくは、「*Assembler H Version 2 Application Programming Language Reference*」を参照してください。代わりに、実際の規則より制限されていますが、有用な一連の規則について述べます。

1. 1 桁目から名前を開始します。マップおよびマップ・セット名の長さは 7 文字までです。フィールド名の最大長 (DFHMDF マクロ) はプログラム言語によって変わります。BMS は、ユーザーのフィールド名に 1 文字の接尾部を追加してラベルを作成します。これらのラベルはプログラムにコピーされるため、ターゲット言語が許可する長さより長くてはいけません。したがって、マップ・フィールド名の限度は、COBOL の場合は 29 文字、PI/I および アセンブラー H の場合は 30 文字、アセンブラー F の場合は 7 文字です。C および C++ では、マップがプログラムに内部データ・オブジェクトとしてコピーされる場合は 30 文字、外部データ・オブジェクトの場合は 6 文字です。(マップのコピーについて詳しくは、 801 ページの『マップ用ストレージの獲得および定義』を参照してください)。
2. 10 桁目からマクロ ID を開始するか、または名前が 8 桁目の位置を超える場合はマクロ ID と名前の間にブランクを 1 つ置きます。ID は、フィールド定

義では常に DFHMDF、マップ定義では DFHMDI、マップ・セットの開始と終了を行うマップ・セット・マクロでは DFHMSD です。

3. フィールドの記述の残りの部分はキーワード (例えば位置パラメーターの場合は POS) で構成され、その後に値が続きます。キーワードには値がないこともありますが、値がある場合には必ずキーワードと値が等号 (=) で分離されています。
4. マクロ ID の後にブランクを 1 つ置いてから、キーワードを開始してください。キーワードは任意の順序で表示できます。
5. キーワードはブランクではなくコンマで分離しますが、最後のキーワードの後にはコンマを入れないでください。
6. キーワードは 71 桁目まで指定できます。さらにスペースが必要な場合は、途中で改行しないように完全にキーワードを書いてその後にコンマを付けてから、次の行の 16 桁目で再開してください。
7. 初期値 (INITIAL、XINIT、および GINIT キーワード) は、改行で開始してもフィットしないため、規則にはあてはまりません。2 バイト文字が含まれるとき以外は、これらを初期値自体の先頭文字の後の任意の場所で分割することができます。この方法で分割するときは、71 桁までのすべての桁を使用して、次の行の 16 桁目から継続してください。2 バイト文字セット (DBCS) データは通常の 1 バイト (SBCS) データより表し方が複雑です。 DBCS の初期値を使用する場合は、ステップ 12を参照してください。
8. 初期値を単一引用符マークで囲みます。テキストの*中に* 単一引用符が必要な場合は、2 つの連続する単一引用符を使用してください (アセンブラーが余分な 1 つを除去します)。アンパーサンドもアセンブラーに対して特殊な意味を持つため、同じ技法を用います。つまり、アンパーサンドを 1 つ置きたい所で 2 つ使用すると、アセンブラーが余分な 1 つを除去します。
9. マクロに 2 行以上使用する場合は、文字を 1 つ (ブランク以外何でも) *最後の行以外の* すべての行の 72 桁目に入れます。
10. マップに注釈を入れたい場合は、単一マクロを構成する行の間ではなく、マクロの間で注釈行を使用します。注釈行には、1 桁目にアスタリスクを、72 桁目にブランクを入れます。2 桁目から 71 桁目の任意の個所に注釈を書くことができます。
11. **INITIAL** パラメーターの値および注釈の中以外では、大文字のみを使用してください。
12. **DBCS を含む初期値の場合**。*完全に* DBCS のみの初期データの場合は、データに GINIT キーワードを使用し、PS=8 キーワードも指定してください。データが DBCS 文字と SBCS 文字の両方を含む場合、すなわち両方の**混合**である場合は、INITIAL を使用し、SOSI=YES と指定してください(3 つめの方法である XINIT についても、保守しているコードで検出されることがあるため説明する必要があります。ただし、可能であれば GINIT および INITIAL を使用してください。XINIT は使用が難しく、データが完全には妥当性を検査されないためです。XINIT は DBCS のみ、または混合している DBCS のいずれでも使用できます。PS=8 を指定する XINIT は GINIT の規則に従い、SOSI=YES を指定する XINIT は INITIAL の規則に従います (少なくともほとんどの場合そうなります)。主な違いは、XINIT ではデータを 16 進数で表すことですが、GINIT および INITIAL では通常の文字を使用します)。

以下に DBCS 初期値の書き方を示します。

- 通常の INITIAL パラメーターの場合と同様に、データを単一引用符で囲みます。
- 定数の各 DBCS 文字に対して 2 つの通常文字を使用し (XINIT の 16 進数字の 2 つのペア)、各 SBCS 文字に対して 1 つの通常文字を使用します (XINIT の 1 つのペア)。
- 各 DBCS 文字ストリングを囲み、シフト・アウト (SO) 文字を直前に、シフト・イン (SI) 文字を直後に入れます。SO は 16 進数の X'0E' であり、ほとんどのキーボードでは「<」で表され、SI は X'0F' (「>」) です。(PS=8 を指定する XINIT は例外です。SO/SI の囲みは暗黙的に扱われるのでこれらは入力しません)。例えば以下のものはすべて同一の初期値を定義し、完全に DBCS です (ここでは LENGTH 値は無視してください。これについてはすぐに説明します)。

```
GINIT='<D1D2D3D4D5>',PS=8,LENGTH=10
INITIAL='<D1D2D3D4D5>',SOSI=YES,LENGTH=12
XINIT='C4F1C4F2C4F3C4F4C4F5',PS=8,LENGTH=10
XINIT='0EC4F1C4F2C4F3C4F4C4F50F',SOSI=YES,LENGTH=12
```

- SBCS および DBCS の各シーケンスは、互いにいかなる順序でもよく INITIAL (および SOSI=YES を指定する XINIT) のいかなる組み合わせにも従うことができます。前の例で DBCS ストリングの前に「ABC」を追加し、ストリングの後に「def」を追加する場合は次のようになります。

```
INITIAL='ABC<D1D2D3D4D5>def',SOSI=YES,LENGTH=18
XINIT='C1C2C30EC4F1C4F2C4F3C4F4C4F50F848586',SOSI=YES,LENGTH=18
```

- 初期値の長さを計算するためには、通常の文字で表すか 16 進数のペアで表すかにかかわらず、各 DBCS 文字では 2 つ、各 SBCS 文字では 1 つをカウントします。GINIT (および PS=8 を指定する XINIT) では SO 文字および SI 文字をカウントしませんが、INITIAL (および SOSI=YES を指定する XINIT) では各 SO および各 SI に対し 1 つを追加します (上の例の、同一定数に対する異なる LENGTH 値に注意してください)。いずれの場合においても、LENGTH 値が 256 を超えてはいけません。
- GINIT および INITIAL では、定数が 1 行におさまらない場合、「拡張」継続規則を使用します。これは今までに述べたものとはやや異なります。拡張継続を使用すると、初期値内のいずれの完全文字 (SBCS 文字、DBCS ペア、または DBCS ストリングを終了する SI) の後でも停止できます。DBCS ストリングのまん中の場合は、SI を 1 つ追加します (同一行に SO と SI の対がなければなりません)。次に、その行の 72 桁目に継続文字を入れます。いずれの文字でも、それがその行の最後の意味のある文字と異なるかぎり使用できます。

  DBCS ストリング内で停止した場合は、次の行の 16 桁目に SO 文字を 1 つ入れ、17 桁目から再開します。それ以外は 16 桁目から再開します。次のようになります。

```
GXMPL1   DFHMDF POS=(02,21),LENGTH=20,PS=8,GINIT='<D1D2D3D4D5D6>******
               <D7D8D9D0>'
IXMPL1   DFHMDF POS=(02,21),LENGTH=23,PS=8,INITIAL='abc<D1D2D3D4>ABC**
               DEFGHIJ'
```

  XINIT では拡張継続は使用できません。ステップ 7 (786 ページ) で説明する規則に従ってください。

- LENGTH の長さ指定が、提供する初期値の長さを超えている場合、GINIT (または PS=8 を指定する XINIT) を使用していれば、その値は右方を DBCS ブランクで、LENGTH 値まで埋められます。INITIAL を使用した場合、定数の最後の部分が SBCS だった場合は充てん文字は SBCS ブランク、最後の部分が DBCS だった場合は DBCS ブランクです。SOSI=YES を指定する XINIT を使用する場合は、充てん文字は常に SBCS ブランクです。

マップのアセンブル

このタスクについて

コーディングを開始する前に、マップ・セットのアセンブルとリンク・エディットを行う必要があります。マップ・セットを 2 つの異なる形式で作成するためには、通常 2 回アセンブルしなくてはなりません。DFHMSD マクロの TYPE オプションは、特定のアセンブルで作成する形式をアセンブラーに指示します。

物理的およびシンボリックのマップ・セット

リンク・エディットの前に行われる TYPE=MAP アセンブルは、**物理マップ・セット**と呼ばれるロード・モジュールを生成します。物理マップ・セットにはエンコード形式の形式設定情報が含まれます。 CICS ではこれを実行時に定数フィールドに使用し、プログラムから変数データにどのように組み合わせるかを判別します。

物理マップ・セットは通常、アプリケーション・プログラムと同一のライブラリーに格納され、プログラムが PROGRAM リソース定義を必要とするのと同様に、CICS 内に MAPSET リソース定義を必要とします。

TYPE=DSECT アセンブルの出力は一連のデータ構造で、ひとまとめにして**シンボリック・マップ・セット**と呼ばれ、LANG オプションで指定するソース言語でコーディングされます。各マップの入力用の構造が 1 つあり、**記号入力マップ**といいます。また、各マップの出力用の構造は**記号出力マップ**といいます。

シンボリック・マップ・セットはコンパイル (アセンブル) 時に使用します。このマップ・セットをプログラムにコピーすると、マップにあるフィールドを名前によって参照することができ、また変数データを物理マップ・セットによって指示された形式で渡すことができます。COBOL での記号出力マップの例は、説明済み (779 ページの図 143 を参照) で、例示されたコードで使用しています。シンボリック・マップ・セットは通常、プログラムにコピーされるソース・コード用にご使用のシステムが定義するライブラリーに格納されています。メンバー名は通常マップ・セット名と同じですが、同じである必要はありません。

プログラムをコンパイルまたはアセンブルする前に TYPE=DSECT アセンブルが必要です。物理マップ・セットは実行時まで使用されないため、TYPE=MAP アセンブルおよびリンク・エディットは、テストする準備ができるまで延期することができます。ただし、最終的には両方とも行わなくてはならないため、多くのインストール・システムではこれを自動的に行うためのカタログ式プロシージャーが提供されています。このプロシージャーはマップ・セット用ソース・ファイルをコピーし、TYPE=MAP を用いて 1 回、また TYPE=DSECT を用いてもう 1 回処理します。アセンブラー・プロシージャーの SYSPARM オプションを使用して、特定のアセンブルで TYPE 値を指定変更することもできます。マップのアセンブルに関連した

SYSPARM の説明については「*Assembler H Version 2 Application Programming Language Reference*」を参照し、マップのアセンブルについて詳しくは、 157 ページの『第 9 章 マップ・セットおよび区分セットのインストール』を参照してください。

注:

1. シンボリック・マップ・セットが特定の言語でコーディングされるということは、異なる言語でコーディングされた複数のプログラムで同一のマップは使用できないということではありません。必要な LANG 値それぞれに対して TYPE=DSECT でアセンブルし、注意してそれらの出力を異なるライブラリーに格納するか、または異なる名前で格納するようにします。LANG 値は TYPE=MAP アセンブルには影響せず、こちらは 1 回しか必要ありません。
2. 既存のマップをシンボリック・マップに影響するように修正する場合は、そのマップを使用するすべてのプログラムを*再コンパイル (再アセンブル) しなくてはなりません*。そうすると、コンパイルは新規の物理構造に対応する記号構造を使用します。名前のないマップ・フィールドへの変更はシンボリック・マップに影響しませんが、名前付きフィールドの追加、削除、再配置、および長さの変更は影響します(再配置については DFHMDF マクロを参照してください。DFHMDF マクロを画面上のフィールドの順序と同一にしておく方が効率的ではありますが、画面上のフィールドの順序はシンボリック・マップに影響しません)。したがって、マップ定義の DSATTS オプションを変更してください。このオプションは、プログラムで変更可能な拡張属性を示します。もちろん再コンパイルするのが常に最も安全です。

SDF II による代替

これらのアセンブリーのステップまたはリンク・エディットのステップは、いずれも IBM ライセンス・プログラムの表示画面定義機能 II を使用する場合には必要ないものです。SDF II はシンボリック・マップ・セットと物理マップ・セットの両方を、対話式マップ作成処理の最後のステップで作成します。 SDF II は、MVS (プログラム 5665-366) または VM (5664-307) のいずれでも実行できます。「*Screen Definition Facility II Primer for CICS/BMS Programs*」、「*Screen Definition Facility II General Introduction Part 1*」、および「*Screen Definition Facility II General Introduction Part 2*」を参照してください。詳しくは、「*Screen Definition Facility II General Information*」および「*Screen Definition Facility II Primer for CICS/BMS Programs*」を参照してください。

マップ・セットへのマップのグループ化

マップ・セット内のすべての物理マップは一緒にアセンブルされるため、単一のロード・モジュールを構成します。BMS では、タスクでマップ・セットを最初に使用するときに出される単一のロード要求で、すべてのマップへのアクセスができるようになります。

異なるセットのマップを要求しない限りはこれ以上のロードは必要ありませんが、その場合には BMS は古いマップ・セットを解放して新規のマップ・セットをロードします。その後最初のマップ・セットに戻る場合は、最初のマップ・セットを再びロードします。ロードおよび削除は必ずしも入出力に関連しませんが、マップを

マップ・セットにグループ化するときはパス長に配慮する必要があります。一般に、一緒に使用するマップは同一のマップ・セットに、そうではないマップは異なるマップ・セットにしてください。

1 つのセット内に入るマップ数の限度は 9998 ですが、所定のロード・モジュールのサイズはすべて、適切な大きさを保持するようにしてください。そうすると、めったに使用されないマップと、所定のプロセスで通常使用されるマップとを分離できます。

同様に、マップ・セットのすべてのシンボリック・マップは単一の記号構造にあります。このことは、802 ページの『BASE および STORAGE オプション』の説明にあるとおり、マップの使用中に必要なストレージの量に影響します。プログラム言語によっては高水準の名前に影響することもあり、この点も考慮してマップの分離または結合を考えてください。

アプリケーション・データ構造 (ADS)

BMS マクロによって生成されたシンボリック・マップは、アプリケーション・データ構造 (ADS) としても知られます。

CICS Transaction Server for z/OS バージョン 4 リリース 1 によって作成された物理マップにも、出力ロード・モジュール内に ADS 記述子が組み込まれています。これは、BMS アプリケーション・データ構造 (SEND 要求および RECEIVE MAP 要求のデータ用に、アプリケーション・プログラムが使用する構造) による変換処理を可能にするために提供されるものです。この場合、コンパイル時に、関係のある DSECT またはコピーブックがプログラムに組み込まれている必要はありません。

ADS 記述子には、マップについての一般情報を備えたヘッダー、および ADS 内にある各フィールドのフィールド記述子 (マップ定義マクロ内のそれぞれの名前付きフィールドに対応する) が含まれています。この記述子は、DFHMAPDS 内のオフセット・フィールドからのマップ・セットに配置することができます。

ADS 記述子は、すべてのマップに対して生成されます。DSECT=ADS|ADSL オプションを指定することによって、ADS の長い形式をマップするか、短い形式をマップするかを選択することができます。デフォルトは ADS の短い (通常) 形式です。ADS の長い形式は、すべてのフィールドを 4 バイトの境界に桁揃えし、WebSphere MQ などの他の製品との何らかのインターフェース用として必要とされます。

CICS Transaction Server for z/OS バージョン 4 リリース 1 以前の各 CICS リリースで生成されたマップ・セットには、ADS 記述子が含まれていません。

ADS 記述子の形式は、以下のコピーブックに含まれています。

表 43. ADS 記述子コピーブック

言語	コピーブック
アセンブラー	DFHBRARD
C	DFHBRARH

表 43. ADS 記述子コピーブック (続き)

言語	コピーブック
PL/I	DFHBRARL
COBOL	DFHBRARO

ADS 記述子について詳しくは、「*CICS External Interfaces Guide*」を参照してください。

マップを再組み立てする必要があるのに、ソースにアクセスできない場合は、マップ・セット・ロード・モジュールから BMS マクロ・ソースを再作成するためのユーティリティー・プログラム DFHBMSUP が、CICS Transaction Server for z/OS バージョン 4 リリース 1 に提供されています。

DFHBMSUP の詳細については、「*CICS Operations and Utilities Guide*」のBMS マクロ生成ユーティリティー・プログラム (DFHBMSUP)を参照してください。

複合フィールドの使用

このタスクについて

これまでに示したシンボリック・マップは、各命名済みマップ・フィールドごとの固定された一連のフィールドで構成されていました (779 ページの図 143 の A サブフィールドおよび O サブフィールドなど)。このようなフィールドが最も一般的ですが、BMS では フィールド定義のために 2 つのオプションが提供されており、やや異なる構造を生成して 2 つの一般的なプログラミング状況に対応します。

複合フィールド: GRPNAME オプション

このタスクについて

表示上で、単一フィールド内のサブフィールドを参照しなければならない場合もあります。例えば、次のように画面に表示される日付フィールドがあるとします。

```
03-17-92
```

これは、画面上の 1 つのフィールド (桁 0 の直前に 1 つの属性バイトがある) ですが、プログラムで月、日、および年のコンポーネントを個別に操作できなければなりません。

DFHMDF マクロの GRPNAME オプションを使用し、「グループ・フィールド」でこれを実行できます。グループ・フィールドを作成するために、DFHMDF マクロをそれぞれのコンポーネントのサブフィールドに対してコーディングします。それぞれの定義は GRPNAME オプションの同一のグループ・フィールドを命名します。上記の日付を、10 行目で開始するグループ・フィールドとして定義するには、例えば、次のように書き込みます。

```
MO       DFHMDF POS=(10,1),LENGTH=2,ATTRB=BRT,GRPNAME=DATE
SEP1     DFHMDF POS=(10,3),LENGTH=1,GRPNAME=DATE,INITIAL='-'
DAY      DFHMDF POS=(10,4),LENGTH=2,GRPNAME=DATE
SEP2     DFHMDF POS=(10,6),LENGTH=1,GRPNAME=DATE,INITIAL='-'
YR       DFHMDF POS=(10,7),LENGTH=2,GRPNAME=DATE
```

これらの定義は記号出力マップに次のものを生成します。

```
      02 DATE.
         03 FILLER PICTURE X(2).
         03 MOA    PICTURE X.
         03 MOO  PIC X(2).
         03 SEP1 PIC X(1).
         03 DAO  PIC X(2).
         03 SEP2 PIC X(1).
         03 YRO  PIC X(2).
```

グループ・フィールドを使用するときは、いくつかの規則を考慮しなければなりません。

- 属性バイトは 1 つだけです。これはグループ・フィールド全体の前にきて、フィールド全体に適用されます。これは DFHMDF マクロ上で最初のサブフィールドのために一度だけ指定します。ここでは MO です。
- 属性バイトが 1 つだけなので、カーソルはこのグループ・フィールドが単一フィールドである場合と同様に動作します。ここに示す例では、カーソルは月の最後の位置から日の最初の位置へ、または日から年へというように、ハイフンを超えた移動はしません。これは、グループは、ハードウェアに関する限り実際には単一のフィールドであるためです。このグループは、コンポーネント・サブフィールドへのプログラム・アクセスのためにだけ再分割されています。
- 例に示すとおり、最初のサブフィールドの後のサブフィールドには属性バイトがありませんが、属性バイトがある場合と同様に POS オプションを定義します。つまり、POS はサブフィールドが始まる前の 1 文字を指し、例で見るとおり直前のサブフィールドの最後の文字にオーバーラップすることができます。
- この例ではすべてのコンポーネント・サブフィールドが隣接していますが、そうである必要はありません。サブフィールドの間にはギャップがあることもあります (そのギャップに他のフィールドを定義しない場合)。グループ・フィールドはすべての桁が最初のサブフィールドから最後のサブフィールドまでに及ぶため、コンポーネントの DFHMDF マクロを、サブフィールドが画面に表示される順に置かなくてはなりません。グループは、名前を指定しない最初の DFHMDF マクロで終了します。
- フィールド名は、参照するつもりがない場合でも各サブフィールドに割り当てなくてはなりません (例に示した SEP1 サブフィールドおよび SEP2 サブフィールドを参考にしてください)。
- OCCURS オプション (次のセクションで説明します) は、グループ・フィールドまたはそのコンポーネントのいずれにも使用できません。

反復フィールド: OCCURS オプション

このタスクについて

画面には、プログラムで配列として扱いたい一連の同一フィールドが含まれることがあります。例えば、事務担当者が未使用の電話番号を新規顧客に割り当てるとき使用するための、40 個の番号の表示を作成する必要があるとします (この想定は、

顧客に選択肢を与えるためです)。また、最近使われ始めた電話番号を強調表示し、顧客がその番号の以前の所有者に電話することのないように注意を促す必要があるとします。

電話番号を表示する画面の一部を単一フィールド定義で定義することができます。

```
TELNO    DFHMDF POS=(7,1),LENGTH=9,ATTRB=NORM,OCCURS=40
```

このステートメントにより、隣接するが分離されている 40 個の表示フィールドが生成されます。これは、(7,1) の位置で始まり、必要な行数分 (この例では 5) 進みます。ここでは画面の幅を均等に分割する長さ (属性バイトの追加も含む) を選択したため、番号は垂直の桁に表示され、行の境界で分割されません。指定する属性、および初期値もともに各フィールドに適用されます。

シンボリック・マップでのこれらのフィールドの記述は、COBOL では以下のようになります。

```
       02 TELNOG    OCCURS 40.
          03 FILLER PICTURE X(2).
          03 TELNOA    PICTURE X.
          03 TELNOO  PIC X(9).
```

この構造により、マップをプログラム (またはその他のソース) の配列から以下のように埋めることができます。

```
           PERFORM MOVENO FOR I FROM 1 THROUGH 40.
           ...
       MOVENO.
           MOVE AVAIL-NO (I) TO TELNOO (I).
           IF DAYS-SINCE-USE (I) < 90, MOVE DFHBMBRY to TELNOA (I).
```

(DFHBMBRY は、フィールド輝度を高輝度に設定するための、CICS 提供の定数です。 806 ページの『属性値の定義: DFHBMSCA』で詳しく説明します。)

OCCURS フィールドのラベルは、CICS がサポートする他の言語に対しては少し異なりますが、機能は同じです。

OCCURS オプションによって作成された配列の各エレメントは、単一マップ・フィールドです。一連のフィールド (言い換えれば構造の配列) を反復する必要がある場合には、OCCURS は使用できません。プログラムでこのような配列を使用するためには、すべてのフィールドを OCCURS を使用せずに個別に定義して、必要な物理マップを生成しなくてはなりません。その後で結果として生ずるシンボリック・マップを修正でき、個別のフィールド定義を、反復したい構造をエレメントとして持つ配列に置き換えます。当然ですが、改訂されたシンボリック・マップは元のものとまったく同一のフィールド構造を持つようにしなくてはなりません。代替の方法として SDF II を使用でき、こちらではこのような配列を直接定義できます。

ブロック・データ

BMS にはブロック・データ形式というシンボリック・マップのための代替形式があり、特定の状況においては便利に使用できます。ブロック・データ形式では、記号出力マップは端末に送られる画面またはページのイメージです。これには、各命名済みマップ・フィールドに対して通例のフィールド属性 (A) サブフィールドおよび出力値 (O) サブフィールドがありますが、各マップ・フィールドに対するサブフィ

ールドは、シンボリック・マップ構造での変位が画面の位置に対応する充てん文字フィールドによって分離されます。長さサブフィールドはなく、その結果記号カーソルの位置決めは利用できません。

例えば、 778 ページの図 142 における「クイック・チェック」画面のシンボリック・マップは、ブロック・データ形式 (幅が 80 桁のマップと想定) では下記のようになります。これを、同一のマップ定義からの (779 ページの図 143 における) 通常の「フィールド・データ」形式と比較してください。

```
01  QCKMAPO.
    02  FILLER PIC X(12).           <---TIOAPFX still present
    02  FILLER PICTURE X(192).      <---Spacer
    02  ACCTNOA    PICTURE X.       <---Position (3,13)
    02  ACCTNOO  PIC X(7).
    02  FILLER PICTURE X(72).       <---Spacer
    02  SURNAMEA    PICTURE X.      <---Position (4,13)
    02  SURNAMEO  PIC X(15).
    02  FNAMEA    PICTURE X.        <---Position (4,30),
    02  FNAMEO  PIC X(10).
    02  FILLER PICTURE X(52).       <---Spacer
    02  CHGA    PICTURE X.          <---Position (5,13)
    02  CHGO  PIC $,$$0.00
    02  FILLER PICTURE X(139).      <---Spacer
    02  MSGA    PICTURE X.          <---Position (7,1).
    02  MSGO  PIC X(30).
```

図 146. ブロック・データ形式の「クイック・チェック」用シンボリック・マップ

プログラムにはフィールド属性のみを設定できます。BMS はマップの DSATTS オプションを無視し、拡張属性のためのサブフィールドをブロック・データ形式では生成しません。ブロック・データは入力にも使用できます。入力マップは出力マップと構造が同一です。ただしフィールド形式にあるとおり、フラグ (F) がフィールド属性 (A) サブフィールドの代わりになり、入力 (I) が出力 (O) サブフィールドの代わりになります。

ブロック・データ形式は、アプリケーション・プログラムが、画面に表示したいプリンター・ページ・イメージを作成したか、またはそれへのアクセスを持つ場合に便利です。しかしほとんどの状況では、通常のフィールド・データ形式の方が優れた機能と柔軟性を備えています。

3270 以外の端末のサポート

最小 BMS は 3270 ディスプレイおよびプリンターしかサポートしません。このカテゴリーには 3178、3290、8775、5520、LU タイプ 2 およびタイプ 3 の装置、およびその他の 3270 データ・ストリームを受け入れる端末すべてが含まれます。「*IBM 3270 Information Display System Data Stream Programmer's Reference*」には、すべてリストされています。**標準** BMS は、SCS プリンター (3270 データ・ストリームを使用しない 3270 ファミリーのプリンター)、および 798 ページの表 44 にリストされているすべての端末タイプに対して 3270 のサポートを拡張します。BMS および SCS データ・ストリームについて詳しくは、 748 ページの『非 3270 CICS プリンター』を参照してください。

これらの端末タイプはそれぞれ機能が異なるため、各端末タイプに対してまったく同一の方法で BMS を作動させることはできません。以下のセクションでは、一定の機能のための基本ハードウェアがない装置において BMS を使用するときの制限を概説します。

3270 以外の装置の出力に関する考慮事項

BMS では出力データ・ストリームの装置依存部分と、論理部分が分離されるため、BMS 出力を作成するときに配慮が必要な、3270 装置と 3270 以外の装置との違いはごくわずかです。

3270 と 3270 以外の装置の基本的な違いは、3270 は**フィールド指向**であり、3270 以外のほとんどがそうではないことです。そのため、3270 以外の端末に送信される出力フィールドに関連したフィールド属性も拡張属性も存在しません。BMS は、出力をフィールドごとに正しい場所に置くことができますが、フィールド構造はデータ・ストリームに反映されません。BMS は、一部の端末で一部のフィールド属性をエミュレートすることもできますが (強調表示したフィールドに下線を引くなど)、その場合修正データ・タグがない、フィールド入力に対する保護がない、などといったことがあります。

端末がサポートしない属性を出力に指定すると、BMS はそれらを無視します。機能が欠けていても理解できる出力が得られる場合は、このことを懸念する必要はありません。

入力における差異

フィールド構造がない場合、入力操作により多くの影響があります。これは、BMS の機能の多くが、修正されたフィールドのみを (フィールドごとに) 読み取る機能に依存しているためです。ハードウェアがフィールドごとの入力に、画面上の位置を提供しない場合は、同等の情報を提供しなくてはなりません。

2 つの方法でこれを行うことができます。1 つは、1 文字から 4 文字分の長さのフィールド分離シーケンスを、マップ定義の FLDSEP オプションに定義する方法です。このシーケンスを入力の各フィールドの間に置き、入力フィールドを画面上またはページ上に表示されるのと同じ順序で指定します。画面にあるすべてのフィールドを、いかなるデータを含むものでも最後のものまで指定しなくてはなりません。フィールドに入力がない場合、終了フィールド分離シーケンスのみで構成されていると考えられます。ハードコピー装置では、用紙の移動があるため入力は出力をオーバーレイできません。このような端末をエミュレートするディスプレイでは、同じ手法が一般に使われます。入力フィールドはこの目的のために予約されている領域に順序どおりに入力され、フィールド分離シーケンスによって分離されます。

2 つ目は、入力に制御文字を組み込む方法です。マップから FLDSEP オプションを省略すると、BMS は制御文字を使用して「ページ」上でのデータの位置を計算し、それに従ってマップします。BMS が認識する制御文字は次のとおりです。

NL	改行	X'15'
IRS	交換レコード分離文字	X'1E'
LF	改行	X'25'
FF	用紙送り	X'0C'

HT	水平タブ	X'05'
VT	垂直タブ	X'0B'
CR	復帰	X'0D'
RET	TWX 上のリターン	X'26'
ETB	テキスト・ブロック終了	X'26'
ESC	エスケープ、2780 の場合	X'27'

このようなデータを RECEIVE MAP コマンドを使用して読み取る場合、本来の 3270 入力とはいくつか異なる点があります。

- フラグ・バイト (F サブフィールド) は設定されず、ヌルを含みます。オペレーターがフィールドを消去したのかどうか、またカーソルがフィールドに残っているのかどうかを判別することはできません。
- フィールドが入力に返されるように、出力で修正データ・タグをオンに事前設定することはできません。

3270 以外の端末の特殊オプション

BMS には 3270 以外の装置のために、追加の形式設定オプションがいくつか提供されており、データ・ストリームを短縮する装置機能を利用することができます。以下のことが含まれます。

- 垂直タブおよび水平タブ。装置がサポートする場合は、水平タブおよび垂直タブの順番で出力の位置決めをすることができます。タブ文字はマップ・セット定義の HTAB オプションおよび VTAB オプションによって定義されます。次の水平タブに位置を定めたいときは、データに HTAB 文字を組み込みます。また次の垂直タブに位置を定めるときは、データに VTAB 文字を組み込みます。BMS は、固有のプリンシパル装置に必要なタブ・シーケンスに、これらの文字を変換します。

  BMS 出力でタブを使用する前に、同一端末において現行または以前のタスクで、タブが必要な位置に設定されていなくてはなりません。これは通常端末制御 SEND コマンドによって行います。 595 ページの『第 35 章 データ伝送コマンドの使用』に説明があります。
- 外部形式制御。論理装置の中には形式設定情報を格納できるものがあり、形式設定処理に利用できます。これにより、BMS が送信するデータがかなり削減され (主にシンボリック・マップの内容)、物理マップとシンボリック・マップの組み合わせの作業を論理装置に代替させることができます。詳しくは、 883 ページの『外部形式制御』を参照してください。
- NLEOM (改行、メッセージ終結)。標準 BMS では、BMS が出力を 3270 バッファー制御命令ではなく、ブランクおよび改行 (NL) 文字で形式設定するように要求するオプションも使用できます。この手法を使用すると、プリンターでのページ幅をより柔軟に設定できます。 746 ページの『NLEOM オプション』に説明があります。

装置依存マップ

マップ・フィールドの位置、デフォルト属性、およびデフォルトの内容は物理マップにのみ表示され、シンボリック・マップには表示されないため、単一のプログラムを使用して、変数情報は同一だが定数情報は異なるマップを、画面上に異なる配置で作成できます。これは、複数の装置を異なる特性でサポートしなくてはならないプログラムを作成する場合に非常に便利です。

それぞれ異なる接尾部を持つ、名前は同一だが属性とレイアウトが異なる複数のマップを定義することによって、このマップを作成します。

例えば、「クイック・アップデート」トランザクションを使用する事務担当者のうち何人かは 3270 の 2 型を使用し (これまでにもあった想定です)、他の者は行が 3 つ、桁が 40 しかない特殊な目的のための端末を使用すると想定します。大きい画面のために設計した形式は小さい画面には合いませんが、情報は再配置すれば表示させることができます。

```
QUP Quick Account Update:
Current charge okay; enter next
Acct:  _______    Charge: $ _______
```

図 147. 小さい画面用の「クイック・アップデート」

次のマップ定義が必要です。

```
QUPSET   DFHMSD TYPE=MAP,STORAGE=AUTO,MODE=INOUT,LANG=COBOL,SUFFIX=9
QUPMAP   DFHMDI SIZE=(3,40),LINE=1,COLUMN=1,CTRL=FREEKB
         DFHMDF POS=(1,1),LENGTH=24,ATTRB=(ASKIP,BRT),                 X
               INITIAL='QUP Quick Account Update'
MSG      DFHMDF LENGTH=39,POS=(2,1),ATTRB=(ASKIP,NORM)
         DFHMDF POS=(3,1),LENGTH=5,ATTRB=(ASKIP,NORM),                 X
               INITIAL='Acct:'
ACCTNO   DFHMDF POS=(3,11),LENGTH=6,ATTRB=(UNPROT,NUM,IC)
         DFHMDF POS=(3,18),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
         DFHMDF POS=(3,20),LENGTH=7,ATTRB=(ASKIP,NORM),INITIAL='Charge:'
CHG      DFHMDF POS=(3,29),LENGTH=7,ATTRB=(UNPROT,NORM),PICIN='$$$$0.00'
         DFHMDF POS=(3,37),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
         DFHMSD TYPE=FINAL
```

図 148. マップ定義

マップのこのバージョンのアセンブルによって生成されたシンボリック・マップ・セットは、 817 ページの『入出力の例』に示したものと同一です。これは、名前を持つフィールドは同一の名前と同一の長さを持ち、マップ定義に同じ順序で表示されるためです (ただし画面で同じ順序に表示される必要はありません。すべてのマップにおいて命名済みフィールドの定義を同じ順序で保持する場合は、これらを再配置することもできます)。シンボリック・マップのコピーを必要とするだけで、同一コードを使用してマップを作成することができます。

CICS は、トランザクションを実行している端末に対する TYPETERM リソース定義の ALTSUFFIX オプションでコード化された値から使用するために、物理マップを選択します。また、トランザクション PROFILE リソース定義に SCRNSZE(ALTERNATE) を指定する必要があります。TYPETERM リソース定義および PROFILE リソース定義について詳しくは、「*CICS Resource Definition Guide*」を参照してください。

同じ方法を、特殊な目的に使用する標準端末の識別に使用してもいいでしょう。例えば、アプリケーションを英語とフランス語の両方で使用する場合、物理マップを 2 セット作成して 1 つは定数をフランス語に、もう 1 つは定数を英語にすることができます。それぞれに接尾部を割り当て、英語の接尾部を英語の端末の定義に ALTSUFFIX 値として指定し、フランス語の接尾部をフランス語の端末に指定します。このマップを使用するトランザクションは、代替画面サイズを指定した PROFILE を示します。次にマップを送信すると、BMS はその端末に一致する接尾部を持つバージョン (つまり、適切な言語) を選択します。

装置依存マップを提供するもう 1 つの方法は、BMS が端末タイプに基づく接尾部を生成し、さらに、ユーザーが SEND MAP または RECEIVE MAP を出したときに、BMS が現行の実行での端末に一致する物理マップを選択することです。

装置依存サポート: DDS

これを行う BMS 機能を「装置依存サポート」(DDS) といいます。 DDS は次のように作動するインストール・オプションです。

マップ・セットをアセンブルするとき、マップがある端末のタイプを TERM オプションに指定します。これによりアセンブラーは、その端末タイプを示す文字の接尾部を持つ MAPSET 名の物理マップ・セットを格納します。また、JCL またはリンク・エディットの NAME ステートメントを使用し、マップ・セットを格納するメンバー名を制御することもできます。SEND MAP または RECEIVE MAP を、DDS をアクティブにして出すと、BMS は 1 文字の接尾部を MAPSET オプションで指定する名前に追加します。端末の定義に基づいて接尾部を選択するため、どのような実行に対しても端末に対応する物理マップをロードします。

BMS は一般的な端末タイプに使用される接尾部を定義します。例えば、画面サイズが 24 行 80 桁の 3270 の 2 型には、「M」の文字が割り当てられています。タイプが表 44 に示す標準タイプのいずれか 1 つである場合、TYPETERM 定義から判別されます。

表 44. BMS 用の端末コード

コード	端末または論理装置
A	CRLP (カード読み取り装置入力、ライン・プリンター出力) または TCAM 端末。 **注:** CICS Transaction Server for z/OS バージョン 4 リリース 1 では、ローカル TCAM 端末はサポートされません。サポートされている TCAM 端末は、TCAM の DCB インターフェース (ACB ではない) によって CICS TS 3.1 以前の端末専有領域に接続されているリモート端末のみです。
B	磁気テープ
C	順次ディスク
D	TWX 33/35 型
E	1050
F	2740-1、2740-2 (バッファー受信なし)
G	2741
H	2740-2 (バッファー受信あり)
I	2770

表 44. BMS 用の端末コード (続き)

コード	端末または論理装置
J	2780
K	3780
L	3270-1 ディスプレイ (幅 40 文字)
M	3270-2 ディスプレイ (幅 80 文字)、LU タイプ 2
N	3270-1 プリンター
O	3270-2 プリンター、LU タイプ 3
P	対話式 LU すべて、3767/3770 インタープリター LU、3790 全機能 LU、SCS プリンター LU
Q	2980 の 1 型および 2 型
R	2980 の 4 型
U	3600 (3601) LU
V	3650 ホスト会話型 (3653) LU
W	3650 インタープリター LU
X	3650 ホスト会話型 (3270) LU
Y	3770 バッチ LU、3770 および 3790 バッチ・データ交換 LU、LU タイプ 4
ブランク	3270-2 (TERM を省略する場合はデフォルト)

インストール・システムでは、上に示した小型画面のような追加の端末タイプも定義することができます。システム・プログラマーは、ID を端末タイプに割り当て、端末のための TYPETERM 定義の ALTSUFFIX オプションにその ID を指定することによって、これを行います。このような端末にマップを作成するときは、この ID を TERM オプションではなく SUFFIX オプションに指定します。このマップを使用するトランザクションはまた、代替画面サイズを指定する PROFILE も示さなくてはなりません。これには、ALTSUFFIX が使用されます。

DDS を使用する場合、BMS が物理マップを選択するために従う規則は次のとおりです。

- 定義が ALTSUFFIX および ALTSCREEN の両方を指定し、トランザクションのための画面サイズが代替サイズである場合、BMS は 端末定義の ALTSUFFIX 値をマップ・セット名に追加します (トランザクション PROFILE が代替サイズを呼び出すためか、あるいはデフォルト・サイズと代替サイズが等しいためのいずれかによります)。
- これらの条件に合わない場合、または BMS がこの接尾部を持つマップを検出できない場合は、端末定義の端末タイプに対応する接尾部を持つマップを検出しようとします。
- このマップも検出できない場合、BMS は接尾部のないマップを探します (ブランクの接尾部は全目的用マップを示し、それを使用するどのような端末にも適応します)。

DDS を使用しない場合、BMS は常に最初に、接尾部のないマップを探します (そして、そのマップしか探しません)。

装置依存サポートは BMS のインストール・オプションで、システム・プログラマーによってシステム初期設定テーブルに設定されます。これを利用する前に、シス

テムに組み込まれていることを確認してください。サポートする装置タイプが 1 つだけの場合でも、組み込まれているかどうかの確認が必要です。

システムに DDS が組み込まれている場合、サポートする装置タイプが 1 つだけの場合でも、接尾部を持つマップ・セットの作成を効率よくできるという利点があります。これは、BMS が、汎用マップ・セット (ブランクの接尾部) にデフォルト設定される前に、存在しないマップ・セットをロードしようとするのを回避できるためです。

一方 DDS を組み込まれていない場合、マップに接尾部を付ける必要はありません。BMS は汎用接尾部 (ブランク) を探し、接尾部を持つマップの位置付けに失敗するためです。

ご使用の端末情報の検出

BMS の設計全般、特に装置依存サポートから考えて、形式設定するための端末について詳しく知る必要は概してありません。しかし、プリンシパル装置の特性を知る必要がある場合は、ASSIGN コマンドおよび INQUIRE コマンドを使用する方法があります。例えば、端末が特定の拡張属性をサポートするかどうか、どの言語が使用できるか、また画面サイズなどを知ることができます。このような情報は、BMS を使用する場合でも、また端末と通信するための端末制御を使用する場合でも適用されます。端末の制御では、この情報および適用するオプションに対するニーズがさらに増大します。これらについては、603 ページの『第 38 章 ご使用の端末情報の検出』の解説を参照してください。

また、BMS に固有の ASSIGN オプションもありますが、これらに対するニーズは ACCUM オプションを使用するときに最大になるので、後の 843 ページの『累積処理用の ASSIGN オプション』で説明します。

第 62 章 BMS マップ出力の送信

このタスクについて

シンボリック・マップ・セットのアセンブルを終えたら、コーディングができます。 781 ページの『第 61 章 マップの作成』で使用する例では、アプリケーション・プログラムからマップにデータを入手する方法について説明しています。ここではその処理をさらに詳しく解説し、実行する必要のあるすべてのステップを記載し、また提供されているオプションの詳細についても説明します。

マップ出力を作成するには、以下のステップを実行してください。

1. マップを作成するストレージを獲得する
2. このストレージの構造を定義するためにシンボリック・マップ・セットをコピーする
3. そのストレージを初期化する
4. 出力データをマップ構造に移動する
5. フィールド属性を設定する
6. SEND MAP コマンドを用いてマップを画面に書き、必要な装置制御情報をすべて追加する。

次のタスク

関連概念

775 ページの『第 60 章 基本マッピング・サポート』

マップ用ストレージの獲得および定義

このタスクについて

マップ出力を作成する最初のステップは、プログラムが BMS に渡す変数マップ・データを配置するためのストレージを提供することです。マップ構造を作業用ストレージに配置する場合は、CICS が割り振りを行います (CICS はプログラムの実行のたびに、作業用ストレージの専用コピーを割り振るため、あるタスクからのデータが他のタスクからのデータと混同されることがありません。 263 ページの『プログラム・ストレージ』に説明があります)。作業用ストレージを使用するためには、この目的のために提供されている言語ステートメントを用いて、その作業用ストレージにシンボリック・マップ・セットをコピーします。

```
COPY        in COBOL and assembler
%INCLUDE    in PL/I
#include    in C and C++
```

作業用ストレージは、COBOL では WORKING-STORAGE SECTION、PL/I、C、および C++ では自動ストレージ、CICS アセンブラー・プログラムでは DFHEISTG です。例:

© Copyright IBM Corp. 1989, 2009

```
WORKING-STORAGE SECTION.
...
01  COPY QCKSET.
...
```

他の方法として、CICS GETMAIN コマンドを使用して必要なときにマップ・セット・ストレージを獲得したり解放したりすることができます (GETMAIN については 653 ページの『ストレージ制御』 に説明されています。) この場合、ポインター変数によってアドレッシングされたストレージ (COBOL では LINKAGE SECTION、PL/I、C、および C++ では基本ストレージ、アセンブラーでは DSECT) 内に、マップをコピーします。 GETMAIN からの戻りでは、SET オプションに返されたアドレスを使用して、プログラム言語の機能に従いストレージとデータ構造とを関連付けます。

780 ページの図 144 に示した例では、作業用ストレージを使用しましたが、GETMAIN を使用することもできます。その場合には、コードを次のように変更します。

```
LINKAGE SECTION.
...
01  COPY QCKSET.
...
PROCEDURE DIVISION.
...
MOVE LENGTH OF QCKMAPO TO LL.
EXEC CICS GETMAIN SET(ADDRESS OF QCKMAPO)
            LENGTH(LL) END-EXEC.
...
```

GETMAIN コマンド上で必要な長さは、マップ名に「O」の文字の接尾部がある名前の変数の長さです。COBOL、PL/I、C、および C++ では、上記の例のように、言語機能を使用してこの長さを判別できます。アセンブラーでは、ラベルが「L」の接尾部を持つマップ名である EQUate ステートメントで定義されます。

BASE および STORAGE オプション

BASE および STORAGE=AUTO という DFHMSD マップ・セット定義マクロの 2 つのオプションは、マップのためのストレージをどのように定義するかに影響します (STORAGE オプションには常に値 AUTO があります)。この 2 つのいずれかを使用するか、またはいずれも使用しないか、*3* つ の選択肢があります。複数のマップを含むマップ・セットにいずれも指定しない場合は、互いにオーバーレイするようにこれらのマップに記号構造が定義されます。STORAGE=AUTO を指定するとそうはならず、それぞれが分離したスペースを占めます。このため、STORAGE=AUTO ではより多くのストレージが必要です。

しかし、単一のプログラムにおいて互いにオーバーレイする複数のマップを使用するときは、それらを連続して使用するか、またはプログラミングによってストレージの再利用を補償しなくてはなりません。ストレージが主な問題とならない限りは、STORAGE=AUTO を使用するとプログラミングが単純化され、エラーのリスクが軽減されます。

PL/I、C、および C++ の場合、STORAGE=AUTO にはマップを自動ストレージ (CICS が割り振るストレージ) として定義するという追加の機能があります。STORAGE=AUTO がないと、これらのコンパイラーは基本ストレージを想定するよ

うになり、それに対して、一般的に追加の GETMAIN のオーバーヘッドが発生します。BMS では、BASE オプションで別の名前を指定しない限り、BMSMAPBR という名前を関連したポインター変数に割り当てます。

3 つめの選択肢である BASE を指定すると、複数のマップ・セットにあるすべてのマップに対して同一のストレージを使用できます。その結果はプログラム言語によって少し異なりますが、本質的には、マップ・セット内の、同じ BASE 値を持つすべてのマップは互いにオーバーレイします。COBOL では、BASE=xxxx を指定すると 01 レベル (つまり、各個別マップ) に REDEFINES xxxx 文節が含まれます。PL/I、C、および C++ では、ポインター変数 xxxx に基づいて、各マップをストレージとして指定します。BASE は、プログラム言語がアセンブラーのときは使用できません。

出力マップの初期化

このタスクについて

出力の作成を始める前に、マップ・ストレージがヌルに初期化されているようにしてください。そうすれば、直前のプロセスによってストレージに残ったデータが不注意により使用されることはありません。この同じマップ、またはこのマップをオーバーレイするマップを使用して入力データを読み取っていた場合は、このデータを処理または保管したことを最初に確認する必要があります。入力マップと出力マップの関係については、 819 ページの『記号入力マップ』に説明があります。また、入力に使用したのと同じマップの使用については、 828 ページの『マップ入力後のマップ出力の送信』に説明があります。

ヌル (X'00') を構造に移動することにより初期化を行います。マップ域全体を *O* の文字の接尾部を持つマップ名で参照できるように、シンボリック・マップ構造が定義されています。これについては、 779 ページの図 143 に示していますが、実際のステートメントは以下のようになります。

```
MOVE LOW-VALUES TO QCKMAPO.
```

また実際、上に示したステートメントが、「クイック・チェック」の例で作成したマップがある区域を消去します。マップを入力と出力の両方に使用する場合は、1 つのフィールドのマップを、入力を編集しながら同時に消去する方が簡単かもしれません (826 ページの『入力エラーの取り扱い』を参照)。

CICS GETMAIN 命令でマップ・ストレージを入手するときには、INITIMG オプションを使用するという初期化方法もあります。

変数データのマップへの移動

マップのためのストレージを入手し、マップ構造とストレージとの関係を設定し、初期化を終えたら、いいいよ出力を作成できます。これは、データそのものとデータの表示属性という 2 つの部分に分かれます。最初にデータについて説明し、続いて属性について説明します。

通常の場合、出力表示は (物理マップが提供する) 定数またはデフォルト・データと、(プログラムが提供する) 変数データとから構成されています。プログラムによ

って提供する各フィールドに対し、接尾部の文字が *O* のマップに割り当てた名前のシンボリック・マップのフィールドにデータを移動します。例については、BMS の出力例を参照してください。

フィールドに値を指定しない場合 (つまり、初期化したときのままヌルにしておく場合)、BMS は、マップに割り当てられた初期値があれば通常それを使用します。定数 (つまり、名前のないフィールド) も、マップに指定された初期値を使用します。ただし、SEND MAP コマンドの DATAONLY オプションおよび MAPONLY オプションでは、このプログラムとマップ・データを組み合わせる方法が変わります。これらのオプションについては 808 ページの『シンボリック・マップと物理マップの組み合わせ』 で説明し、正確な規則については 810 ページの『出力画面の作成』 に要約されています。

表示特性の設定

表示属性は出力データの 2 番目のコンポーネントです (属性については 626 ページの『3270 フィールド属性』を参照してください)。 BMS 出力例の例では、ATTRB オプションによってマップ・フィールド用の 3270 フィールド属性がどのように定義されるか、およびフィールドに命名する場合に、プログラムによってマップ値を指定変更するよう BMS がどのように「A」サブフィールドを生成するかを説明しています。

すべての 3270 装置はフィールド属性をサポートするため、BMS は常に *A* サブフィールドを提供します。また、多くの 3270 装置では、表 45 に示す拡張属性のいくつかも提供します。BMS はこれらの属性を、フィールド属性ではひとまとめに行うのとほとんど同じ方法で、それぞれ個別にサポートします。属性値を DFHMDF フィールド定義に割り当てることができ、またフィールドに命名する場合は、BMS はシンボリック・マップにサブフィールドを生成するため、マップ指定の値をプログラムで指定変更できます。拡張属性の各タイプには分離したサブフィールドがあります。

DFHMDI または DFHMSD の DSATTS オプション内の必須属性を指定することによって、拡張属性のサブフィールドを要求することができます。また、MAPATTS オプションに拡張属性のリストを含める必要があります (これらの属性タイプがどの DFHMDF マクロにも表示されていなくてもこれを行う必要があります)。

表 45. BMS 属性タイプ: この表のカラムは、属性のタイプ、MAPATTS および DSATTS の値に関連した名前、シンボリック・マップでの関連したサブフィールドの接尾部を示しています。

属性タイプ	MAPATTS、DSATTS 値	サブフィールドの接尾部
フィールド属性	なし (デフォルト)	A
カラー	COLOR	C
強調表示	HILIGHT	H
枠線	OUTLINE	U
背景透明	TRANSP	T
妥当性検査	VALIDN	V
2 バイト文字機能	SOSI	M

表 45. BMS 属性タイプ (続き): この表のカラムは、属性のタイプ、MAPATTS および DSATTS の値に関連した名前、シンボリック・マップでの関連したサブフィールドの接尾部を示しています。

属性タイプ	MAPATTS、DSATTS 値	サブフィールドの接尾部
プログラム式シンボル	PS	P

注: プログラム式シンボルを使用する場合、装置内で常にロードされている記号セットを選択するのでない限り、最初に適切な記号セットを装置に送信するようにしてください。これは、端末制御 SEND コマンドを使用して行うことができます (595 ページの『第 35 章 データ伝送コマンドの使用』を参照)。「*IBM 3270 Information Display System Data Stream Programmer's Reference*」には、送信対象について説明されています。

適用される属性のタイプは、実行時のプリンシパル装置によって異なります。端末が持っていない属性に値を指定すると、BMS はそれを無視します。ただし、異なる端末タイプをサポートする場合は、同程度の視覚的明瞭度を得るために異なった手法を用いることが必要になるかもしれません。使用している端末の種類は、ASSIGN コマンドおよび INQUIRE コマンドを用いて知ることができます。これについては、603 ページの『第 38 章 ご使用の端末情報の検出』に説明があります。また、BMS にはプログラムを端末タイプから独立させておく機能も提供されています。797 ページの『装置依存マップ』を参照してください。

属性の変更

属性変更を使用する例を示します。「クイック・チェック」アプリケーションの端末にはカラー機能と強調表示機能があると想定します。許可される最大請求金額のフィールドは事務担当者にとって最大関心事であるため、このフィールドを画面の他の部分と異なるカラーで表示することにします。また、警告メッセージが出た場合には、事務担当者がこれに気付くことが重要なので、このメッセージを赤で表示します。さらにカードが盗難にあっていて、事務担当者の注意を引くことが非常に重要な場合、このメッセージが明滅するようにプログラムの属性を変更できます。これらの機能を追加するためには、マップ定義を次のように変更する必要があります。

```
QCKMAP   DFHMDI SIZE=(24,80),...,                                      X
               MAPATTS=(COLOR,HILIGHT),COLOR=GREEN,HILIGHT=OFF,DSATTS=HILIGHT
```

MAPATTS オプションは、マップにカラーと強調表示を指定することを BMS に指示します (DSATTS にリストされた属性はすべて MAPATTS にも組み込まなくてはならないため、プログラムに指定する場合も同様です)。COLOR 値および HILIGHT 値は、カラー指定がされていないフィールドを緑にすること、また強調表示が指定されていなければオフにすることを指示します。

変更する必要のあるフィールド定義は、緑ではない または強調表示される という定義のみです。

```
CHG      DFHMDF POS=(5,13),LENGTH=8,ATTRB=(ASKIP,NORM),PICOUT='$,$$0.00',    X
               COLOR=NEUTRAL
MSG      DFHMDF LENGTH=20,POS=(7,1),ATTRB=(ASKIP,NORM),COLOR=RED
```

COLOR=NEUTRAL の指定により、端末では、そのフィールドが白色で表示されるようになります。

DSATTS オプションは、実行時に一部のフィールドの強調表示を変更するよう BMS に指示するため、変更が行えるようにシンボリック・マップに「H」の接尾部を持つサブフィールドを生成します。命名済みフィールドはそれぞれ余分のサブフィールドを取得します。例えばメッセージ・フィールドは、779 ページの図 143 にある現行の 3 行から次のように拡張されます。

```
02  FILLER PICTURE X(2).
02  MSGH    PICTURE X.
02  MSGA    PICTURE X.
02  MSGO  PIC X(30).
```

明滅させるプログラム・ステートメントは次のとおりです。

```
MOVE DFHBLINK to MSGH.
```

一般に BMS は、プログラムに属性値を指定してあればプログラムから、指定していなければ (つまりプログラムの値を初期設定のままヌルにしてあれば) マップから、属性値を取得します。ただし、SEND MAP コマンドの MAPONLY オプションおよび DATAONLY オプションは、フィールド・データだけでなく属性値にも影響します。811 ページの『値の取得元』に説明があります。

属性値の定義: DFHBMSCA

属性を設定するために必要な 1 バイト値は、3270 ハードウェアで定義されるビットの組み合わせです。これらは覚えるのが難しく、言語によっては表し方が不便です。この問題を解決するため、CICS にはプログラムにコピーできるソース・コードが提供されています。DFHBMSCA と呼ばれるこのコードは、すべての属性に対して一般に使用されるすべての値を定義し、それぞれの組み合わせに意味のある名前を割り当てます。上に示したコードの行の DFHBLINK を例にします。DFHBLINK を定義するためには、次のように DFHBMSCA を作業用ストレージにコピーします。

```
WORKING-STORAGE SECTION.
...
01  COPY DFHBMSCA.
```

各プログラム言語に対して DFHBMSCA の別々のバージョンがありますが、値名はすべてのバージョンで同一です。 DFHBMSCA に含まれていない属性の組み合わせが必要な場合は、「*IBM 3270 Information Display System Data Stream Programmer's Reference*」を参照して値を決定できます。その値の使用頻度が高い場合は、その値を含むように DFHBMSCA を修正できます。

注: アセンブラー言語だけは値が EQUate を用いて定義されるため、MVC 命令ではなく MVI 命令を使用します。

第 63 章 SEND MAP コマンドの使用

SEND MAP コマンドは BMS に以下のことを指示します。

- 使用するマップ (MAP オプション)、およびそのマップがある場所 (MAPSET オプション)
- そのマップのための変数データがある場所 (FROM オプション) およびそのデータとマップからの値を組み合わせる方法 (MAPONLY および DATAONLY)
- データ・ストリームに組み込む装置制御、およびその他の制御オプション
- カーソルを置く場所 (マップ定義での位置を指定変更したい場合) (CURSOR オプション)
- メッセージは完了するか、または累積して作成されるかどうか (ACCUM オプション)
- 形式設定出力をどう扱うか (TERMINAL、SET および PAGING の各オプション)

MAP オプションおよび MAPSET オプションは自己説明型です。残りのほとんどのものについては、単純な SEND MAP に先行するプログラミング・ステップを解説しながら説明します。最後の 2 つのトピックでは、BMS 論理メッセージ機能の知識が必要です。これについては 814 ページの『出力後処理オプション: TERMINAL、SET、および PAGING』で説明します。

その点にいくまでは、次のようなデフォルトを前提とします。すなわち、各 SEND MAP は 1 つのメッセージを作成し、ここではそのメッセージを自分自身の端末に送信します。

関連概念

775 ページの『第 60 章 基本マッピング・サポート』

SEND MAP の制御オプション

BMS SEND コマンドには多くの制御オプションがあります。この中には特定の装置や BMS の特殊な機能にのみ適用されるものもあり、それらについては関連する装置のサポートまたは機能について述べるまで説明を待つことにします。しかし、以下の装置制御オプションは一般に適用されます。

- **ERASE、ERASEAUP**、および **FRSET** はすべて、端末にバッファーがある場合には、出力をバッファーに書き込む前に、装置バッファーの内容を修正します。ERASE はバッファー全体をヌル (X'00') に設定します。端末に代替画面サイズ機能があれば、ERASE はバッファー・サイズも設定します。このため、タスクの最初の SEND MAP は通常、バッファーのクリアとバッファー・サイズの選択の両方を行うために ERASE オプションを指定します (代替画面サイズについて詳しくは、 624 ページの『3270 書き込みコマンド』を参照してください。)

  ERASEAUP (全無保護フィールド消去) は、バッファーにある無保護のフィールド (つまり、オペレーターが変更できるフィールド) すべての内容をヌルに設定します。これはデータ入力に便利です。 809 ページの『DATAONLY オプション』に説明があります。

© Copyright IBM Corp. 1989, 2009

FRSET (フィールド・リセット) はバッファーにあるすべてのフィールドの修正データ・タグをオフにします (このオプションについては、 827 ページの『正しい入力の保管』および 627 ページの『変更』にさらに説明があります)。

- **FREEKB** (キーボードの解放) は、出力が端末に送信されるとキーボードをアンロックします。これは通常、ディスプレイ端末で使用します。
- **ALARM** は音響アラームを鳴らします (端末に備わっている場合)。
- **FORMFEED**、**PRINT**、**L40**、**L64**、**L80**、および **HONEOM** は印刷専用で、 744 ページの『CICS 3270 プリンターのオプション』に説明してあります。 **NLEOM** も主に印刷に使用され、同じセクションで説明されています。NLEOM には**標準** BMS が必要です。

これらのオプションのいくつかはマップそのものでも指定できます。特に、3270 書き込み制御文字で表され、DFHMDI マクロまたは DFHMSD マクロの CTRL オプションでコーディングされるオプション
(PRINT、FREEKB、ALARM、FRSET、L40、L64、L80、HONEOM) がそうです。

注: CTRL オプションは常にグループとして扱われるため、これらのオプションのいずれかを SEND MAP コマンドに組み込むと、BMS はマップ定義のそれらすべて に対する値を無視し、コマンドにあるものだけを使用します。前述のように、SEND CONTROL コマンドを使用して、装置制御オプションをマップ・データから切り離して送信することもできます。SEND MAP で使用できる SEND CONTROL のすべてのオプションを使用できますが、NLEOM などのように明らかにデータに関連するものは例外です。

その他の BMS SEND オプション: WAIT および LAST

タスクが BMS または端末制御 SEND コマンドを使用して端末に書き込みを行うとき、CICS は通常、伝送をスケジュールしてから再びタスクを実行可能にします。実際の伝送はやや後に行われ、端末タイプ、アクセス方式、またシステムのその他のアクティビティーによって異なります。プログラムが制御を取り戻す前に伝送を確実に完了させたい場合は、WAIT オプションを使用してください。

WAIT は、処理とタスクのための出力伝送との間のオーバーラップを防ぐため、応答時間が少し長くなることがあります (ただしオーバーラップが起こるのは、後続の SEND、RECEIVE まで、またはタスクの終了までの、いずれかの場合のみです。CICS は 1 つの端末操作を完全に終了してから別の操作を開始するためです)。

LAST オプションを使用することにより、応答時間をやや改善できる端末もあります。LAST は送信する出力がタスクの最後の出力であることを示します。このことがわかっていると、CICS は、データの伝送と、タスクの終了時に起こる VTAM のブラケットの終了フローとを結合できます。

シンボリック・マップと物理マップの組み合わせ

ここまでは、各表示は (物理マップが提供する) 定数データと、(プログラムが提供し、シンボリック・マップに従って構造化される) 変数データとから構成されていると想定してきました。しかし、これらのコンポーネントの 1 つ以上が脱落している場合もあります。

MAPONLY オプション

例えば、メニュー・マップにはプログラムが提供するデータのいずれも必要ない場合もあります。このような場合には、SEND MAP コマンドの FROM オプションの所に MAPONLY オプションをコーディングします。すると BMS は、物理マップからすべての情報を取り出し、無名の定数と命名済みフィールドとの両方に初期値を送信します。MAPONLY を使用して常にマップを送信しているプログラムに、シンボリック・マップ・セットをコピーする必要はありません。また実際、すべてのプログラムがこのマップ・セットのすべてのマップをこのような方法で使用する場合は、TYPE=DSECT マップ・セットのアセンブルを省略することができます。

MAPONLY は、入力専用マップを画面に表示する方法でもあります。

DATAONLY オプション

逆の状態もあり得ます。プログラムがすべてのデータを提供することができ、マップからの定数またはデフォルト値のいずれも必要としない場合です。このような状態は、マップの 2 番目とその後続の表示画面で、いろいろな状況において起こります。例えばデータ入力アプリケーション、オペレーターが同一の形式で表示される一連のレコードをブラウズする照会アプリケーション、および入力にエラーが検出された後に再表示される画面などです。

このような状態では、DATAONLY オプションを用いて指示すると BMS では便利に対応できます。位置決め情報のために使用しているマップおよびマップ・セットを BMS に示す必要はありますが、BMS はシンボリック・マップ内でヌルではない属性またはデータ値を持つフィールドのみを送信します。その他のフィールドおよび属性値は未変更のままです。

SEND CONTROL コマンド

データを送信する必要はまったくないが、装置制御を送信する必要がある場合もあります。例えば、画面を消去したりアラームを鳴らしたりするような場合です。このようなときは SEND CONTROL コマンドを使用して、必要なオプションをリストします。

データ入力アプリケーション内のプログラムについて考えます。最初の開始時に、入力フィールド、関連したラベル、画面見出し、および命令を使用して、画面を形式設定するためのデータ入力マップが表示されます。プログラムは変数データを送信しないため、この最初の SEND MAP コマンドが MAPONLY を指定します。その後で、プログラムは一連のデータ入力を受け入れます。この入力が正しければ、プログラムはこれをファイルして別のものを要求します。いずれの変数データも送信する必要はありません。必要なことは、オペレーターに次のレコードの入力を知らせるために、画面から入力を消去してキーボードをアンロックすることです。

```
EXEC CICS SEND CONTROL ERASEAUP FREEKB END-EXEC
```

上に示したように行ってください (これらの装置制御オプション、またその他のものの説明は、807 ページの『SEND MAP の制御オプション』を参照してください)。

エラーが起こった場合は、プログラムは変数データを送信して、オペレーターに問題の修正方法を指示する必要があります。この場合にはエラーの起こったフィール

ドを強調表示するためにそのフィールドの属性が変更され、この目的のために提供されているフィールドにメッセージが送信されます。ここではマップが既に画面に表示されているため、プログラムは DATAONLY オプションを使用しています (入力エラーの処理については詳しくは、 826 ページの『入力エラーの取り扱い』 に説明されています。)

MAPONLY、DATAONLY、および SEND CONTROL が適用できるときはこれらを使用してください。特に、データ入力状態で応答時間が重要な問題であるときはそうしてください。MAPONLY はパス長を節約し、DATAONLY はアウトバウンド・データ・ストリームの長さを短くします。また、SEND CONTROL はその両方を行います。

出力画面の作成

物理マップ定義オプション、SEND MAP オプション、プログラム・データ・オプションおよび組み合わせオプションの相互作用はかなり複雑なため、SEND MAP の後に、画面に表示されるものを決定するための規則の要約を順に示します。

画面 (バッファー) の内容は次のものから決定されます。

- SEND MAP コマンドの前に表示されていたもの
- SEND MAP コマンドから送信されたフィールド (フィールド属性、拡張属性、および表示データ)
- これらのフィールド・エレメントのための複数の値をどこから取るか

この順に、可能な選択肢について説明します。

開始時の実行内容

SEND MAP コマンドで最初に起こることは、ERASE オプションが指定されている場合、マップのサイズまたは元のマップには関係なく、画面 (バッファー) 全体がヌルに設定されることです。代替画面サイズ機能がある端末では、画面サイズも設定されます。 624 ページの『3270 書き込みコマンド』に説明があります。この画面は形式設定されていない状態で、定義されたフィールドもなく、表示データもありません。ERASEAUP が指定されていれば画面上の無保護フィールドはすべて消去されますが、すべてのフィールドのフィールド構造と属性、および保護フィールドの内容は未変更です。

ERASE および ERASEAUP は、SEND MAP データがバッファーにロードされる*前に* 適用されます。これらのオプションがいずれも SEND MAP に指定されない場合、画面バッファーは直前の書き込み操作後の状態、つまりオペレーターによる修正をすべてそのままにして開始します。一般に、フィールドの (すなわち属性バイトの) 位置およびそれらの属性は未変更ですが、無保護フィールドのデータ内容は変更されることがあります。さらに、オペレーターが CLEAR キーを使用した場合、バッファー全体がヌルにクリアされ、画面は形式設定されていない状態になります。この状態は、ERASE オプションを組み込んだ場合と同じ状態です。

送信される内容

次に、BMS はマップの限度内で、バッファーでのそれらの位置のみを変更します。この領域の外側ではバッファーの内容は未変更ですが、マップの外側の領域で表示を変更することは可能です。 812 ページの『マップの外側』に説明があります。

マップ域の内側で何を送信するかは DATAONLY オプションが指定されているかどうかによって異なります。通常の場合ではこのオプションを指定せず、BMS はマップにあるすべてのフィールドのすべてのコンポーネント (フィールド属性、拡張属性、表示データ) を送信します。これにより POS オペランドで指定された位置にフィールドが作成され、POS から LENGTH フィールドに指定されたバイト数がオーバーレイされます。表示データの終了位置と次の属性バイト (POS 値) との間のバッファー位置は未変更のままです (異なるマップを使用した最後の書き込み操作の後で ERASE を指定しなかった場合、これらの間に入るスペースにフィールド (属性バイト) があることもあり、ないこともあります)。

これらのフィールド・エレメントの値は、プログラム、マップ、またはデフォルトから取られます。これについては次のセクションで説明します。

一方 DATAONLY が指定されていると、BMS はそれらのフィールドのみ、およびそれらのフィールドのためのコンポーネントのみを送信します。これらはプログラムによって提供されます。その他の画面データは未変更です。

値の取得元

画面の内容を決定する値は、次の 4 つのソースから取ることができます。

- プログラム
- マップ
- ハードウェアのデフォルト
- 直前の画面内容

BMS は、各マップ・フィールドの各コンポーネントをそれぞれ別々に考え、プログラムから値を取りますが、それは以下のような場合です。

- MAPONLY オプションが使用されていません。
- マップ内のフィールドに名前があるため、記号出力マップは、データを入手するための対応するサブフィールドのセットを含んでいます。フィールド属性値は、名前が A の接尾部を持つマップ・フィールド名であるプログラム・サブフィールドから取ります。表示データは 0 の接尾部を持つ同一名のサブフィールドから取り、拡張属性値は、その属性を識別する文字の接尾部を持つ同一名のサブフィールドから取ります (804 ページの表 45 を参照)。拡張属性の場合、その属性は、シンボリック・マップが対応するサブフィールドを含むことができるように、DSATTS の中にも表示されなくてはなりません。
- 値が設定されています。「設定」の定義はフィールド・コンポーネントによって少し異なります。
  - フィールド属性バイトでは、値はヌル (X'00') または入力操作から残されることのある値のいずれか (X'80'、X'02'、または X'82') であってはなりません。
  - 拡張属性バイトでは、値はヌルではいけません。

注: BMS は、端末がサポートするものとして定義されている拡張属性値のみを送信します。その他の拡張属性の値は、最終的なデータ・ストリームから省略されます。

– 表示データでは、データの先頭文字がヌルであってはいけません。

これらの条件のいずれもそろわない場合、次のステップは DATAONLY が指定されているかどうかによって変わります。DATAONLY が指定されていると、BMS はここで処理を停止し、プログラムから入手したデータのみを送信します。BMS は、プログラムによって変更されないコンポーネントは画面上で変更されないという方法でこれを行います。特に拡張属性値は、新規の値を指定するか、またはハードウェアのデフォルトを要求しない限り変更されません (X'FF' の値は背景透過性以外のすべての拡張属性に対してハードウェアのデフォルトを要求します。背景透過性については、ハードウェアのデフォルトを入手するためには X'F0' を指定します)。

DATAONLY を指定しないと、先に挙げた条件のいずれか 1 つがそろわない場合、BMS は次のようにしてマップからデータを入手します。

- フィールド属性では、そのフィールドのための ATTRB オプションで値を取ります。何も指定されていないと、BMS は (ASKIP、NORM) の ATTRB 値を想定します。
- 拡張属性では、BMS は次のようにして値を取ります。
  – DFHMDF フィールド定義の対応するオプション。
  – ここに指定されていない場合は、DFHMDI マップ定義の対応するオプションから値を取ります。
  – どちらも指定されていない場合、DFHMSD マップ・セット定義の対応するオプションから値を取ります。

  (どこにも値が指定されていない場合 BMS は値を送信しないため、3270 はハードウェアのデフォルト値を使用します。)
- 表示データでは、マップの初期値から取ります (INITIAL、XINIT、または GINIT オプション)。初期値がない場合フィールドはヌルに設定されます。

マップの外側

ここまでは、マップが画面またはプリンターのページと同サイズであると想定してきましたが、そうである必要はありません。アプリケーションで画面領域の一部のみを使用したり、出力を増大させたり、またはその両方を行うこともできます。

BMS 論理メッセージを使用すると複数のマップから 1 つの画面を作成することができ、それを一回の端末への書き込みで送信します。ACCUM オプションを使用してこれを行います。 833 ページの『第 65 章 BMS の論理メッセージ』に説明があります。ACCUM を使用しない場合でも、端末が 3270 と似たタイプの、バッファーのある装置であれば複数のマップから 1 つの画面を作成できます。この場合は画面 (バッファー) の異なる領域に書き込まれた複数の SEND MAP コマンドを使用し、最初のコマンドの後に消去しません。各 SEND MAP は出力を作成し、ディスプレイ装置に短い「明滅」を生成することがあります。この理由から、また余分な入出力のパス長を除去するため、このような複合画面には論理メッセージを使用するといいでしょう。

送信されたばかりのマップの外側ではバッファーの内容は未変更ですが、先に述べたように ERASE および ERASEAUP の効果は例外です。一般にこのことは、画面の対応する領域が未変更であることを意味します。ただし、マップの外側の画面位置は、その属性をマップ内部のフィールドから入手することができます。(異なるマップを使用して) マップの境界を超え、問題の位置の前にフィールドを定義しない場合、このようになります。この外側の位置を管理するマップの中のフィールドの属性を変更すると、この位置の表示が、内容が変わらなくても変更されることがあります。

GDDM および BMS の使用

ここまでに説明したバッファー・オーバーレイの技法の使用例の 1 つに、BMS と図形データ表示管理プログラム (GDDM) 出力の混合を含む画面の作成があります。一般には先に BMS 出力を書き、次に GDDM 出力を書きます。 BMS マップに GDDM 出力用のスペースを残しておくことができます。または、フィールドを持たないマップを書くことにより、任意の表示に「図形ホール」を作成し、そのホールを配置できます。このようなマップを「ヌル・マップ」といい、サイズ (高さおよび幅) はホールのサイズに一致します。

GDDM を使用して、図形と BMS 出力を結合する場合は、GDDM PSRSRV 呼び出しを組み込んで、BMS が使用する可能性のあるプログラム式シンボル・セットを GDDM が壊さないようにする必要があります。

カーソルの位置決め

通常、カーソルの初期位置は、必要なフィールドの ATTRB 値に「カーソル挿入」(IC) を組み込むことによりマップ定義に設定します (カーソル位置はこれまでに説明した出力専用マップでは重要ではありませんが、マップを入力にも使用するときには重要になります)。

SEND MAP コマンドの CURSOR オプションを使用すると、マップが表示されるときに、必要であればこの指定を指定変更できます。CURSOR(値) を指定すると、BMS は画面上でその絶対位置にカーソルを置きます。最初の行および桁 (0 の位置) で数え始め、行にまたがって進みます。つまり、80 桁の表示画面でカーソルを 3 行目の 4 桁目に置くためには、CURSOR(163) とコーディングします。

値なしで CURSOR を指定すると、BMS では「記号カーソルの位置決め」と解釈します。カーソルをマイナス 1 (-1) にしたいフィールドの長さサブフィールドを設定することにより、これを行います。長さサブフィールドは出力専用マップでは定義されないので、記号カーソルの位置決めを使用するためには、マップを INOUT として定義しなくてはなりません (長さサブフィールドについては 822 ページの『形式設定画面の入力』に、また INOUT マップについては 817 ページの『第 64 章 マップ・データの受信』に説明があります)。この方法で複数のフィールドをマークする場合、BMS は先に検出したものを使用します。

記号カーソルの位置決めは、端末オペレーターが誤ったデータを入力したとき入出力マップに特に便利です。フィールドの妥当性検査を行う場合、エラーになっているフィールドの長さを -1 に設定すると、再表示するときに BMS は最初のエラーのもとにカーソルを配置します。 826 ページの『マップ入力の処理』 には、この手法が記載されています。

SEND CONTROL コマンドを使用してカーソルを置くこともできますが、その場合は CURSOR に絶対値を指定することによってのみ可能です。SEND CONTROL の CURSOR を省略するとカーソルは移動しません。

無効データおよびその他のエラーの送信

SEND MAP コマンドおよび SEND CONTROL コマンドで発生する可能性のある例外状態については、そのコマンドの説明とともに、 「*CICS* アプリケーション・プログラミング・リファレンス」にリストされています。これらのほとんどは、高水準の BMS オプションである論理メッセージ、区分画面、および特殊な装置に対してのみ当てはまります。

しかし、無効データを端末に送信することもあり得ます。BMS は、カラーなどの拡張属性を、その属性をサポートするように定義されていない端末に送信を試みることはありませんが、シンボリック・マップの属性値およびデータ値の妥当性検査はしません。

無効データの影響は、特定の端末およびその誤ったデータの性質の両方によって変わります。無効データが制御シーケンスとして解釈されてしまうと、装置はそのデータを受け入れますが、誤った出力が生成されます。画面がエラー標識を表示することもあり、ATNI 異常終了になることもあります。タスクが ATNI の通知を受けるタイミングは、WAIT オプションを指定したかどうかによって異なります (808 ページの『その他の BMS SEND オプション: WAIT および LAST』を参照)。

出力後処理オプション: TERMINAL、SET、および PAGING

これまでに説明した中で唯一の後処置オプションは TERMINAL です。これは出力をタスクのプリンシパル装置に送信します。TERMINAL は、他の後処置を指定しない場合に指定されるデフォルト値です。ただし、他に 2 つの選択肢があります。

1. BMS は形式設定された出力ストリームを、端末に送信するのではなくタスクに戻すことができます。これを要求するためには SET 後処置オプションを使用します。このようにして、伝送を延期するか、データ・ストリームを修正して、特別な要件を満たすこともできます。 801 ページの『マップ用ストレージの獲得および定義』 には、SET の使用方法および用途ついて説明されています。
2. 後で端末に送るために、BMS が出力を CICS 一時記憶域に格納して管理するようにすることができます。このオプションを PAGING といい、メッセージが複数の画面またはページにわたることを暗黙指定します。ディスプレイ端末に、その画面容量を超えるメッセージを送信したいとき特に便利です。BMS は完了したことを指示するまでメッセージ全体を一時記憶域に保管します。次に、端末で出力を参照できる機能をオペレーターに提供します。オペレーター制御は必要ではありませんが、PAGING はディスプレイだけでなくプリンターにも使用できます。TERMINAL で十分な場合もあります。

PAGING を使用する場合、今述べたように出力は、間接的ながらまだ自分自身のプリンシパル装置に向かいます。完全 BMS にもルーティングの機能があり、この機能を使用すると他の 1 つ以上の端末に、自分自身の端末の代わりに、またはそれに加えて、メッセージを送信できます。ルーティングについては、前提条件の解説の後に 853 ページの『第 68 章 メッセージ・ルーティング』で説明します。

注: PAGING オプションおよび SET オプション、またその関連オプションには**完全** BMS が必要です。TERMINAL は、最小 BMS および標準 BMS で使用可能な唯一の後処置オプションです。

SET の使用

BMS メッセージに SET の後処置を指定すると、BMS は出力を形式設定し、それを装置依存データ・ストリームの形式で返します。端末の入出力は起こりませんが、通常返されたデータ・ストリームは、その後端末に送信されます。

BMS がデータ・ストリームを送信せずに形式設定するようにするのは、いくつかの理由があります。以下に示すことのいずれかを行う場合が考えられます。

- CICS によって明示的にサポートされない特殊機能または制約事項のある装置の要件に合うように、データ・ストリームを編集する。
- 標準 3270 機能または特殊装置特性に基づいて、データ・ストリームを圧縮する。
- CICS に直接接続されていない端末にデータ・ストリームを転送する。例えば、APPC リンクによって CICS に接続されたシステムに付加された 3270 端末にデータを渡してもいいでしょう。このデータを SET によって形式設定し、それによって生成されるページを、このリンクを介してパートナー・プログラムに送信できます。端末がプリンシパル装置と異なるタイプの場合は、適切なタイプのダミー端末を定義して、次に SET を使用してその端末にルーティングすると、正しい形式設定にすることができます。862 ページの『SET でのルーティング』に説明があります。

BMS は、SET オプションに名前を指定されたポインター変数を設定することにより、形式設定された出力をページ・リスト のアドレスに返します。このリストは、1 つ以上の 4 バイト項目で構成されています。この項目の形式は、それぞれ 1 ページの出力に相当しています。次のとおりです。

表 46. ページ・リスト項目形式

バイト	内容
0	端末タイプ (798 ページの表 44 を参照)
1 ～ 3	形式設定された出力ページを含む TIOA のアドレス

端末タイプで -1 (X'FF') を含む項目はページ・リストの終了を示します。このリストのアドレスは 24 ビット長のみであることに注意してください。プログラムが 31 ビット・アドレッシングを使用する場合は、24 ビット・アドレスの前に 2 進ゼロを付けてフルワードに拡張してから、そのアドレスを使用しなくてはなりません。

各 TIOA (端末入出力域) は、これらの区域では標準形式です。

表 47. TIOA 形式

フィールド名	位置	長さ	内容
TIOASAA	0	8	CICS ストレージ・アカウンティング情報 (8 バイト)
TIOATDL	8	2	ハーフワード 2 進数形式による TIOADBA フィールドの長さ
(名前なし)	10	2	予約フィールド

表 47. TIOA 形式 (続き)

フィールド名	位置	長さ	内容
TIOADBA	12	TIOATDL	形式設定された出力ページ
(名前なし)	TIOATDL + 12	4	ページ制御域、SEND TEXT MAPPED コマンドのとき必要 (使用する場合)

BMS がページを返すためにリストを使用する理由は、BMS コマンドの中に複数ページを生成するものがあるためです。SEND MAP はこれを行いませんが、SEND TEXT では可能です。さらに、ルーティング環境を設定した場合は、BMS は宛先の中の端末タイプごとに別々の論理メッセージを作成するため、単一の BMS コマンドから複数の異なる端末タイプに対するページを入手することもできます。端末タイプは、ページが属するメッセージを示します (所定のタイプに対するページは、常に順番に表示されます)。ルーティングしない場合、端末タイプは常にプリンシパル装置のタイプです。

ACCUM オプションを使用しない場合、ページはそれらを作成する BMS コマンドからの戻りで利用可能です。しかし ACCUM を使用すると、BMS はページの使用可能スペースが使用されるまで待機します。BMS は RETPAGE 条件をオンにして、プログラムにページが使用可能であることを通知します。HANDLE CONDITION コマンドを使用して、または BMS コマンドからの応答 (EIBRESP 内の、または RESP オプションに返された値) をテストすることにより、RETPAGE を検出できます。

BMS はページ・リストを再利用するため、BMS が戻すたびに、このリストの情報を取り出さなくてはなりません。ページのアドレスのみを保管してください。内容を保管する必要はありません。BMS はページそのものは再利用せず、実際はページのためのストレージを、BMS の制御からタスクの制御に移動します。これにより、終了するときにページのためのストレージを解放することができます。これを行うと、FREEMAIN コマンドの DATA オプションまたは DATAPOINTER オプションは、TIOASAA フィールドではなく TIOATDL フィールドを示します。

第 64 章 マップ・データの受信

形式設定画面は出力に対してと同様、入力に対しても重要です。データ入力アプリケーションがそのいい例ですが、他のほとんどのアプリケーションでも形式設定入力を、少なくとも部分的に使用します。入力では、BMS は出力の場合と大体逆のことを行います。つまり装置制御文字をデータ・ストリームから除去し、入力フィールドをデータ構造に移動するため、それらを名前で識別できます。

マップは入力専用、出力専用 (この例は既に説明しました)、またはその両方に使用できます。入力専用マップは比較的まれなので、ここでは入出力マップの特殊な例として、これらを使用する場合の違いを指摘しながら説明します。

関連概念

775 ページの『第 60 章 基本マッピング・サポート』

入出力の例

入力構造の詳細を説明する前に、「クイック・チェック」の例をもう一度考えてみます。顧客の新規請求金額が取引に通知されるとき、夜間実行の間に請求金額が何度も繰り返して限度額に達することが方針に合わないと想定します。一日ごとの実行合計を保持することにより「クイック・チェック」の処理能力を増大する新しいトランザクションが必要になります。

さらに、入力と出力の両方に同一の画面を使用すると想定するため、顧客ごとの画面項目は 1 つだけです。新しいトランザクション「クイック・アップデート」では、事務担当者は取引番号と請求金額の両方を同時に入力します。通常の応答は、以下のとおりです。

```
QUP                     Quick Account Update
Current charge okay; enter next
Account:    _______
Charge:   $ _______
```

図 149. 通常の「クイック・アップデート」の応答

トランザクションを受け入れないときは、入力情報を画面に残しておくと、事務担当者は問題の説明と一緒に入力されたものを見ることができます。

```
QUP                     Quick Account Update
Charge exceeds maximum; do not approve
Account:     482554
Charge:   $ 1000.00
```

図 150. 「クイック・アップデート」のエラー応答

(ここでも説明を簡単にする便宜上、マップを短くするために実際より単純化しています。)

この例で必要なマップ定義は次のとおりです。

© Copyright IBM Corp. 1989, 2009

```
QUPSET   DFHMSD TYPE=MAP,STORAGE=AUTO,MODE=INOUT,LANG=COBOL,TERM=3270-2
QUPMAP   DFHMDI SIZE=(24,80),LINE=1,COLUMN=1,CTRL=FREEKB
         DFHMDF POS=(1,1),LENGTH=3,ATTRB=(ASKIP,BRT),INITIAL='QUP'
         DFHMDF POS=(1,26),LENGTH=20,ATTRB=(ASKIP,NORM),                X
               INITIAL='Quick Account Update'
MSG      DFHMDF LENGTH=40,POS=(3,1),ATTRB=(ASKIP,NORM)
         DFHMDF POS=(5,1),LENGTH=8,ATTRB=(ASKIP,NORM),                  X
               INITIAL='Account:'
ACCTNO   DFHMDF POS=(5,14),LENGTH=6,ATTRB=(UNPROT,NUM,IC)
         DFHMDF POS=(5,21),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
         DFHMDF POS=(6,1),LENGTH=7,ATTRB=(ASKIP,NORM),INITIAL='Charge:'
CHG      DFHMDF POS=(6,13),ATTRB=(UNPROT,NORM),PICIN='$$$$0.00'
         DFHMDF POS=(6,21),LENGTH=1,ATTRB=(ASKIP),INITIAL=' '
         DFHMSD TYPE=FINAL
```

図 151. 入出力マップのマップ定義

この入出力マップのためのマップ・フィールド定義は、画面内容の変更を許可している場合に、出力専用の「クイック・チェック」マップの定義と非常によく似ていることがわかります。注意すべき相違点を以下に述べます。

- DFHMSD マップ・セット定義の MODE オプションは INOUT で、このマップ・セットのマップが入出力の両方に使用されることを示します。INOUT により、BMS はマップ・セットのすべてのマップに対し、出力だけでなく入力のためにも記号構造を生成します。もしこれが入力専用マップなら、MODE=IN とし、BMS は入力構造のみを生成します。
- 出力 (MSG) を送信するフィールドだけでなく、入力 (ACCTNO および CHG) を入手するフィールドにも名前を付けます。出力専用マップと同様、シンボリック・マップ内のスペースを節約するため、定数フィールドには命名しないようにします。
- 入力フィールドの ACCTNO および CHG は無保護 (UNPROT) になっており、オペレーターはここにデータを入力できます。
- IC (カーソル挿入) は ACCTNO に指定されています。これにより、マップが最初に表示されたときにカーソルが顧客番号フィールドの先頭に置かれ、オペレーターが入力しなければならない最初の項目の準備ができているようになります (この位置はマップを送信するときに指定変更できます。IC はデフォルトの位置を提供するだけです)。
- ACCTNO フィールドのすぐ後に単一ブランクから成る固定フィールドがあり、CHG フィールドの後にも同様のものがあります。これらは「ストッパー」フィールドといいます。通常これらは、直後に他のフィールドが続かない各入力フィールドの後に置かれます。これらはオペレーターが提供されたスペースを超えて、画面の未使用領域にデータを入力することを防ぎます。

  ストッパー・フィールドを「自動スキップ」として定義すると、カーソルは、オペレーターが前の入力フィールドに入力した後で、次の無保護フィールドにジャンプします。これはほとんどの入力フィールドが固定長である場合に便利です。オペレーターは、フィールドからフィールドへカーソルを先に進める必要がないためです。

  ストッパー・フィールドを「自動スキップ」ではなく「保護」として定義すると、オペレーターがフィールドの終わりを超えて入力しようとしたときにキーボードがロックされます。ほとんどの入力フィールドが可変長である場合 (通常、カーソル前進キーを使用する必要がある) は、こちらをお勧めします。これは、

オペレーターがオーバーフローを即時に警告されるためです。どちらを選択するにしても、可能な限りアプリケーション全体を通して同一の選択を行ってください。そうすると、オペレーターに対して一貫したインターフェースになります。

- CHG フィールドには PICIN というオプションがあります。PICIN は COBOL および PL/I で便利な編集マスクをシンボリック・マップに生成し、フィールド長を暗黙指定します。PICIN の使用の詳細については、「*CICS アプリケーション・プログラミング・リファレンス*」の を参照してください。

図 152 は、この INOUT マップ定義の結果のシンボリック・マップ・セットを示します。

```
01  QUPMAPI.
    02  FILLER PIC X(12).
    02  FILLER PICTURE X(2).
    02  MSGL    COMP  PIC  S9(4).
    02  MSGF    PICTURE X.
    02  FILLER REDEFINES MSGF.
      03 MSGA    PICTURE X.
    02  MSGI  PIC X(40).
    02  ACCTNOL    COMP  PIC  S9(4).
    02  ACCTNOF    PICTURE X.
    02  FILLER REDEFINES ACCTNOF.
      03 ACCTNOA    PICTURE X.
    02  ACCTNOI  PIC X(6).
    02  CHGL    COMP  PIC  S9(4).
    02  CHGF    PICTURE X.
    02  FILLER REDEFINES CHGF.
      03 CHGA    PICTURE X.
    02  CHGI  PIC X(7) PICIN '$,$$0.00'.
01  QUPMAPO REDEFINES QUPMAPI.
    02  FILLER PIC X(12).
    02  FILLER PICTURE X(3).
    02  MSGO  PIC X(40).
    02  FILLER PICTURE X(3).
    02  ACCTNO  PICTURE X(6).
    02  FILLER PICTURE X(3).
    02  CHGO  PIC X.
```

図 152. 「クイック・アップデート」のシンボリック・マップ

この構造の、QUPMAPO で開始する 2 つ目の部分は記号*出力* マップです。この構造は、データを画面に再度送信するために必要です。再定義したフィールド以外は、MODE=INOUT の代わりに MODE=OUT を指定した場合に生成されるものとほとんど同じに見えます。比較のため 778 ページの図 141を参照してください。主な違いはフィールド属性 (A) サブフィールドが脱落しているように見えることですが、これについてはすぐに説明します。

記号入力マップ

QUPMAPI というラベルの下にある、構造の最初の部分は新規です。これは記号入力 マップです。この構造は、QUPMAP マップで形式設定された画面からのデータを読み取るために必要です。マップの各命名済みフィールドには 3 つのサブフィールドが含まれます。記号出力マップの場合と同様、各サブフィールドはマップ・フィールドと同一の、目的を示す文字の接尾部を持つ名前を持ちます。入力に関連する接尾部およびサブフィールドは以下のとおりです。

L　　マップ・フィールドの入力の長さ

F フラグ・バイト。オペレーターがフィールドを消去したかどうか、またカーソルがフィールドに残っているかどうかを示します。

I 入力データそのもの

入出力構造は、フィールドごとに互いにオーバーレイするように定義されます。つまり、所定のマップ・フィールドの入力 (I) サブフィールドは常に、対応する出力 (O) サブフィールドと同一のストレージを占有します。同様に、入力フラグ (F) サブフィールドは出力属性 (A) サブフィールドをオーバーレイします (インプリメンテーション上の理由から、サブフィールド定義の順序は言語によってやや異なります。COBOL の場合、A サブフィールドの定義は INOUT マップの入力構造に移動しますが、出力専用マップの場合と同様に出力にも適用されます。アセンブラーの場合、入出力サブフィールド定義は各マップ・フィールドにはさみこまれます)。

BMS はダミー・フィールドを使用して、構造の一部分に、他の部分では発生しないサブフィールドのためのスペースを残します。例えば、出力マップには、たとえ出力専用マップであっても、常に入力マップの長さ (L) サブフィールドに対応する 2 バイトの充てん文字があります。カラーまたは強調表示など、拡張属性のための出力サブフィールドがある場合、BMS はこれらに一致させるため入力マップにダミー・フィールドを生成します。これらのフィールドの例 (COBOL では FILLER) は、778 ページの図 141 と 819 ページの図 152 の両方で見ることができます。

入出力マップ構造のフィールドの対応は、マップを入力に使用して、次に同じ形式で再度書き込むプロセスで非常に便利です。例えば、データ入力トランザクションや、誤った入力を入手してオペレーターに正しい入力を要求しなければならないときなどです。

マップ入力のプログラミング

マップ入力に必要なプログラミングはマップ出力の場合と同様ですが、ここでは当然、データは逆方向に向かいます。マップ出力の場合と同様、最初にマップを定義してアセンブルします。端末から読み取る 1 つ以上のプログラムで、以下のことを行います。

1. シンボリック・マップ・セットに対応するストレージを獲得する
2. このストレージの構造を定義するためにシンボリック・マップ・セットをコピーする
3. RECEIVE MAP コマンドを使用して入力データを形式設定する
4. 入力を処理する

これらのタスクおよび関連するトピックについて、以下の段落で述べますが、初めに RECEIVE MAP コマンドについて説明します。また、「クイック・アップデート」トランザクションのコードも開発します。

「クイック・アップデート」および他のほとんどのトランザクションがそうですが、トランザクションがマップ出力も呼び出す場合、前に 801 ページの『第 62 章 BMS マップ出力の送信』 で概説したステップを継続してください。マップ入力の考慮事項およびショートカットについては、 828 ページの『マップ入力後のマップ出力の送信』に記載されています。

RECEIVE MAP コマンドの使用

RECEIVE MAP コマンドを使用すると、BMS は端末入力データを形式設定し、そのデータにアプリケーション・プログラムがアクセスできるようにします。次のことが BMS に指示されます。

- 入力データ・ストリームの形式設定で使用するマップ。つまり、画面上の形式とプログラムが求めるデータ構造 (MAP オプション)
- このマップを検出する場所 (MAPSET オプション)
- 入力を受け取る場所 (TERMINAL オプションまたは FROM オプション)
- 大文字への変換を抑制するかどうか (ASIS オプション)
- 形式設定された入力データを入れる場所 (INTO オプションおよび SET オプション)

MAP オプションおよび MAPSET オプションはともに、使用するマップを BMS に指示します。これらは SEND MAP コマンド上のものとまったく同じ働きをします。

FROM オプションを使用しない限りは、BMS は形式設定する入力データを、タスクに関連した端末 (そのプリンシパル装置) から入手します。FROM は、デフォルトである TERMINAL の代替で、比較的まれな状況で使用されます (830 ページの『その他の入力の形式設定』を参照)。

BMS はまた、小文字の入力を自動的に大文字に変換する場合もあります。変換の制御の仕方については、 823 ページの『大文字変換』で説明します。

RECEIVE MAP の INTO オプションまたは SET オプションを使用して、形式設定済みの入力の位置を BMS に指示します。

画面上のデータに加えて、RECEIVE MAP コマンドではオペレーターがカーソルを残した場所、および伝送に使用したキーを知ることができます。この情報は、RECEIVE MAP コマンドを完了すると EIB で利用可能になります。EIBAID は伝送キー (「アテンション ID」または AID) を識別し、EIBCURSR はカーソルが残された場所を示します。

マップ入力用ストレージの取得

RECEIVE MAP コマンドを発行する場合、BMS ではその入力マップ構造を作成するストレージを必要とします。このスペースは、ユーザー自身が、プログラムの作業用ストレージで、または CICS GETMAIN を使用して、提供することができます。これらは出力マップを作成するためのストレージの割り振りで行うのと同一の選択であり、同じ方法で使用します (これらの詳細および例については、 801 ページの『マップ用ストレージの獲得および定義』を参照)。どちらの場合も、RECEIVE コマンドで INTO オプションをコーディングし、形式設定した入力が置かれる変数に命名します。例えば「クイック・アップデート」の例では、必要なコマンドは次のとおりです。

```
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET')
          INTO(QUPMAPI) END-EXEC.
```

通常、受信する変数はシンボリック入力マップによって定義された領域であり、BMS は、上記のように「I」の文字の接尾部を持つマップ名を割り当てます。ただし、必要であれば他の変数を指定することもできます。

入力操作では、ストレージ獲得のために 3 つ目の選択があります。SET オプションをコーディングすると、BMS は RECEIVE コマンド時にストレージを獲得し、SET オプションで命名されたポインター変数にアドレスを返します。したがって、「クイック・アップデート」の例では、次のように RECEIVE MAP コマンドをコーディングすることもできます。

```
LINKAGE SECTION.
...
01 QUPMAP COPY QUPMAP.
...
PROCEDURE DIVISION.
...
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET')
        SET(ADDRESS OF QUPMAPI) END-EXEC.
...
```

この方法で獲得されたストレージは、FREEMAIN を出して解放しない限り、タスクの終了まで保持されます (653 ページの『ストレージ制御』を参照)。

形式設定画面の入力

先にふれたように、3270 装置を使用した端末からの入力の受信について説明します。3270 以外の端末を使用する場合は、 794 ページの『3270 以外の端末のサポート』もお読みください。

CICS は通常、「変更読み取り」コマンドを使用して 3270 画面を読み取ります。CICS には端末制御 RECEIVE コマンドのための BUFFER というオプションがあり、これを使用して 3270 画面全体の内容を獲得できます。これ実行する必要がある場合は、 637 ページの『3270 端末からの読み取り』を参照してください。伝送されるデータは、オペレーターが伝送を行うために何をしたかによって変わります。

- ENTER キーまたは PF キー
- CLEAR、CNCL または PA キー (「短縮読み取り」キー)
- フィールド選択 (カーソル選択、ライト・ペン検出、またはトリガー・フィールド)

必要であれば、いずれの方法が行われたかを知ることができます。 824 ページの『アテンション ID の使用』にこの説明があります。また 3270 入力操作の詳細は、 635 ページの『3270 端末からの入力』にも記載されています。

短縮読み取りキーは、アテンション ID (キーそのものの識別) のみを伝送します。フィールド・データはまったく伝送されず、マップするものも何もありません。この理由から、 829 ページの『MAPFAIL およびその他の例外状態』で説明されているように、短縮読み取りキーが原因で MAPFAIL 状態が発生する場合があります。フィールド選択機能はフィールド・データを伝送しますが、ほとんどの場合 ENTER キーおよび PF キーの場合と同一のデータではありません。これについては以下の段落で述べます。これらの機能の使用を計画している場合、 877 ページの『第 71 章 特殊ハードウェアのサポート』の例外事項を参照してください。

ほとんどのアプリケーションは、ENTER キーまたは PF キーによって伝送を行うように設計されています。これらのうちいずれか 1 つが伝送に使用されると、画面上の修正されたフィールドがすべて、しかもそれらのフィールドのみ が伝送されます。

データの変更

627 ページの『変更』 に説明されているように、3270 画面フィールドは、フィールド属性バイト内のビットの 1 つである「修正データ・タグ」(MDT) がオンである場合にのみ変更されたとみなされます。オペレーターが、データの入力、既にあるデータの変更、または消去などでフィールドを変更した場合に、端末のハードウェアはこのビットをオンにします。マップを送信するときに、フィールドのための ATTRB 値の中に MDT を組み込むことにより、このビットをプログラムでオンにすることもできます。オペレーターが変更しない場合でも、特定のフィールドのデータを返すようにしたいときにこれを行います。

特定のマップ・フィールドからの入力があったかどうかを、対応する長さ (L) サブフィールドを見ることによって確認できます。長さがゼロであれば、そのフィールドから読み取られたデータはありません。BMS は入力マッピング操作を実行する前に入力構造全体をヌル (X'00') に設定するため、関連した入力 (I) サブフィールドにはすべてヌルが収容されています。修正データ・タグがオフ (つまり、タグがオフの状態でフィールドが送信され、オペレーターがそれを変更しなかった) の場合、またはオペレーターがフィールドを消去した場合にも、長さはゼロになります。注意してフラグ (F) サブフィールドを検査すると、この 2 つの状況を区別できます。このフィールドがヌルを含むが MDT がオンである (つまり、オペレーターが消去することによりフィールドを変更した) 場合は、フラグ・サブフィールドの高位ビットがオンになっています。フラグ・サブフィールドについて詳しくは、825 ページの『カーソルの検出』を参照してください。

長さがゼロ以外の場合、データはフィールドから読み取られています。オペレーターが何かを入力したか、既にあったものを変更したか、またはフィールドが MDT がオンの状態で送信されたかのいずれかです。データそのものを、対応する入力 (I) サブフィールドで検出することがあります。長さサブフィールドでは、文字がいくつ送信されたかを知ることができます。3270 端末は非ヌル文字のみを送信するため、BMS はどのくらいの量のデータがフィールドに入力されたかを認識します。フィールド定義の JUSTIFY オプション以外を指定しない限り、文字フィールドは右方をブランクで埋められ、数値フィールドは左方をゼロで埋められます。それが ATTRB=NUM のある数値であると明示しない限り、BMS は、フィールドが文字データを含むと想定します。

大文字変換

CICS では、小文字の入力文字が自動的に大文字に変換される状況がいくつかあります。端末定義とトランザクションはともに、変換を行うかどうかを決定します。

これらの指定の相互の影響については、「*CICS Resource Definition Guide*」に記載されているPROFILE および TYPETERMリソース定義の UCTRAN オプションを参照してください。

RECEIVE MAP コマンドで ASIS オプションを使用することにより、この変換を抑制できますが、端末入力が開始するタスクでの最初の RECEIVE は*例外* です (最初の RECEIVE は RECEIVE MAP (FROM なし) または端末制御 RECEIVE のいずれかであることがあります)。CICS は既にこの入力を読み取って変換しているため、変換を抑制するには遅すぎるためです (このコマンドの到着によってタスクが呼び出されます。 244 ページの『タスクの開始方法』に説明があります)。結果として、ASIS は疑似会話型トランザクション・シーケンスで完全に無視され、このシーケンスでは定義によって、せいぜい 1 つの RECEIVE MAP (FROM なし) がタスクごとに発生します。同じ理由から、ASIS は FROM オプションと一緒には使用できません (830 ページの『その他の入力の形式設定』を参照)。

アテンション ID の使用

この情報は多くのアプリケーションにおいて入力の一部でもあるので、入力を正しく解釈するために必要になる場合もあります。

例えば、「クイック・アップデート」トランザクションでは事務担当者がトランザクションを終了できるための何らかの方法が必要ですが、これについてまだ触れていませんでした。PF12 を押すと、トランザクションの制御を終了するという規則を設定することにします。RECEIVE MAP コマンドの後に次のようにコーディングします。

```
IF EIBAID = DFHPF12,
    EXEC CICS SEND CONTROL FREEKB ERASE END-EXEC
    EXEC CICS RETURN END-EXEC.
```

これにより、次に何を実行するか指定されなくてもトランザクションが終了するため、オペレーターは制御を取り返します。RETURN に先行する SEND CONTROL コマンドがキーボードをアンロックして画面を消去するため、オペレーターは次の要求を入力できます。

いろいろなアテンション・キーに対応する 16 進値は、DFHAID というコピーブックに定義されています。これらの定義を使用するためには、DFHAID を作業用ストレージにコピーします。これは、事前定義された属性バイトの組み合わせを使用するために DFHBMSCA をコピーするのと同じ方法です (806 ページの『属性値の定義: DFHBMSCA』を参照)。DFHAID サンプル集の内容は、「*CICS アプリケーション・プログラミング・リファレンス*」にリストされています。

HANDLE AID コマンドの使用

HANDLE AID コマンドを使用して、使用されているアテンション・キーを識別することもできます (ただし、HANDLE AID コマンドをサポートしていない、C または C++ で作成する場合を除きます)。HANDLE AID は他の HANDLE コマンドと同様に作用します。つまり、それが適用される最初の RECEIVE コマンドの前に出すと、HANDLE AID で命名されたキーが使用されている場合、プログラムは後続の RECEIVE の終わりにブランチします。

例えば今説明した「エスケープ」コードに代わる方法は次のようになります。

```
EXEC CICS HANDLE AID PF12(ESCAPE) END-EXEC.
...
EXEC CICS RECEIVE MAP('QUPMAP') MAPSET('QUPSET') ...
```

```
    ...
ESCAPE.
    EXEC CICS SEND CONTROL FREEKB ERASE END-EXEC
    EXEC CICS RETURN END-EXEC.
```

HANDLE AID は同一プログラムでは RECEIVE コマンドにのみ適用されます。キーの指定は、別の HANDLE AID が同一プログラムでそのキーに新しいラベルを指定することによって指定を替えるか、またはキーをラベルなしで指定することによって終了させるまでは有効です。RECEIVE コマンドの RESP、RESP2、または NOHANDLE オプションは、その特定コマンドを HANDLE AID 指定の影響を受けないようにしますが、それ以外では有効です。

入力操作の間に受信する AID に対して HANDLE をアクティブにすると、いかなる例外条件が発生しても、また HANDLE CONDITION がその例外条件に対してアクティブであるかないかにかかわりなく、制御は HANDLE AID で指定されたラベルに移動します。このため HANDLE AID は、HANDLE CONDITION を使用して検査する場合、例外条件をマスクできます。この理由から、AID または例外条件、またはその両方に対して代替テストを行った方がいいかもしれません。AID のために EIBAID を検査して RESP オプションを使用するか、または例外条件のために EIBRESP を検査するといいでしょう。 829 ページの『MAPFAIL およびその他の例外状態』に示されているように、ここでは特に MAPFAIL に注意する必要があります。

カーソルの検出

アプリケーションによっては、オペレーターが送信時にカーソルを置いた位置を知る必要があります。これには 2 つの検出方法があります。マップで CURSLOC=YES が指定されている場合、BMS は、カーソルが置かれたマップ・フィールドのフラグ・サブフィールドにある 7 つ目 (X'02') のビットをオンにします。当然ですが、これはカーソルが名前を割り当てたマップ・フィールドに置かれている場合のみ作動します。

また、フラグ・サブフィールドはカーソルの存在とフィールド消去の両方の指示に使用されるため、特定の 1 つを確認する場合は各ビットを個別にテストする必要があります。X'80' ビットはフィールド消去用であり、X'02' ビットはカーソル用です。ビットのテストに不向きな言語を使用する場合は、組み合わせをテストすることができます。X'80' または X'82' の値は消去をシグナル通知し、X'02' または X'82' のいずれかはカーソルを示します。「*CICS アプリケーション・プログラミング・リファレンス*」に記載されている DFHBMSCA 定義には、これらのすべての組み合わせが含まれています。

EIB にある EIBCPOSN フィールドからもカーソルの位置を判別することができます。これは画面上での絶対位置で、上方左 (ゼロの位置) から数え、行にまたがって進みます。このため、40 文字分の幅がある画面上の 41 の値では、カーソルは 2 番目の行の 2 番目の桁に置かれます。この方法は、プログラムが画面上のフィールド配置、および端末タイプに依存するようになるため、可能な限り使用しないでください。

マップ入力の処理

入力サブフィールドの使い方を説明するため、再び「クイック・アップデート」の例に戻ります。入力を入手したら、継続する前に検査が必要です。初めに、請求金額が入力された (入力の長さがゼロより大きくなった) こと、また正数で数値になっていることを確認します。

```
IF CHGL = 0, MOVE -1 TO CHGL
    MOVE 1 TO ERR-NO
ELSE IF CHGI NOT > ZERO OR CHGI NOT NUMERIC,
        MOVE DFHUNIMD TO CHGA,
        MOVE -1 TO CHGL
        MOVE 2 TO ERR-NO.
```

ここで 'MOVE -1' ステートメント、およびそれ以下のステートメントは、813 ページの『カーソルの位置決め』で説明されているとおり、マップを再表示したときにカーソルを最初の誤ったフィールドに置いています。メッセージ番号を見ると、メッセージ領域にどのメッセージが示されているかを知ることができます。例えば 1 は「請求金額を入力してください。」というように続き、最後の 6 は「請求金額は限度オーバーです。」などとします。これらの検査をおおよその重要度の高い順に行い、最も基本的なエラーのメッセージを入手するようにします。検査の最後に ERR-NO がゼロであれば、問題のないことがわかります。

請求金額だけでなく、顧客番号も入力しなくてはなりません。取引番号があれば (請求金額の状態がどうであろうと)、その顧客の取引記録を取り出して、その取引が存在することを確認できます。

```
IF ACCTNOL = 0, MOVE -1 TO ACCTNOL
    MOVE 3 TO ERR-NO
ELSE EXEC CICS READ FILE (ACCT) INTO (ACCTFILE-RECORD)
                RIDFLD (ACCTNOI) UPDATE RESP(READRC) END-EXEC
    IF READRC = DFHRESP(NOTFOUND), MOVE 4 TO ERR-NO,
        MOVE DFHUNIMD TO ACCTNOA
        MOVE -1 TO ACCTNOL
    ELSE IF READRC NOT = DFHRESP(NORMAL) GO TO HARD-ERR-RTN.
```

ここまできたら、エラーが起こって進めなくなるまで検査を継続します。オペレーターが適切な取引番号 (問題のないもの) を入力したこと、また請求金額がその取引に対して多過ぎないことを確認する必要があります。

```
IF ERR-NO NOT > 2
    IF ACCTFILE-WARNCODE = 'S', MOVE DFHBMBRY TO MSGA
        MOVE 5 TO ERR-NO
        MOVE -1 TO ACCTNOL
        EXEC CICS LINK PROGRAM('NTFYCOPS') END-EXEC
    ELSE IF CHGI > ACCTFILE-CREDIT-LIM - ACCTFILE-UNPAID-BAL
                    - ACCTFILE-CUR-CHGS
            MOVE 6 TO ERR-NO
            MOVE -1 TO ACCTNOL.
IF ERR-NO NOT = 0 GO TO REJECT-INPUT.
```

入力エラーの取り扱い

上記の「クイック・アップデート」の例で説明したように、処理でオペレーターが入力する場合は常に誤ったデータが入力される可能性があるため、コーディングではこのような不測の事態に備える必要があります。入力が誤っているときにしなければならないことは、通常以下のとおりです。

- オペレーターにエラーを通知する。すべてのエラーを同時に診断するようにします。それらを 1 つずつ提示するとオペレーターには面倒です。
- 既に入力されたデータを保管する。こうすることで、オペレーターは訂正以外のものを再入力せずにすみます。
- オペレーターが訂正した後、入力を再検査する

フラグのエラー

「クイック・アップデート」トランザクションの前のコードでは、エラーを記述するためにメッセージ・フィールドを使用しました (少なくとも最初のものはそうです)。フィールドすべてを誤って強調表示し (フィールドに強調表示するデータがあるという条件で)、また BMS がカーソルを誤ったフィールドの最初のものに置くように長さサブフィールドを -1 に設定しました。この情報を、同一マップを使用して次のように送信します。

```
REJECT-INPUT.
    MOVE LOW-VALUES TO ACCTNOO CHGO.
    EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') FROM(QUPMAPO)
              DATAONLY END-EXEC.
```

DATAONLY オプションを指定することに注意してください。これを行えるのは、このマップの定数部分がまだ画面上にあり、それを再書き込みする場所が画面にないためです。出力フィールドの ACCTNOO および CHGO を消去して受信した入力を送り返さないようにし、異なる属性の組み合わせを使用して ACCTNO フィールドを高輝度にしました (DFHBMBRY ではなく DFHUNIMD)。DFHUNIMD はそのフィールドを強調表示して修正データ・タグをオンのままにしておくため、オペレーターがフィールドを変更せずに再送すると、その顧客番号は再送されます。

正しい入力の保管

次のステップは、オペレーターが入力した正しいデータがすべて保管されるようにすることです。1 つの簡単な方法はデータを画面に保管することです。これには追加的に何かを行う必要は一切ありません。オペレーターが最初にフィールドに書き込んだときにそのフィールドの MDT はオンになりますが、一度そうなると、何回画面を読み取ってもオンのままです。画面を消去するか、SEND で FRSET オプションを使用して明示的にタグをオフにするか、または属性サブフィールドをタグをオフにする値に設定するまでは、タグはオフになりません。

画面でデータを保管する場合の欠点は、オペレーターが CLEAR キーを使用するとすべてのデータが失われることです。タスクが会話型の場合は、エラー情報を送信して訂正を要求する前に入力をプログラムの安全な領域に移動することによってこの危険を回避できます。疑似会話型シーケンスではコンポーネント・タスクが端末との対話をスパンしないため、エラーを検出して、訂正済み入力を処理するタスクへ古い入力を渡すことにより、タスクをこのような危険から保護します。タスクを終了する RETURN コマンドの COMMAREA を介して、一時記憶域に書き込むことによりデータを受け渡しできます。他にもいくつか方法があります (他の方法については 281 ページの『第 19 章 トランザクション間のデータの共用』を参照)。

プログラム内または一時記憶キューにデータを保管することにより、CLEAR キーの問題が回避されるだけでなく、インバウンド伝送時間が短縮されます。その理由は、エラー訂正サイクルで変更されたフィールドのみが伝送されるからです (エラ

一情報を送信するときは、既に送られていてかつ訂正されていないフィールドが、再度入らないようにするために FRSET を指定しなくてはなりません)。フィールド監査の繰り返しを避けることもできます。最初の実行時の後は、ユーザーがフィールドまたはそれに関連するものを変更した場合のみ監査すればいいためです。

しかし、これらの利点は余分なプログラミングと複雑化を招き、回線時間または監査パス長によくない影響を与え、エラーの起こる可能性も高くなるので、この方法の選択には十分な検討が必要です。新しい入力と古い入力とを組み合わせるためにコードを追加しなくてはなりません。また、MDT をオフにしていた場合、オペレーターがマップ・フィールドを修正したかどうかを判別するために、長さとフラグ・サブフィールドの両方を検査する必要があります。新規データの入ったフィールドは長さがゼロ以外になっています。また、データがあったがその後消去されたフィールドは、フラグ・サブフィールドで高位ビットがオンになっています。

妥協案として、データを両方の方法で保管するのがいいでしょう。オペレーターが画面を消去したら保管データを使用して最新表示にし、それ以外では画面から得られるデータを使用します。組み合わせ論理はまったく必要ありませんが、オペレーターを意図しない CLEAR による時間のロスから保護します。

「クイック・アップデート」のコードでは監査および伝送は最小限であり、「何もしない」アプローチを選択して情報を画面に保管します。

再検査

最後に必要なのは、入力データが再検査されるようにすることです。タスクが会話型であれば、訂正済み入力を受信 (必要であればさらに組み合わせ) した後で、コードの監査セクションを繰り返すという意味です。疑似会話型シーケンスでは、通常失敗したトランザクションを繰り返します。この例では、訂正済みデータが新規データと区別できないような方法でデータを画面に保管したため、必要なのは訂正済みデータに対して再度同じトランザクションを実行するための配置をすることだけです。次のようになります。

```
EXEC CICS RETURN TRANSID('QUPD') END-EXEC.
```

ここで、'QUPD' は「クイック・アップデート」トランザクションの ID です。

マップ入力後のマップ出力の送信

トランザクションが入力監査を行い、それに付随する障害もない場合は、マップ入力に対する特定の処理が完了します。多くの場合次のステップは、トランザクション出力を作成して送信することです。一般には、出力がマップされる場合は 801 ページの『第 62 章 BMS マップ出力の送信』に概説したステップに従ってください。ただし、マップ作成のためのストレージの獲得は、既に行った入力マッピングにより影響を受けることがあります。入出力のマップがそれぞれ異なるが、同一のマップ・セット、または互いにオーバーレイするように定義されたマップ・セット内にある場合は、入力マッピング処理の間に既にストレージ獲得が行われています。入出力のマップが互いにオーバーレイする場合は、出力マップ作成を始める前に、必ず必要なマップ入力をすべて保管し、出力構造をヌルに設定するようにしてください。これがうまくいかない場合は、マップが互いにオーバーレイしないように定義することも可能です。(この点に関する選択については、802 ページの『BASE および STORAGE オプション』を参照してください)。

トランザクションで、入力用と同じマップを出力用に使用するために呼び出すこともできます。これは既に見たように入力エラーを処理するコードにあるルーチンであり、「クイック・アップデート」のような簡単なトランザクションにもあります。1 画面のデータ入力トランザクションが、もう一つの一般的な例です。

マップを使用して新規データを画面に既に送信しているときに、DATAONLY オプションを用いて伝送を短縮できます。SEND CONTROL コマンドのみの使用でもいいでしょう。これらのオプションの説明は、 808 ページの『シンボリック・マップと物理マップの組み合わせ』を参照してください。

しかし「クイック・アップデート」トランザクションでは、メッセージ・フィールドに「了解」応答を入力する必要があります (さらにファイルの請求金額を更新して、処理を終了します)。

```
MOVE 'CURRENT CHARGE OKAY; ENTER NEXT' TO MSGO
ADD CHGI TO ACCTFILE-CUR-CHGS
EXEC CICS REWRITE FILE('ACCT') FROM (ACCTFILE-RECORD)....
```

また、入力フィールドを消去して、画面に次の入力ができるようにする必要があります。これを、画面上 (ERASEAUP オプションがすべての無保護フィールドを消去する) および出力構造 (出力サブフィールドは入力サブフィールドをオーバーレイし、入力データがまだそこにあるため) の両方で行う必要があります。

```
MOVE LOW-VALUES TO ACCTNOO CHGO.
EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') FROM(QUPMAPO)
          DATAONLY ERASEAUP END-EXEC.
```

最後に、同一トランザクションが次の入力用に実行されるように指定して、CICS に制御を返すことができます。

```
EXEC CICS RETURN TRANSID('QUPD') END-EXEC.
```

MAPFAIL およびその他の例外状態

RECEIVE コマンドで発生する可能性のある例外状態は、すべて 「*CICS アプリケーション・プログラミング・リファレンス*」で説明されており、そのほとんどは自己説明型です。しかし、このうちの 1 つについては、オペレーターの単純なエラーから起こることがあるため説明が必要です。これが MAPFAIL であり、使用可能なデータが端末から送信されていない場合、または送信されたデータが不定形式である (3270 の基準による、 639 ページの『不定形式モード』 を参照) 場合に発生します。MAPFAIL は、オペレーターが CLEAR キーまたは PA キーのいずれか 1 つを使用した場合に RECEIVE MAP で起こります。また、オペレーターが次のような画面から ENTER キーまたは PF キーを使用する場合にも起こります。

- マップに定義されたどのフィールドも修正データ・タグがオンになっていない (つまり、オペレーターが何も入力せず、既に設定されたタグとともに送ったフィールドがないということで、読み取りに返されるデータはありません)。かつ、
- カーソルが、マップに定義され命名されたフィールドにないか、またはマップが CURSLOC=YES と指定しなかった。

早まって ENTER を押すか、または偶然「短縮読み取り」キーを押すということは、オペレーターが犯しやすいミスです。ユーザーへの親切という点では、MAPFAIL の後はエラーのトランザクションを終了するのではなく、画面を最新表示にするのがいいでしょう。

MAPFAIL はまた、現行または直前のタスクで、SEND MAP または同等のものを用いて最初に形式設定せずに RECEIVE MAP を出すと起こります。また、送信したのと異なるマップを使用すると起こることがあります。論理のエラーが通知される場合もありますし、トランザクションが始動段階にあるということもあります。例えば「クイック・アップデート」の例では、開始するための準備を行っていません。つまり、オペレーターがトランザクションの使用を開始できるようにするため、画面に空のマップを作成していません。これを行うために別のトランザクションを使用することもできますが、MAPFAIL の後で画面を最新表示にするために必要なコードを利用してもいいでしょう。必要なコードは次のとおりです。

```
IF RCV-RC = DFHRESP(MAPFAIL)
    MOVE 'PRESS PF12 TO QUIT THIS TRANSACTION' TO MSGO
    EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET')
                FROM(QUPMAPO) END-EXEC.
```

オペレーターにエスケープの仕方を示します。これを行おうとして、最初の段階で MAPFAIL を起こしたのかもしれないためです。このメッセージを送信したくない場合、またはそれがマップのデフォルトである場合は、MAPONLY オプションを使用することができます。

```
EXEC CICS SEND MAP('QUPMAP') MAPSET('QUPSET') MAPONLY END-EXEC.
```

MAPFAIL が起こると、他の場合と同様入力マップ構造はヌルに設定されないため、プログラム論理がこの設定に依存する場合はこの条件のテストが重要です。

他の例外条件と同様、HANDLE CONDITION コマンドを出して MAPFAIL を代行受信できます。ただしこの場合 HANDLE AID も、受信する AID のためにアクティブにしてあると、制御は AID のために指定されたラベルに移り、MAPFAIL のラベルには行きません。824 ページの『HANDLE AID コマンドの使用』に説明があります。この状況では、そのために HANDLE を出している場合でも、EIBRESP もテストしない限り MAPFAIL に気がつきません。

EOC 状態

EOC も、BMS を使用していてよく出会う条件です。この条件は、チェーンの終わり (EOC) 標識が VTAM から返された要求/応答単位に設定されている場合に生じます。EOC はエラーを明示せず、BMS のデフォルト・アクションではこの条件は無視されます。

その他の入力の形式設定

RECEIVE MAP コマンドで形式設定するデータは、通常端末から送られてきますが、端末からではないデータ、または間接的にきたデータを形式設定することもできます。例えば、入力を受信してその一部を検査するまではどのマップを使用するかわからないとします。これは、区分化または論理装置コードのような特殊ハードウェア機能を使用する場合、またある種の論理状況において、起こることがあります。また、形式設定画面から中間プロセス (マッピングなし) により読み取られ、その後トランザクションに渡されたデータを形式設定する必要も考えられます。

RECEIVE MAP コマンドの FROM オプションは、これらの状況を示します。FROM は BMS に、データが既に読まれていること、また元の入力ストリームから入力マップ構造への変換が必要であることを通知します。

入力が既に読まれているため、FROM を使用する場合はその長さを指定する必要があります。BMS は通常行うようにこの情報をアクセス方式から入手できないためです。データがもともと別のタスクの RECEIVE コマンドからきたものであれば、RECEIVE MAP FROM コマンドの長さは元の RECEIVE によって生成された長さでなくてはなりません。

同じ理由から、FROM を使用するときは ASIS オプションを用いて大文字への変換を抑制することはできません。さらに、BMS は RECEIVE FROM の後で EIBAID および EIBCURSR を設定しません。

最後に、BMS は入力がどの装置からのものであるかを認識せず、それが現行のプリンシパル装置だったと想定します (プリンシパル装置なしでは RECEIVE FROM の使用さえもできません。たとえ入出力が起こらない場合でも同じです)。タイプの異なる装置からデータが送られた場合、同様のプリンシパル装置を使用してトランザクションでマッピングを行い、入力データ・ストリームの変換が正しく行われるようにする必要があります。

注: 入力データは不定形式であるため (3270 の基準による)、端末制御 RECEIVE で BUFFER オプションを用いて読まれたデータはマップできません。このような入力を RECEIVE MAP FROM で試みると MAPFAIL が起こります。

第 65 章 BMS の論理メッセージ

後処置オプションは SEND MAP コマンドと出力のページとの間の対応に影響しません。完全 BMS の 2 番目の機能である ACCUM オプションも使用しない限り、各 SEND MAP コマンドに対して 1 ページを入手します。ACCUM を使用すると、複数のマップを使用してばらばらにページを作成することができ、また PAGING 同様、メッセージを 2 ページ以上にすることもできます。改ページや、特定のページまたは画面の容量に出力を調整することについて懸念する必要はありません。BMS はこれらを自動的に処理し、必要な場合には改ページを制御することができます。累積ページ作成の詳細は、839 ページの『第 66 章 累積出力 — ACCUM オプション』にあります。

2 ページ以上の出力メッセージ、または複数の異なるマップで構成される単一ページを作成するとすぐ、BMS 呼び出しの**累積**マッピングに影響が出ます。PAGING は複数ページを暗黙指定し、ACCUM は複数ページと複合ページの両方を暗黙指定します。そのためこれらのオプションはいずれも最初の表示では、BMS は累積マッピング・モードになり**論理メッセージ**を開始します。 SEND コマンドとメッセージとの間の 1 対 1 対応は終了し、後続の SEND MAPS は現行論理メッセージに追加されます。メッセージ内の個別ページはまだ、完了するとすぐ処理されますが、これらはすべて同一の論理メッセージに属し、BMS に終了を指示するまで継続されます。

論理メッセージの作成

論理メッセージを開始するときは、いくつかの規則に従う必要があります。

- 一度に作成できる論理メッセージは 1 つだけです。このメッセージをルーティングする場合、BMS は複数の論理メッセージを内部に作成することがありますが、内容については 1 つだけです。メッセージを完了して処理した後に、同じタスクで別のメッセージを作成できます。必要であれば異なるオプションも使用できます。
- メッセージ管理に関連するオプションは、このメッセージを作成するすべてのコマンドで同一でなくてはなりません。そのオプションは、以下のとおりです。
  - 後処理オプション: PAGING、TERMINAL、または SET
  - ページ構成を管理するオプション: ACCUM はすべてのコマンドにあるか、またはすべてのコマンドにないようにしてください。
  - CICS 一時記憶域にあるメッセージ用の ID: REQID オプション値

  メッセージの途中でオプションを切り替えると、INVREQ 条件になります。また REQID の場合は IGREQID 条件になります。
- ERASE、ERASEAUP、NLEOM、および FORMFEED の各オプションは、そのページにかかわる BMS コマンドのいずれか に使用されている場合、優先して適用されます。
- ページに対する CURSOR、ACTPARTN、および MSR の各オプションの値は、最新の SEND MAP コマンドに指定されていればそこから取られ、指定されていなければマップから取られます。

© Copyright IBM Corp. 1989, 2009

- 最新の SEND MAP コマンドからの 3270 書き込み制御文字 (WCC) が使用されます。WCC は、コマンドの
  ALARM、FREEKB、PRINT、FRSET、L40、L64、L80、および HONEOM の各オプションから、これらのいずれか が指定されるたびにアセンブルされます。それ以外の場合、マップの同一オプションから作成されます。このコマンドからのオプションは、マップのオプションと混合されることはありません。
- メッセージ作成に使用するすべてのコマンドからの FMHPARM が組み込まれます。
- 上に述べたオプションが対立しない限り、論理メッセージを作成するために SEND MAP コマンドと SEND CONTROL コマンドの両方を使用できます。また、SEND TEXT コマンドと SEND CONTROL コマンドの組み合わせを使用して論理メッセージを作成することもできます (SEND TEXT はテキスト出力の形式設定のための SEND MAP の代替です。 847 ページの『SEND TEXT コマンド』に説明があります)。ただし、区分画面または論理装置コードを使用しない限り、SEND MAP と SEND TEXT とを同一メッセージで混合して使用することはできません。これらについては 867 ページの『第 70 章 区分画面サポート』および 877 ページの『論理デバイス・コンポーネント』にそれぞれ説明があります。

  また、SEND TEXT には結合マッピングおよびテキスト出力ができる 2 つの特殊な形式がありますが、これには違った制約事項が適用されます。詳しくは、 851 ページの『SEND TEXT MAPPED および SEND TEXT NOEDIT』を参照してください。
- 論理メッセージを作成している間に端末と対話することもできます。ルーティングもしない限り BMS コマンドを使用して端末に書き込みを行うことはできませんが、BMS RECEIVE MAP コマンド、および端末制御の SEND コマンドと RECEIVE コマンドを使用できます。

SEND PAGE コマンド

論理メッセージを完了したら、SEND PAGE コマンドを使用して BMS に通知します。ACCUM オプションを使用した場合、SEND PAGE を使用すると BMS は現行ページを完了し、設定した後処置オプションに従ってそれ以前のページと同様に処理します。後処置オプションが TERMINAL の場合、最後のページはプリンシパル装置に書き込まれます。SET の場合はプログラムに返されます。PAGING の場合は一時記憶域に書き込まれます。後処置オプションが PAGING だった場合、BMS はメッセージ全体をプリンシパル装置に送達するための配置も行います。SEND PAGE コマンドのオプションは、これをどのように行うかを管理します。 835 ページの『RETAIN および RELEASE』 に説明があります。

また、SYNCPOINT コマンドまたはタスクの終了によっても論理メッセージは終了しますが、明示的にではなく暗黙的に終了します。この場合 BMS は、SYNCPOINT または RETURN の前に SEND PAGE を出したときと同様に作動しますが、ACCUM オプションを使用した場合出力の最後のページは失われます。結果として、SEND PAGE を明示的にコーディングする必要があります。

何かの理由でメッセージの送信を中止する場合、不完全な論理メッセージを削除することもできます。その場合は SEND PAGE の代わりに PURGE MESSAGE コマ

ンドを使用します。 PURGE MESSAGE により、BMS は現行の論理メッセージおよび関連する制御ブロックを削除します。この制御ブロックには、既に CICS 一時記憶域に書き込まれているすべてのページが含まれます。必要な場合は、同一タスクで他の論理メッセージを続けて作成できます。

RETAIN および RELEASE

PAGING の後処置を使用して論理メッセージを完了すると、BMS は論理メッセージ全体を送達するための配置を行います。これは一時記憶域に累積されています。ページの表示または印刷は、SEND PAGE コマンドの直後にインラインで行うことができますが、この目的のために別のタスクをスケジュールする方が一般的です。どちらの場合でも、CICS は必要なプログラムを提供します。これらのプログラムを使用すると、端末オペレーターはメッセージの表示、前後へのページ送り、特定ページの表示などを制御することができます。別のタスクを使用すると、トランザクション・コード CSPG を用いて疑似会話型で実行されます。表示がインラインの場合、この作業は (同じ CICS 提供プログラムによって) メッセージを作成したタスク内で行われ、結果として会話型になります。

SEND PAGE コマンドに RETAIN、RELEASE を指定するかまたはいずれも指定しないことによって、メッセージをいつどのように送信するかを示します。最も一般的な選択で、デフォルトでもあるのはいずれも指定しないことです。こうすると CICS は CICS 提供トランザクション CSPG をスケジュールしてメッセージを表示し、次に制御をタスクに返します。CSPG トランザクションは、その端末で実行を待機している他のすべてのトランザクションとともにキューに入れられ、端末が解放されると優先順位の高い順に実行されます。通常の場合待機するタスクが他にないため、CSPG は作成するタスクが終了するとすぐに実行します。

注: 端末は、CICS の自動トランザクション開始に CSPG を自動開始する許可を与えるときに定義されなくてはなりません (関連した TYPETERM 定義で ATI(YES) とする)。そうでない場合、オペレーターはトランザクション・コード CSPG またはページング・コマンドのいずれか 1 つを入力してこの処理を開始しなくてはなりません (RELEASE または RETAIN のいずれも指定されていない場合)。

RELEASE オプションも同様に作動しますが、タスクは SEND PAGE RELEASE の後に制御を取り返しません。その代わり BMS は、メッセージの最初のページを端末に即時に送信します。次に、CICS RETURN が最高レベルのプログラムで実行されたときと同様にタスクを終了し、オペレーターが残りのページを表示できるように、端末で CSPG トランザクションを開始します。CSPG コードは、RELEASE または RETAIN のいずれも指定しない場合と同様に疑似会話型で実行しますが、元のタスクはそれが先であれば疑似会話型のままです。

RELEASE を指定する場合と、いずれのオプションも指定しない場合では他に 2 つの違いがあります。

- RELEASE を指定すると、オペレーターが CSPG を使用してメッセージの表示を終えた後で、端末から次の入力のためのトランザクション ID を指定できます。
- RELEASE はまた、端末オペレーターが複数のトランザクションからの出力をチェーニングすることも可能にします (836 ページの『端末オペレーターのページング: CSPG トランザクション』 を参照)。

SEND PAGE RETAIN を使用すると BMS はメッセージを即時に送信します。このプロセスが完了すると、SEND PAGE コマンドの後でタスクは即時に制御を再開します。端末がディスプレイの場合、BMS は CSPG トランザクションと同一の、メッセージ全体のページングのためのオペレーター機能を提供しますが、これはタスクの枠内に限られます。BMS がこの目的のために使用するコードは、RECEIVE コマンドを出してオペレーターの表示要求を入手します。またこれにより、タスクは会話型になります。

注: SEND PAGE コマンド処理の間にエラーが起こった場合は、論理メッセージが不完全であると認識され、表示するための試みはされません。BMS は、エラーの後で制御を取り戻すようにしない限り、このメッセージをクリーンアップ処理で廃棄します。その場合、PURGE コマンドを使用して論理メッセージを削除するか、または SEND PAGE を再試行することができます。障害を起こしたエラーが修正されるまでは、再試行しないでください。

AUTOPAGE オプション

SEND PAGE コマンドはまた、ページを端末に送達する方法も BMS に指示します。ディスプレイ端末では、端末オペレーターの要求に基づき、CSPG が一度に 1 ページ送信するようにしたいとします。プリンターでは、ページが続いて送信されるようにしたいとします。SEND PAGE コマンドの AUTOPAGE オプションまたは NOAUTOPAGE オプションを使用してこれを制御します。NOAUTOPAGE では、端末オペレーターがページの表示を制御できます。AUTOPAGE は、装置が受け入れ可能な限り速くページを昇順で送信します。いずれも指定しない場合、BMS はどちらが適切か端末定義から判別します。

注: プリンシパル装置がプリンターの場合、PAGING ではなく TERMINAL の後処置を使用できる場合があります。プリンターへの連続的な送信は、ディスプレイの場合のように互いにオーバーレイすることはないためです。TERMINAL ではオーバーヘッドがより少なく (ACCUM も必要でない場合は特に)、論理メッセージの作成はしないでください。

端末オペレーターのページング: CSPG トランザクション

CICS 提供のページング・トランザクションである CSPG を使用すると、端末のユーザーはページ検索要求を入力することにより、論理メッセージの個別のページを表示することができます。システムのスタッフは、CSPG がサポートする検索およびその他の要求のためのトランザクション ID を、システム初期設定テーブルに定義します。プログラム・ファンクション・キーにより、オペレーターの労力が最小限に抑えられる場合もあります。

検索は、順番に (次のページまたは前のページ)、またはランダムに (特定のページ、最初のページ、最後のページ) 行うことができます。ページ検索に加えて、CSPG は以下の要求をサポートします。

ページ・コピー
現在表示されているページを別の端末にコピーする。ターゲットの端末のページ・サイズまたは形式制御特性が異なる場合、BMS はそのページの形式を再設定します (その端末が BMS によってサポートされるタイプである場合)。

メッセージ照会
CSPG を使って、端末で表示を待っているメッセージをリストする。このリストには BMS が割り当てるメッセージ識別、および、ルーティング・メッセージについては、送信側がメッセージの表題を提供していればその表題が含まれます。

メッセージの除去
論理メッセージを除去する。

ページ・チェーン
メッセージの表示を開始した後で現行の CSPG トランザクションを中断して、1 つ以上の他のトランザクションを実行し、次に元の CSPG 表示を再開する。間に実行されるトランザクションは、それ自体が BMS または端末出力を生成することがあります。この出力が、PAGING オプションおよび RELEASE オプションまたは RETAIN オプションを使用して作成される BMS 論理メッセージである場合、このメッセージは元のメッセージに「チェーニング」されるため、オペレーターはこのメッセージともう一方のメッセージを切り替えられます。

自動ページへの切り替え
NOAUTOPAGE 表示モードから AUTOPAGE モードに切り替える。キーボードとハードコピー出力とを結合する端末の場合、これによりオペレーターは特定ページの検査に基づき、メッセージの除去または印刷することができます。

ページ検査のプロセスは、オペレーターがメッセージが除去可能であることを通知するまで継続されます。CSPG では、上で述べたようにこの目的のための特別な要求が提供されています。SEND PAGE コマンドに OPERPURGE というオプションが含まれていた場合、この要求はメッセージを削除して CSPG から制御を取り戻す唯一の方法です。

しかし OPERPURGE がない場合、CSPG 要求ではない端末からの入力はすべて、メッセージ削除要求として解釈され、CSPG は終了します。メッセージが RETAIN オプションで表示されていた場合、その表示を終了する CSPG 以外の入力は、タスクが実行を再開するとき BMS または端末制御 RECEIVE でのアクセスが可能です。CSPG トランザクションの詳細については、「*CICS Supplied Transactions*」の CSPG - ページの取得を参照してください。

論理メッセージのリカバリー

PAGING の後処置で作成された論理メッセージは、作成と送達との間 CICS 一時記憶域に保持されます。BMS は、SEND コマンドで、2 文字の REQID からメッセージの一時記憶キュー名を構成します。この 2 文字の後には、一意性を維持するための 6 つの位置番号が続きます。REQID を指定しない場合、BMS は 2 つのアスタリスク (**) を値として使用します。

一時記憶域はリカバリー可能リソースにすることができるため、PAGING の後処置で作成された論理メッセージは CICS が異常終了した後もリカバリーすることができます。実際 CICS では一時記憶域がリカバリー可能であることは総称キュー名に基づいているため、メッセージに REQID を選択することにより、メッセージのい

くつかはリカバリー可能、その他のものはリカバリー不能にすることができます。論理メッセージがリカバリー可能になる条件については、「*CICS Recovery and Restart Guide*」に記載されています。

ルーティング・メッセージも、プリンシパル装置のために作成されたメッセージと同様リカバリー可能にすることができます。ルーティングについての説明は、853 ページの『第 68 章 メッセージ・ルーティング』を参照してください。

第 66 章 累積出力 — ACCUM オプション

ACCUM オプションを使用すると、任意の数の SEND MAP コマンドおよびページ・サイズ未満のマップから、出力を累積して作成することができます。このオプションを使用しないと、各 SEND MAP コマンドは 1 ページ (後処置が PAGING の場合)、またはメッセージ全体 (後処置が TERMINAL または SET の場合) に対応します。しかし ACCUM を使用すると、BMS は出力を形式設定しますが、出力がページに収まる以上に累積されるか、または論理メッセージが終了されるまでは処理をしません。必要であれば、改ページで代行受信するか、または BMS がそれらを自動的に処理するようにできます。

ページ・サイズは端末定義の PAGESIZE 値または ALTPAGE 値によって判別されます。PAGESIZE は、トランザクションの実行に使用する PROFILE がデフォルト画面サイズを指定する場合に使用し、ALTPAGE は代替画面サイズが指示される場合に使用します (画面サイズと異なり、ページ・サイズは ERASE コマンドで組み込むことのできる DEFAULT オプションおよび ALTERNATE オプションから影響を受けません)。

浮動マップ: ACCUM を使用した BMS におけるマップの配置

784 ページの『マップの定義: DFHMDI』ページの例では、左上隅に行および桁の数を指定することにより、画面上またはページ上にマップを絶対的に配置する方法を説明しました。しかし、マップは**浮動させる**こともできます。つまりマップは、同一ページに既に書き込まれたマップ、およびページの端の任意の場所に関連させて位置を決めることができるのです。浮動マップは、複数の画面サイズをサポートする必要があるとき、またはページをヘッダー、明細行およびトレーラーからばらばらに作成する必要があり、明細行の数はデータによって異なるような場合に、プログラム論理を簡単にします。

これを行うことができる BMS オプションは次のとおりです。

- JUSTIFY
- HEADER および TRAILER
- LINE オプションおよび COLUMN オプションに関連する値 (NEXT および SAME)

ACCUM オプションを使用して複合画面を作成するとき、任意の特定マップの画面上の位置は次のものによって決定されます。

- 送信される時点で画面上に残っているスペース
- マップ定義での JUSTIFY、LINE および COLUMN の各オプションの値

ページ上に残るスペースも、次のものによって変わります。

- 既に現行ページに配置されたマップ。
- 改ページで発生する処理である「オーバーフロー処理」を行っているかどうか。行っている場合、マップ・セットのトレーラー・マップのサイズも要因となります。

© Copyright IBM Corp. 1989, 2009

これから述べる配置規則は、ACCUM を指定しない場合でも適用されますが、FIRST および LAST の JUSTIFY 値は無視されます。ただし、ACCUM を指定しないと各 SEND MAP は別々のページに対応するため、ページ上に残るスペースは常にそのページ全体になります。

改ページ: BMS のオーバーフロー処理

マップされた論理メッセージを作成するときは、改ページがくるとき、つまり直前に送信したマップが現行ページに適合しないときに BMS が通知するようにすることができます。これは、ACCUM を使用して複合ページを作成するときに非常に便利です。これを使用すると、トレーラー・マップを現行ページの最下部に、ヘッダー・マップを次のページの最上部に、それぞれ配置したり、ページに番号を付けたりすることができます。

BMS では次のいずれかがあてはまる場合、改ページでのプログラム制御が与えられます。

- HANDLE CONDITION コマンドを出して OVERFLOW 条件のためのラベルに名前を付けた場合。
- SEND MAP コマンドの RESP オプションまたは NOHANDLE オプションのいずれかを使用して NOFLUSH オプションを指定する場合。

これらのアクションのいずれも、2 つの影響を及ぼします。

- ページ上に残るスペースの計算結果が変更される。送信中のマップそのものがトレーラー・マップでない限り、BMS は、ユーザーが結局現行ページにスペースを残しておきたいという想定をします。このため、同一マップ・セットに最大のトレーラー・マップのためのスペースを確保します (最大トレーラー・マップは、ほとんどの行を持つ TRAILER オプションを含むマップです)。改ページを代行受信しない場合 (またはトレーラー・マップを送信する場合)、BMS はページの実際の終了部分を使用して現行マップが適合するかどうかを判別します。
- マップが現行ページに適合しない場合、制御のフローが変更される。この状況が検出されると、BMS は OVERFLOW (オーバーフロー) 条件を起こします。次に、オーバーフローを起こした SEND MAP コマンドを処理しないで タスクに制御を返します。制御は HANDLE CONDITION コマンドで名前指定された位置に行きます (使用していた場合)。NOFLUSH を指定すると、制御は通常どおり SEND MAP の後のステートメントに移動し、RESP 値または EIB の EIBRESP をテストして、オーバーフローが起こったかどうかを判別する必要があります。

オーバーフロー後にプログラムが制御を得るときは、次のことが必要です。

- 現行ページに表示したいトレーラー・マップをすべて追加する。BMS はそのための場所を、使用したばかりのマップ・セットにほとんどの行とともに残しておきます。これが確保する正しい行数でない場合、または複数のマップ・セットを使用する場合は、適用できる任意のマップ・セットにダミー・マップを組み込むことにより、正しい数にすることができます。ダミー・マップは TRAILER を指定しなくてはならず、また確保したい行数を含まなくてはなりません。ダミー・マップは、いずれの SEND MAP コマンドでも使用する必要はありません。
- 次のページの最上部に置きたいヘッダー・マップをすべて書き込む。
- 最初にオーバーフローを起こしたマップを再送信する。オーバーフロー発生時のデータおよびマップ名を把握していなければなりません。BMS ではこの情報を保

管しません。オーバーフローを起こす場合がある SEND MAP コマンドが複数ある場合、再発行に必要なコマンドを判別するために必要なプログラム・ロジックは、NOFLUSH より HANDLE CONDITION OVERFLOW を使用する方が複雑になるので注意してください。

OVERFLOW がオンになると、BMS は出力が現行ページに適合しないときにプログラムに制御を返すのを中断しますが、残りスペースの計算にはまだオーバーフロー規則を使用します。OVERFLOW は、BMS がヘッダーまたはトレーラーではないマップに命名する最初の SEND MAP を処理するまでオンのままです。これにより、OVERFLOW のための HANDLE CONDITION を使用不可にしたり、または応答コード・テストを変更したりせずにトレーラーおよびヘッダーを送信することができ、また定期出力に戻るとすぐにオーバーフロー・ロジックが回復されます (もともとオーバーフローを起こしたマップの再送信は通常、オーバーフロー条件をオフにするイベントです)。

オーバーフローを代行受信しない場合、BMS は改ページがくるときにプログラムに通知しません。代わりに、現行ページを、設定した後処置オプションに従って処理し、オーバーフローを起こしたマップの新しいページを開始します。

マップ配置の規則

画面上でのマップの基本的な配置は上部から下部に向かいます。スペースがある所ではマップを横並びに配置することができます (上部から下部までの全体のフローを保守する場合)。所定の SEND MAP ACCUM コマンドに対する正確な規則は以下のとおりです。

1. マップが開始する一番上の行は、以下のように決定されます。
   a. マップ定義に JUSTIFY=FIRST が含まれる場合、BMS は、既にページ上にあるマップのみが、オーバーフロー処理の間にそこに配置されたヘッダーでない限り、即時に (ステップ 5 (843 ページ) の) 新しいページに進みます。この場合、BMS はステップ 1c から続行されます。
   b. マップが JUSTIFY=LAST を指定する場合、BMS はページに適合できる最も低い行でマップを開始します。マップがトレーラー・マップであるか、ユーザー側でオーバーフローを代行受信していないか、あるいは、既にオーバーフロー処理を行っている場合、BMS はページ上のすべてのスペースを使用します。それ以外の場合、BMS は、一方で最大トレーラー・マップのためにまだ余裕を保ちながら、ページ上のできるだけ低い位置にマップを配置します。マップがこの開始行を使用して垂直に適合する場合、処理はステップ 3 (842 ページ) から続行されます (LINE オプションは JUSTIFY=LAST の場合には無視されます)。適合しない場合はオーバーフローが起こります (ステップ 5 (843 ページ))。

      注: JUSTIFY=BOTTOM は、ACCUM を使用する出力操作では JUSTIFY=LAST と同一です (ACCUM を指定しない場合や、入力マッピングの場合は異なります。 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください)。
   c. 垂直 JUSTIFY 値がない場合 (または JUSTIFY=FIRST によって起こったオーバーフロー処理が完了した後で)、LINE オペランドが検査されます。LINE に絶対値が指定されていれば、その行が使用されます (そのページに最後に配

置されたマップの開始行またはその下にある場合)。値がそのポイントの上にある場合、BMS はステップ 5 (843 ページ) の新しいページに進みます。

LINE=NEXT の場合、最初の完全未使用行 (現在ページにあるすべてのマップの下) が使用されます。LINE=SAME の場合、最後に送信されたマップの開始行が使用されます。

2. BMS は次に、試験的な開始行を使用して、マップが画面に垂直に適合することを検査します。ここでもまた、マップがトレーラー・マップであるか、ユーザー側でオーバーフローを代行受信していないか、あるいは既にオーバーフロー処理を行っている場合、BMS は残りのすべてのスペースを使用します。それ以外の場合 BMS では、マップが適合し、さらに最大トレーラー・マップのためのスペースが残されていることが必要です。マップが垂直に適合しない場合、BMS は新しいページを開始します (ステップ 5 (843 ページ))。
3. 次に、BMS はマップが水平に適合するかどうかを、試験的な開始行を想定して検査します。水平位置においては、LEFT および RIGHT の JUSTIFY オプション値が使用されます。LEFT はデフォルトで、COLUMN 値がマップの左側を示すという意味です。COLUMN に指定される数値はマップの左端が開始するべき場所を示し、ページの左側から数えます。COLUMN=NEXT は、開始行の左から数えて、最初の未使用桁からマップを開始します。 COLUMN=SAME は、最後に画面に配置され、JUSTIFY=LEFT も指定されており、ヘッダー・マップまたはトレーラー・マップではないマップの左端の桁を意味します。

   JUSTIFY=RIGHT は、COLUMN 値がマップの右端を示すという意味です。数値は、マップの右端の開始位置を示し、*右側から数えます*。COLUMN=NEXT は右側から数えて最初の使用可能な桁を意味し、COLUMN=SAME は、最後に配置され、JUSTIFY=RIGHT が指定されており、ヘッダー・マップまたはトレーラー・マップではないマップの右方の桁です。

   マップが水平に適合しない場合、マップが適合する行に到達するか、またはオーバーフローが起こるまで、BMS は開始行を下方に向かって一度に 1 行ずつ調整します。開始行を調整するたびに、垂直検査 (ステップ 2) から処理が再開されます。
4. マップが適合する場合、BMS は現行ページにそのマップを追加して使用可能スペースを更新します。ここでは次の規則が適用されます。
   - マップの最初の行の上にある行はすべて、まったく使用できません。
   - マップが JUSTIFY=LEFT を指定する場合、ページの左端からそのマップの右端の桁までの桁全体は、マップの最上部からページ上の何かがある最後の行までの行全体で使用不能です (このマップまたはそれ以前のものから)。
   - マップが JUSTIFY=RIGHT を指定する場合、ページの右端とマップの左端との間にある桁は、マップの最上部からページ上の何かがある最後の行までの行全体で使用不能です。

   843 ページの図 153 には、残りのスペースが各新規マップ配置によりどのように削減されるかが示してあります。

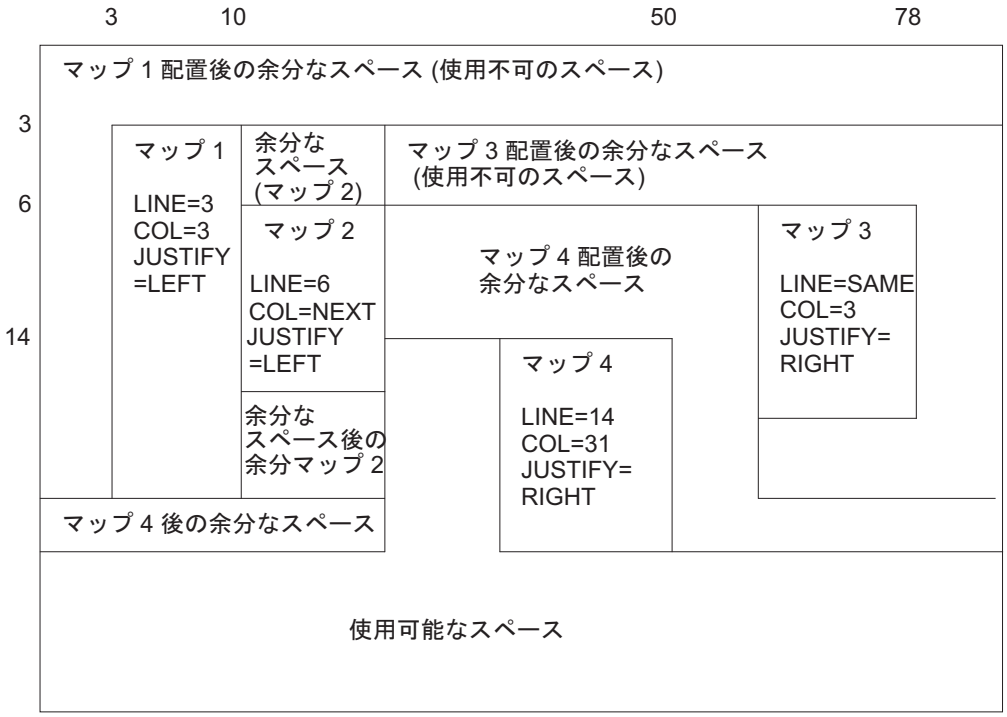


図 153. ページ上のマップの連続配置。それぞれ使用不能になるスペースを示す。

5. 現行マップがページに適合しないとき、BMS は制御をプログラムに戻すべきかどうかを判別します。オーバーフロー発生時に制御を要求して、まだオーバーフロー処理を開始していない場合には、 840 ページの『改ページ: BMS のオーバーフロー処理』で説明しているように BMS は制御を返します。それ以外の場合、BMS は、設定した後処置オプションに従って現行ページを処理し、新しいページを開始して、ステップ 1 (841 ページ) で適合しないマップの処理を再開します。

累積処理用の ASSIGN オプション

複合画面作成の複雑さを緩和する手段として、CICS には特に累積処理に関連する ASSIGN コマンド・オプションが提供されています。

- MAPCOLUMN
- MAPHEIGHT
- MAPLINE
- MAPWIDTH

これらはすべて、最後に送信されたマップに適用されます。MAPHEIGHT および MAPWIDTH はサイズ (行数および桁数) で、MAPLINE および MAPCOLUMN はマップの起点 (左上隅の位置) です。

複合画面からの入力

複数マップで作成された画面から、マップされた入力を読み取ることができますが、これには制限があります。まず、画面が複数のマップで書き込まれていたとしても、RECEIVE MAP コマンドに指定できるマップは 1 つだけです。

二つ目に、BMS は入力のための浮動マップの配置の仕方を認識できず、RECEIVE MAP コマンドのマップが空の画面に書き込まれたと想定します。このため NEXT または SAME の LINE 値または COLUMN 値は、入力では出力の場合と異なる解釈になります。JUSTIFY=LAST はまったく無視されます。マップを画面の最下部に配置して、そこから再度データを読み取りたい場合は、JUSTIFY=BOTTOM を使用してください。

パフォーマンスの考慮事項

エンド・ユーザーから見えるアプリケーションのパーツの全体的効率に関するコンポーネントには、プロセッサー・パス長、通信回線使用率、およびユーザー時間という 3 つのコンポーネントがあります。

オンライン・システムが発達したため、重点は、必要であれば他の要素を犠牲にしてでも、ユーザーにとってできるかぎり作業を簡単で迅速にすることに確実にシフトしてきました。またプロセッサーが以前より安価になったため、設計者はユーザー・インターフェースをより良くするためだけでなく、むしろプログラミングと保守の労力を軽減するために年月を費やしてきました。

推奨に従ってパス長さと回線時間を削減します。これらを削減するために余分なプログラミング労力を要したり、設計の一般性が低くなることが正当かどうか、またその程度を自分で判断する必要があります。

パス長の最小化

通常、単一の CICS コマンドで実行される命令の数は、それを呼び出したアプリケーション・プログラムでの命令の数より、比較的多くなります。結果として、所定のタスクのためのパス長は、通常他のものより CICS コマンドの数およびタイプにより強く依存するため、コマンドは調整に関して最も有効な領域です。コマンドは当然タイプにより異なりますので、所定のコマンドのいずれに対するパス長も、状況によってかなり変化することがあります。

BMS では、次のことをお勧めします。

- 実行するときは、単一のコマンドで画面 (ページ) を作成します。それほど多くない量の追加プログラミングで同じ機能を実行できるときは、ACCUM 機能を使用した複合画面の作成はしないでください。また、812 ページの『マップの外側』に示したような、複数の物理的な書き込みによる複合画面の作成は、特殊な状況以外はしないでください。
- ユーザーが検査するのに適当な量以上の出力を一度に生成しないでください。トランザクションの中には、一定の入力値に対して何ページもの出力を生成するものがあります (特に照会)。こうなると、ユーザーは通常、当初の出力ページ全体ではなく、検索を限定して問い合わせを繰り返します。見ることのできない出力を生成するパス長を避けるためには、ページ数を適当な数に制限し、ユーザーに最新ページで残りがまだあることを通知し、ユーザーがそれを要求する場合はそのポイントから検索を再始動するために必要な情報を保管することができます。追加のプログラミングは最小限ですみます。再始動データの保管方法については、281 ページの『第 19 章 トランザクション間のデータの共用』を参照してください。

- 使用可能であれば、BMS の「高速パス」にあるコマンドを使用してください。(適格なコマンドおよび端末タイプについては、 776 ページの『最小機能 BMS』を参照してください。)
- 非常に簡単な入出力については、端末制御コマンドを使用してください。その場合、BMS 形式設定またはその他の機能は必要ありません。パス長が重要な問題である場合、完全に端末管理を使用してもいいでしょう。ただし、柔軟性、初期プログラミング労力、および保守容易性という点から見た BMS の端末管理における利点は重要であり、通常はパス長が長くなることより重要です。

メッセージ長の削減

3270 ハードウェアの機能を利用して、インバウンドとアウトバウンドの両方のメッセージの長さを削減することができます。端末とプロセッサーとの間のいずれのリンクにおいても帯域幅が制約されている時、より短いメッセージで、全体としてよりすぐれた応答を得ることができます。ただし、所定の伝送に対する時間は、いずれの場合もその時リンクを使用する他のユーザーの動作によって変わるため、改善された点が直接にわかるということはないかもしれません。3270 データ・ストリームの長さを削減するための方法のいくつかを、以下に示します。

- 画面を送信するとき、不必要に MDT をオンにしないでください。関連した入力フィールドが入力に伝送されてしまいます。通常はこのタグをオンにする必要はありません。ユーザーが入力内容をフィールドに入力するとき、ハードウェアがこれを行うためです。プログラムによって明示的にオフにされない限り (FRSET、ERASEAUP、または ERASE を使用するか、またはタグがオフの指定変更属性バイトによって行う)、画面が送信される回数にかかわらず、タグはオンのままです。プログラムによってタグをオンにする必要がある場合とは、画面上のデータをユーザーが変更しないフィールドに格納したいとき、または誤ってあるフィールドを強調表示し、ユーザーがそれを変更するしないにかかわらずそのフィールドを返したいときだけです。この場合には、強調表示だけでなく MDT をオンにする必要があります。
- 画面の入力内容を再送しないとき (つまり、入力内容を保管しており、ユーザーが同一画面の後続の送信でそれを変更する必要がないとき) は、FRSET を使用して MDT をリセットしてください (詳細は 827 ページの『正しい入力の保管』を参照してください)。
- 画面を送信するとき入力フィールドをブランクに初期設定しないでください。入力では、ブランクは伝送され、ヌルは伝送されないためです。したがって、ヌルに初期設定すると、データ・ストリームは各変更フィールドの未使用位置により、短縮されます。入力をマップする場合、画面上の表示は同一で、プログラムに返されたデータも同一です。
- 単一画面のデータ入力操作では、画面を再送信するのではなく、ERASEAUP を使用して画面からデータを消去してください。
- 画面を更新する場合、特に、誤ってフィールドを強調表示した、または画面にメッセージを追加したなど、変更点がそれほど多くない場合には、変更したフィールドのみを送信してください。BMS では DATAONLY オプションを使用して、データ・ストリームの短縮と、パス長の削減の両方ができます (809 ページの『DATAONLY オプション』を参照)。フィールドを強調表示するためには、実際は新規の属性バイトのみを送信します。フィールド・データは変更されることなく画面上に残ります。

- 端末制御コマンドを使用する場合は、設定バッファー・アドレス (SBA) 命令およびアドレス反復 (RA) 命令を使用して形式設定し、ブランクおよびヌルでスペースをとらないようにしてください (BMS を使用する場合、これらは BMS が行います)。

第 67 章 テキスト出力

端末にテキスト出力を送信し、追加の入力のために画面を形式設定する必要がない場合は、マップを作成する必要はありません。BMS では、この目的のために SEND TEXT コマンドが明示的に提供されています。

SEND TEXT を使用するときは、BMS は出力を、送信される端末に対して適切な幅と縦の長さを持つページに切断します。行はワード境界で切り、必要であれば各ページにヘッダーおよびトレーラー・テキストを追加できます。ページ・サイズは他の BMS 出力と同様に判別されます (834 ページの『SEND PAGE コマンド』 を参照)。

SEND TEXT コマンド

実行する形式設定が異なるタイプを除き、SEND TEXT コマンドは SEND MAP と非常に類似しています。形式設定するテキストの位置を FROM オプションに、長さを LENGTH オプションに指定します。マップ出力に適用されるほとんどすべてのオプションが、テキスト出力にも適用されます。次のものが含まれます。

装置制御
FORMFEED、ERASE、PRINT、FREEKB、ALARM、CURSOR

形式設定オプション
NLEOM、L40、L64、L80、HONEOM

後処理オプション
TERMINAL、PAGING、SET

ページ構成オプション
ACCUM

通常、これらのオプションは SEND TEXT コマンドにおいても、SEND MAP コマンドで実行する場合と同じ意味を持ちます。SEND TEXT コマンドそのものは**標準** BMS が必要です。ACCUM、PAGING および SET など、マップ環境において**完全** BMS が必要なオプションは、テキスト環境においても完全 BMS が必要です。

SEND TEXT には、SEND MAP でのマップ関連機能に対応するオプションもあります。HEADER、TRAILER、JUSTIFY、JUSFIRST および JUSLAST です。これらの作用については 848 ページの『テキスト・ページ』で説明します。

SEND MAP オプションのうち、SEND TEXT に引き継がれていないものが 2 つあります。ERASEAUP および NOFLUSH です。ERASEAUP は、テキストがフィールドを最小限にしか使用しないため適用されず、NOFLUSH は、BMS がテキスト出力については OVERFLOW 条件を起こさないため適用されません。

テキスト論理メッセージ

SEND TEXT コマンドに ACCUM オプションまたは PAGING オプションのいずれかがあると、SEND MAP コマンドの場合と同様 BMS に論理メッセージを作成していることが通知されます。テキスト論理メッセージには、マップ論理メッセージ

© Copyright IBM Corp. 1989, 2009

と同一の規則が適用されます (833 ページの『論理メッセージの作成』 を参照)。特に、メッセージの作成に SEND TEXT および SEND CONTROL の両方のコマンドを使用できますが、記載された以外は SEND MAP のコマンドに混合することはできません。また、マップ・メッセージと同一の方法でメッセージを終了します (833 ページの『第 65 章 BMS の論理メッセージ』を参照)。

テキスト・ページ

SEND TEXT によるページ構成は、SEND MAP によるページ構成と多少異なります。まず、単一の SEND TEXT コマンドで、画面またはプリンターのページ・サイズ以上の出力を生成できます (SEND MAP ではこのようにはなりません)。BMS はメッセージ全体を送信します。つまり、複数ページにわたるメッセージを、論理機能を使用せずにプリンターに送達できるということです。ただし、同じ手法をディスプレイに使用することはできません。BMS によりメッセージ全体が送達されても、複数のコンポーネント画面が、通常誰にも読み取れないほど高速に、互いにオーバーレイされるためです。

ACCUM を指定すると BMS は出力をページに切断しますが、二つ目の違いは、SET の後処置を指定しない限りタスクが改ページで制御を入手しないことです。代わりに、現行ページにもう場所がないとき、BMS は新規ページを開始します。ヘッダーおよびトレーラーがあれば自動的に追加し、OVERFLOW 条件を起こしません。これは、ページを単一の SEND TEXT コマンドで生成したか、またはメッセージをばらばらに、複数のコマンドで作成したかにかかわらず当てはまります。タスクに改ページが通知されるのは、後処置が SET である場合のみです。この場合、BMS は RETPAGE 条件を起こして、1 ページまたは複数ページが完了したことを示します。 815 ページの『SET の使用』に説明があります。

BMS が ACCUM を使用してテキスト・ページを作成する方法の詳細を以下に示します。

1. すべてのメッセージは 1 ページ目から開始されますが、最初は空白です。
2. HEADER オプションを指定すると、BMS はすべてのページをヘッダー・テキストで開始します。必要であれば、BMS はヘッダーまたはトレーラーにページ番号を付けます (ヘッダー形式およびページ番号付けについては、 850 ページの『ヘッダーおよびトレーラーの形式』で説明しています)。
3. 位置調整オプション (JUSTIFY、JUSFIRST、JUSLAST) のいずれか 1 つを指定すると、BMS はテキストを指示された行から開始します。JUSFIRST を指定すると、ヘッダーの後の最初の行、またはヘッダーがなければ一番上の行からテキストを開始します。JUSTIFY=n ではテキストを n の行から、また JUSLAST では、テキストとトレーラー (ある場合) の両方が現行ページにおさまる一番下の行から開始します。現行ページの内容によって BMS が位置調整オプションを実行できない場合、BMS はまず新しいページに進みます。ステップ 6 (849 ページ) を参照してください。

   位置調整は各 SEND TEXT コマンドに対し、データの開始にのみ適用されます。データの長さによって追加ページが必要になる場合、BMS はテキストを追加ページの最初の使用可能な位置から継続します。
4. 位置調整を指定しない場合、BMS はテキストを最初の使用可能な位置から開始します。メッセージの最初の SEND TEXT では、JUSFIRST と同一の作業を行

います。その後、テキストは、現行論理メッセージの直前の SEND TEXT のテキストの後、1 文字空けてから続きます (間に入る文字は、3270 端末では属性バイトで、その他の場合はブランクになります)。

5. 開始位置が決定されると、BMS はページの下方に向かって、スペースまたはデータがなくなるまで、『テキスト行』の説明にあるようにデータを行に分割し続けます。トレーラーを指定した場合は、トレーラー用のスペースが必要なため使用可能スペースが削減されます。スペースがなくなる前にデータがなくなった場合は、その時点で処理は終了します。SEND PAGE コマンドまたは PURGE MESSAGE コマンドを使用して終了したことを指示すると、メッセージは完了します。
6. テキストが現行ページに合わない場合、BMS は、下部にトレーラー・テキストがあれば、それを追加することによって、そのテキストを完了し、ちょうどマップされた論理メッセージに対して行ったように、設定した後処理オプション (TERMINAL、PAGING、または SET) に従って後処理します。トレーラーはヘッダーと同様、オプションです。これを指定するには、TRAILER オプションを使用します (850 ページの『ヘッダーおよびトレーラーの形式』を参照)。
7. 次に BMS は改ページし、残りのデータについて、ステップ 2 (848 ページ) から繰り返します。

テキスト行

テキストを行に分割するとき、BMS では次の規則に従います。

1. 通常、各行はブランクとして表示されるものから開始します。3270 装置の場合、これは画面または印刷ページで行の残りの部分を占めるフィールドの、属性バイトです。その他の装置では、ブランクまたは紙送り制御文字です。

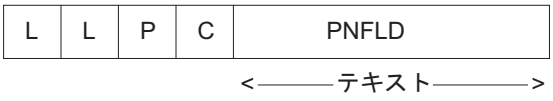
   出力を作成するタスクが PRINTERCOMP(YES) を指定する PROFILE で実行され、しかも出力装置が 3270 プリンターである場合には、例外が発生します。この場合、各行の開始位置には文字は指定されません。 747 ページの『PRINTERCOMP オプション』を参照してください。
2. BMS は、文字用にすべてのブランクを含むテキストの文字をコピーしますが、行末では例外が 2 つ起こります。
   - 行が語の途中で終わる場合、BMS は現在行をブランクで埋め、おさまらない語を次の行の最初の使用可能位置に置きます。この場合の「語」は連続非ブランク文字のあらゆるストリングを指します。
   - 2 つの語が 1 つのブランクで分離されており、最初の語が現在行におさまり、ブランクのための場所が残らない場合、そのブランクは削除され、次の行は 2 つ目の語の先頭から始まります。
3. 宛先の端末がプリンターの場合は、印刷形式を制御するために、ブランクだけでなく改行 (NL) 文字およびその他の印刷形式命令を組み込むこともできます。NL およびタブは特に縦欄のデータで役に立ち、BMS はこれらの文字をフィルター操作はもちろん解釈もしません。ただし、印刷形式命令は表示画面の形式設定はしません。これらの使用についての詳細は、 743 ページの『CICS 3270 プリンター』を参照してください。
4. また、出力に属性設定 (SA) 命令シーケンスを組み込むこともできます (それぞれデータ・ストリームに単一文字の属性を設定します。 632 ページの『属性設定

オーダー』に説明があります)。SA シーケンスが正確に 3 バイトの長さで、有効な属性タイプを表さない限り、BMS はタスクを異常終了します。しかし、有効な SA シーケンスをその属性をサポートしない端末に使用すると、BMS は SA シーケンスを削除してからメッセージを送信します。SA 命令によって設定された属性は、後続の命令によって指定変更されるか、または別の SEND TEXT コマンドがそれらの属性をデフォルト値にリセットするまでそのままです。

テキストに SA 以外の 3270 命令を組み込まないでください。BMS はそれらを表示データとして扱い、望むとおりに形式設定しません。端末エラーを起こすことさえあります。

ヘッダーおよびトレーラーの形式

テキスト・メッセージのページにヘッダーを配置するために、HEADER オプションで次の形式によってデータのブロックを示します。



トレーラー・テキストの場合も同じ形式を使用します。その場合は TRAILER オプションに示します。この図において、

LL ヘッダー (またはトレーラー) データの長さで、LL、P、および C の文字の 4 バイトは含まれません。LL はハーフワード 2 進数形式で表さなくてはなりません。

P ページ番号置換文字です (後の PNFLD を参照)。ページ番号が必要ない場合はブランクを指定してください。

C 予約されている 1 バイトのフィールドです。

TEXT 出力の各ページの最上部 (最下部) に配置するヘッダー (トレーラー) テキストです。行を複数にする場合には改行文字 (X'15') を使用し、改行する位置を指示します。

PNFLD

ヘッダー (トレーラー) テキスト内の、ページ番号フィールドです。出力にページ番号を付けたい場合は、ヘッダー (トレーラー) テキストの中で使われていない文字を 1 つ選択してください。この文字を、ページ番号を表示したい位置に置きます。1 から 5 桁までの隣接する位置を使用できますが、ページ番号がどの程度まで大きい数になるかによって異なります (BMS で指定できる最大数は 32,767 です)。先に示した P フィールドに同じ文字を置き、BMS に置換を行う場所を指示します。
X'0C'、X'15'、X'17'、X'26' または X'FF' は P には指定しないでください。これらの値は他の目的で使用するために予約済みです。ページ番号を付けない場合は、ブランク (X'40') を P に指定してください。

論理メッセージを作成するときは、各 SEND TEXT コマンドで HEADER オプションおよび TRAILER オプションを繰り返してください。そうすると、改ページがきたときにこれらが表示されます。また、メッセージを終了する SEND PAGE コマンドで、再度このトレーラーを指定する必要があります。

ページ番号を付けるヘッダーのための COBOL 定義例を示します。ページ数のための場所は 99 まで空けてあります。

```
    EXEC CICS SEND TEXT FROM (OUTPUT-AREA)
              HEADER(HEADER-TEXT) PAGING ACCUM END-EXEC.
```

ここで、

```
01  HEADER-TEXT
    02  HEADER-LL          PIC S9(4) COMP VALUE +11.
    02  HEADP              PIC X VALUE '@'.
    02  FILLER             PIC X VALUE LOW-VALUE.
    02  HEADING            PIC X(11) VALUE 'PAGE NO. @@'.
```

SEND TEXT で作成された画面は、端末オペレーターからの大量の入力に対応して設計されているわけではありません。ただし、画面のフィールド構造が適切であり、オペレーターが求められているものを認識しているか調べられる場合は、アテンション ID を解釈して簡単な入力 (ページ表示の制御のため CSPG トランザクションで使用するものなど) を読み取ることができます (新規フィールドは各行で開始されます。メッセージを作成した各 SEND TEXT コマンドによって送信されたテキストの先頭文字も同様です。定義済みフィールドは、オペレーターが入力できるように、無保護、英数字で、通常輝度となっています)。通常、端末管理 RECEIVE はこのような状況で使用されます。画面と同じフィールド構造を持つマップを作成できる場合に限り、RECEIVE MAP を使用することができます。

SEND TEXT MAPPED および SEND TEXT NOEDIT

BMS には SEND TEXT コマンドの 2 つの特殊な形式があり、これらを使用すると既に形式設定された出力に対して、BMS のメッセージ送達機能のいくつかを使用することができます。SEND TEXT MAPPED は、*BMS が以前に作成し*、SET オプションによって検索した装置依存データのページ を送信します。もともとは SEND MAP コマンドまたは SEND TEXT コマンドのいずれかを使用して作成したページであるとも考えられます。

SEND TEXT NOEDIT も似ていますが、これはプログラムまたは BMS 以外の方法によって作成した装置依存出力のページを送信するために使用します。

このようなページを、TERMINAL の後処置を使用して自分自身のプリンシパル装置に個別に送達できます。または、PAGING オプションで作成した論理メッセージにこのページを組み込むこともできます。論理メッセージにおいては、各 BMS SEND が別々のページを表す (ACCUM オプションは使用されない) 限り、これらの形式は通常の SEND TEXT コマンドまたは SEND MAP コマンドと混合できます。

これらのコマンドは、ルーティング環境でも使用できます。自分自身の端末に対し、ルーティングするか送信するかにかかわらず、データ・ストリームが、宛先に対して適切であることを確認してください。BMS は、宛先がサポートしない 3270 属性を除去する以外、送信前に検査を行いません。

1. この場合、同じ論理方式でのテキストとマップ出力の混合に対する通常の制約事項は、ページが既に形成されているため、あてはまりません。

ページ構成オプションの ACCUM、JUSTIFY、JUSFIRST、JUSLAST、HEADER、および TRAILER はいずれも、これらのコマンドのいずれにも適用されません。ページは既に定義により、形式設定され、作成されているためです。

MAPPED 形式と NOEDIT 形式の基本的な違いは、SEND TEXT MAPPED の場合、BMS によって SET オプションで返されたページの終わりに追加した 4 バイトのページ制御域 (PGA) を使用するという点にあります。この区域は BMS に、使用する書き込みコマンドおよび書き込み制御文字、そのページで使用した拡張属性、およびページの紙送り制御、SCS 形式によるデータ、14 ビットまたは 16 ビットのバッファー・アドレス、構造化フィールド、および FMH が含まれるかどうかを指示します。これにより BMS は、ある装置のために作成したページを、異なるハードウェア特性を持つ装置に送達することができます。これはページのコピーまたはルーティング操作の際に必要になることがあります。SEND TEXT NOEDIT を使用して、このタイプの情報をコマンドそのものに指定します。BMS で作成した出力には SEND TEXT MAPPED を、その他の方法で形式設定した出力には NOEDIT を使用してください。ただし、SEND TEXT MAPPED または SEND TEXT NOEDIT では、出力に構造化フィールドを組み込むことはできません。このような出力には端末管理 SEND を使用しなくてはなりません。

SEND TEXT MAPPED コマンドのための LENGTH オプションは、ページが作成されたときに返される TIOATDL 値から設定してください。これには PGA は含まれません。ただし、このページを将来 SEND TEXT MAPPED を用いて使用するためにコピーする場合は、ページだけでなく PGA も必ずコピーしなくてはなりません (全部で TIOATDL + 4 バイト)。

関連情報

815 ページの『SET の使用』

853 ページの『第 68 章 メッセージ・ルーティング』

第 68 章 メッセージ・ルーティング

BMS のメッセージ・ルーティング機能を使用すると、メッセージをタスクのプリンシパル装置以外の端末に送信できます (タスクにプリンシパル装置がなくても構いません)。ルーティングによってタスクがこれらの端末を直接制御できるわけではありませんが、宛先ごとにタスクをスケジューリングすることで、メッセージを送信するようにできます。これらのタスクは CICS 提供トランザクションの CSPG を実行します。これは、PAGING の後処置を用いてユーザー自身の端末にメッセージを送信するために使用するものと同じです。したがって、ルーティング・メッセージを受信するディスプレイ端末のオペレーターは、メッセージを表示させるために CSPG 要求を使用します (CSPG についての詳細は、836 ページの『端末オペレーターのページング: CSPG トランザクション』を参照してください)。

メッセージ・ルーティングは、メッセージ交換およびブロードキャストのアプリケーション、また印刷の場合にも便利です (752 ページの『CICS プリンターの使用』を参照)。これは、CICS 提供トランザクションの CMSG の基本です。CMSG を使用すると、端末ユーザーが他の端末およびユーザーにメッセージを送信できます。CMSG に関する説明、およびその機能については、「*CICS Supplied Transactions*」のCMSG - メッセージ交換を参照してください。

メッセージをルーティングする場合は、ROUTE コマンドを発行して開始します。このコマンドは、メッセージを送信する場所、送達するとき、エラーにどう対処するか、およびその他の詳細を BMS に指示します。次にメッセージを作成します。これはマップ・メッセージでもテキスト・メッセージでもかまいませんが、論理メッセージでなくてはならず (つまり ACCUM または PAGING のいずれかを指定する)、後処置は PAGING または SET のいずれかでなくてはならず、TERMINAL ではいけません。PAGING がより一般的な選択で、以下で述べる説明も、これを想定して行います。ルーティングにおける SET については、862 ページの『SET でのルーティング』 に説明されています。

ROUTE コマンドは、SEND PAGE コマンドを使用してメッセージを終了するまで有効であり、これを行うまでは別のコマンドを出してはいけません (メッセージを作成する間に ROUTE を出すと、無効な要求応答を受け取ります。論理メッセージを開始する前に 2 つ目の ROUTE を出すと、最初のものが置き換えられます)。メッセージを送信しないことに決める場合は、PURGE MESSAGE を使用してメッセージを終了することもできます。PURGE MESSAGE を使用すると、ROUTE コマンドによって設定されたルーティング環境は、論理メッセージとともに廃棄されます。

メッセージの宛先

ルーティング・メッセージの宛先は、以下の 3 つの異なる方法で指定することができます。

© Copyright IBM Corp. 1989, 2009

- ROUTE コマンドの OPCLASS オプションを使用することにより、一定のクラスのオペレーターがメッセージを受信するように要求できます。クラスは、RACF ユーザー定義、または CICS サインオン・テーブル項目のオペレーターと関連付けられています。
- 経路リストを使用することにより、メッセージを受信するべき特定のオペレーターを指定できます。この経路リストは、ROUTE コマンドの LIST オプションにより示します。オペレーターは、3 文字の OPIDENT 値によって識別されます。この値は、RACF 定義またはサインオン・テーブル項目にも割り当てられます。
- メッセージを受信するべき特定の端末を指定できます。これも、経路リストを使用して行います。端末は 4 文字の TERMID 値によって識別されますが、この値が適用される端末タイプの場合は、2 文字の論理装置コードです。

注: ルーティング・メッセージの宛先を指定するため、プリンシパル装置にサインオンしたオペレーターの ID またはオペレーター・クラス値を知る必要がある場合は、ASSIGN コマンドを OPID オプションまたは OPCLASS オプションを指定して使用し、調べることができます。

適格である端末

特定の宛先に対してメッセージを正しく形式設定するために、BMS は形式設定するための端末の特性を認識する必要があります。これは、オペレーターまたはオペレーター・クラスによって指定する宛先にも当てはまります。このため、経路リスト処理の最初のステップは、メッセージが送達される*可能性* のある*端末* のリストに宛先を変換することです。この「適格端末」リストは、経路リストおよび OPCLASS 指定に関する情報と、*ROUTE コマンドの時点における* 端末ネットワークの状況を組み合わせています。

その後メッセージを送達する用意ができたら、BMS はこのリストを使用してメッセージを受け取る端末を決定します。端末はメッセージを受信するためにはリストにのっていなくてはなりませんが、リストにのっていても送達されるとは限りません。その端末を担当するオペレーターに制限がある場合があり、さらに、送達は後になって行われるため、端末の状況または、性質までが変更されていることもあるためです。

リスト作成時と送達時の両方で、BMS はそれ自体の CICS 領域 (ルーティング・タスクが実行されている、または実行された) にインストールされた端末定義に制約され、期待する端末定義のすべてを持たない場合も考えられます。最初に、自動インストールされる端末は、(リストに含まれるものからはこれらを除く) ROUTE 実行の際にも、送信しようとする際にもログオンされず送達できない場合があります。

さらに複数領域環境では、ある領域に認識される端末が別の領域では認識されないということも考えられます (これは、「*CICS Resource Definition Guide*」の TYPETERM 定義の属性に説明されているように、その定義の内容によって異なります。) 特に端末定義が、その定義を所有する領域に SHIPPABLE と指定することにより領域の間で共有されている場合、送達の原因となるようなことが起こるまでその端末は他の領域で定義されません。これは、通常、端末が問題の領域にトランザ

クションを初めてルーティングするときに起こります。結果として、この領域の ROUTE には、最初にこのようなイベントが起こる前に端末を組み込むことはできません。

以下のセクションでは、BMS が適格な端末のリストを作成する方法を説明します。これは、ROUTE コマンド時に起こります。

OPCLASS のみで指定された宛先

オペレーター・クラス (OPCLASS オプション) を指定しても、経路リストを指定していない場合、BMS はローカル・システムのすべての端末定義をスキャンします。これらの条件すべてに合致する端末はすべて、適格端末リストに載ります。

- 端末が、BMS がサポートするタイプである。
- 端末の宛先が明確に指定されていなくても、その端末が、ルーティング・メッセージを受信できる (端末定義の ROUTEDMSGS (ALL))。
- オペレーターが端末にサインオンしている。
- オペレーターが、OPCLASS リストのオペレーター・クラスのいずれか 1 つに属している。

OPCLASS リストのオペレーター・クラスの少なくともいずれか 1 つに属するオペレーターがサインオンしていればその場合にのみ送達が行われるように、最終的な項目には印が付けられます (このオペレーターは、ROUTE 時にサインオンしていたオペレーターである必要はありません)。

OPCLASS および LIST の省略

オペレーター・クラスと経路リストのいずれも指定しない場合、BMS は上記テストの最初の 2 つに合致するすべての端末をリストに載せ、送達についてオペレーター制限を設定しません。すべてのルーティング・メッセージを受信するのに適格である端末が多く存在するネットワークでは、これは、できるだけ避けるべき選択です。

提供される経路リスト

ユーザーが経路リストを提供する場合、BMS は端末定義をスキャンせずにそのリストから BMS 用のリストを作成します。項目はそれぞれ次のように処理されます。この処理にはリスト項目に状況フラグを設定することが含まれ、項目が使用されたかまたはスキップされたか、スキップされた理由は何かを示します。

- 項目に端末 ID は含まれるがオペレーター ID は含まれない場合、端末は適格リストに載ります (その端末が定義済みで、BMS がサポートするタイプであり、ルーティング・メッセージの受信に適格である場合)。 BMS が端末定義を検出できない場合、経路リスト項目の状況フラグに「項目がスキップされた」および「端末 ID が無効」のビット (X'C0') が設定されます。端末は存在するが、BMS がサポートしないかまたはいかなるルーティング・メッセージの受信も許可されていない場合、「項目がスキップされた」および「端末が BMS でサポートされていない」のビット (X'A0') が設定されます。

  注: 端末がルーティング・メッセージを受信するための適格性は、端末定義の ROUTEDMSGS オプションにより管理されます。以下の 3 つの値が可能です。すなわち、端末がすべてのルーティング・メッセージの受信を許可される、また

は、端末またはオペレーター名によってルーティングされたメッセージのみの受信を許可される、または、ルーティング・メッセージの受信をまったく許可されない、の 3 つです。経路リストだけでなく OPCLASS も指定した場合、リストしたクラスのいずれか 1 つに属するオペレーターが端末でサインオンしているかどうかについて BMS は検査を行います。サインオンしていない場合、BMS は「オペレーターがサインオンしていない」のビット (X'10') を項目の状況フラグに設定して通知しますが、その場合でも端末を組み込みます。オペレーター・クラスを指定する場合でも、リスト項目に関連したオペレーター制限は存在しません。

- 項目に端末 ID とオペレーター ID の両方が含まれる場合、端末 ID はオペレーター ID がない場合と同じ方法で検査され、同じエラーが起こる恐れがあります。端末がこれらのテストに合格すると、適格リストに載ります。ただし、指定されたオペレーターが同一の端末でサインオンしているときのみメッセージを送達できるように項目には、印がつきます。

  このオペレーターが ROUTE コマンド時に端末にサインオンしていない場合、BMS は「オペレーターがサインオンしていない」のビット (X'10') を状況フラグでオンにすることによって通知しますが、サインオン状況にかかわりなく端末は送達リストに載ります (オペレーター ID がある場合、OPCLASS は完全に無視されます)。
- 項目にオペレーター ID のみが含まれる場合、BMS は、オペレーターがサインオンしている端末を検出するまで、端末定義を検索します (オペレーターが追加の端末にサインオンしていることもありますが、BMS はこれらの端末を無視します)。この端末が BMS によりサポートされないタイプである場合、またはこの端末がルーティング・メッセージを受信できない場合、BMS は「項目がスキップされた」および「オペレーターが、サポートされていない端末でサインオンしている」のビット (X'88') を状況フラグに設定します。また、経路リストにも端末 ID を指定します。端末が適切である場合、BMS は、既に説明した、端末 ID とオペレーター ID の両方を指定した場合と同様に項目を扱います。

  オペレーターがどこにもサインオンしていない場合、BMS は「項目がスキップされた」および「オペレーターがサインオンしていない」のビット (X'90') を状況フラグに設定します。

経路リストの形式

BMS では、経路リストが固定形式でなければなりません。リストの各項目は 16 バイト長で、次のようになっています。

表 48. 経路リスト項目の標準形式

バイト	内容
0-3	端末 ID または論理装置 (LU) ID (末尾ブランクを含めて 4 文字)、またはブランク
4,5	LDC サポートのある論理装置 (LU) のための LDC 略号 (2 文字)、またはブランク
6-8	オペレーター ID、またはブランク
9	ルーティング項目の状況フラグ
10-15	予約済み。ブランクが入っていることが必要。

各項目には、端末 ID またはオペレーター ID のいずれかが指定されていなくてはなりません。論理装置コンポーネント (LDC) はいずれにも伴います。LDC の詳細は、879 ページの『LDC とルーティング』を参照してください。

経路リストの項目は、通常、順々に 1 つずつ続きます。ただし、すべて隣接している必要はありません。リストに連続しない部分がある場合、連続する項目の各グループを終了します。ただし、次のグループの最初の項目を示す 8 バイトのチェーン項目を含む最後のグループは除きます。この項目は次のようになります。

表 49. 経路リスト・チェーン項目の形式

バイト	内容
0, 1	-2、2 進数ハーフワード形式 (X'FFFE')
2,3	予約済み
4-7	隣接する項目の次のグループにある最初の項目のアドレス

リスト全体の終わりは、-1 のハーフワード値 (X'FFFF') を含む 2 バイトの項目で示されます。

リストは、必要なだけの数のグループで構成することができます。宛先の合計数に上限はありますが、これは多くの変数によって変わります。上限を超える場合、BMS は異常終了コード ABMC を用いてタスクを異常終了します。

ROUTE コマンドからの戻りで、BMS は条件コードを起こしてリスト内のエラーを通知します。

RTESOME
経路リストのうち少なくとも 1 つの項目が使用できず、スキップされたことを意味します。デフォルトのアクションでは、正常に処理された宛先を使用してルーティング操作が継続されます。

RTEFAIL
リスト内の宛先がいずれも使用できず、そのためルーティング環境がまったく設定されなかったことを意味します。デフォルトのアクションでは、タスクに制御を返します。ルーティング環境が設定されないと、結果として、後続の BMS SEND コマンドが、意図しないのにプリンシパル装置に対して実行されることになるため、この条件はテストする必要があります。

RTESOME および RTEFAIL により反映される一般的な情報に加え、BMS は状況フラグ (バイト 9) を設定することによって、リストの各項目で行われたことを通知します。ヌルの値 (X'00') は項目が完全に正常だったことを意味します。高位ビットは、項目が使用されたかまたはスキップされたかを、他のビットは何が起こったかを詳細に示します。表示される各ビットの意味は次のとおりです。

項目がスキップされた (X'80')
項目は使用されませんでした。このビットがオンになっているときは、別のビットもオンになってその理由を示します。

端末 ID が無効 (X'40')
項目に指定された端末の端末定義がありません。この項目はスキップされます。

端末が BMS でサポートされていない (X'20')
経路リスト項目に指定された端末は BMS がサポートしないタイプであるか、またはルーティング・メッセージの受信ができないよう制限されています。この項目はスキップされます。

オペレーターがサインオンしていない (X'10')
項目に指定されたオペレーターがサインオンしていません。次の条件のうちいずれの場合でも、このフラグが設定されます。

- オペレーター ID と端末 ID が共に指定されたが、そのオペレーターがその端末でサインオンしていませんでした。項目はスキップされません。
- 端末 ID は指定されずにオペレーター ID が指定されたが、このオペレーターがどの端末でもサインオンしていませんでした。この項目はスキップされます。
- OPCLASS が ROUTE コマンドに指定され、端末 ID が経路リスト項目に指定されたが、端末がサインオンしたオペレーターに、指定されたオペレーター・クラスがありません。項目はスキップされません。

オペレーターが、サポートされていない端末でサインオンしている (X'08')
経路リスト項目にオペレーター ID のみが指定され、そのオペレーターは、BMS がサポートしない端末、またはルーティング・メッセージの受信に適格でない端末でサインオンしています。この項目はスキップされます。この端末の名前は、項目の端末 ID フィールドに返されます。

LDC 簡略記号が無効 (X'04')
次の条件のいずれの場合でも、このフラグが設定されます。

- 経路リストに指定した LDC 簡略記号は、この端末には定義されていません。つまり、この端末は LDC をサポートしていても LDC リストがないか、または LDC リストが拡張されているがこの項目を含まないということです。
- この LDC 項目のための装置タイプは、LDC のある経路リストの最初の項目の装置タイプと異なります (879 ページの『LDC とルーティング』 で説明しているように、使用できる LDC 装置タイプは 1 つだけです)。

この項目はスキップされます。

注: CICS は、標準経路リスト項目、および状況フラグ・ビットの組み合わせのテストに必要な値を定義する、ソース・コードを提供しています。DFHURLDS メンバーの COPY または INCLUDE を使用して、このコードをプログラムに挿入することができます。BMS アテンション ID や属性バイト定義を組み込むのと同じ方法です。

メッセージの送達

BMS がルーティング・メッセージの受信に適格な端末を判別する方法については説明しました。実際の送達はそれより後に行われ、場合によってはかなり遅くなりますが、これは ROUTE コマンドのスケジューリング・オプション (INTERVAL、TIME、AFTER および AT) によって異なります。一定の時間が経過した後、または一日の特定の時刻に、即時送達するように要求することができます。

指定した時間になると、BMS は適格な端末リストにあるすべての端末にメッセージの送達を試みます。メッセージが特定の端末に送達されるためには、次に示す条件がすべてそろっていなくてはなりません。

- 端末は、BMS がサポートするタイプとして、また ROUTE コマンドが処理されたときと同じタイプとして定義されなくてはなりません。メッセージの作成と送達との間に長い遅延が起こる場合は、特定の TERMID で定義された端末が、(特に自動インストール環境において) 特性を変更したり、消失したりしている可能性があります。3270 端末は、ROUTE コマンドの発行時と完全に同じ拡張属性を持つ必要はありません。これは、BMS が送達時に、サポートされない属性をデータ・ストリームから削除するためです。
- 端末は作動中で、使用可能でなくてはなりません (この端末で、プリンシパル装置として実行中のタスクがあってはいけません)。
- 端末は自動トランザクション開始に適格でなくてはなりません。または、端末オペレーターが CSPG トランザクションを使用してメッセージ送達を要求しなくてはなりません。

  注: 複数のメッセージが特定の端末に送達されるように累積されている場合には、オペレーターがそれらのメッセージを特定の順序で見るという保証はありません。実際には、状況によって、CSPG トランザクションにより、オペレーターが送達順序を制御できる場合もあります。ページに特定のシーケンスが必要な場合には、ページを 1 つのメッセージとして送信しなければなりません。
- 送達リスト項目が、ある一定のクラスに属する特定のオペレーターまたは複数のオペレーターへの送達を制限している場合には、その端末にサインオンしているオペレーターが適格でなければなりません (これらの制限事項を生成する OPCLASS 指定および LIST 指定については、853 ページの『メッセージの宛先』を参照してください)。
- パージ遅延を終了させてはいけません。これについては次のセクションで説明します。

送達不能なメッセージ

BMS が適格な端末にメッセージを送達できない場合、次の条件のいずれか 1 つが起こるまで定期的に継続します。

- 端末状況が変わって、メッセージが受け取り可能になる。
- 宛先端末オペレーターがメッセージを削除する。
- パージ遅延が経過した。

パージ遅延とは、一度送達をスケジュールされたメッセージの、送達に許された一定の時間のことです。この一定の時間が経過した後、そのメッセージは廃棄されます。パージ遅延はシステム全体に対する値で、システム初期設定テーブルの PRGDLY オプションによって設定します。この値の使用はオプションです。システム・プログラマーが PRGDLY をゼロに設定すると、メッセージは無期限に保管されます。

BMS はメッセージをこの方法でパージするときに、ERRTERM に指定している端末にエラー・メッセージを送信します (特定の端末名を指定せずに ERRTERM を使

用すると、メッセージはもともとそのメッセージを作成したタスクのプリンシパル装置に送信されます。ERRTERM を完全に省略すると、メッセージは送信されません)。

リカバリー可能メッセージ

PAGING の後処置を使用した経路指定メッセージの作成と送達との間に、BMS はそのメッセージを、通常の PAGING メッセージの場合と同様、CICS 一時記憶域に格納します。その結果、REQID オプション値を選択することにより、ルーティング・メッセージをリカバリー可能にすることができます。これは、ルーティング・メッセージ以外のメッセージの場合と同様です (837 ページの『論理メッセージのリカバリー』を参照してください)。

ルーティング先の端末のタイプが複数ある場合、BMS は適切な装置依存データ・ストリームを使用して、各タイプごとに別々の論理メッセージを作成し、各タイプごとに別々の一時記憶キューを使用します。

注: 代替画面サイズ機能を持つ端末の宛先の場合、2 つのメッセージ形式が使用できますが、BMS は、メッセージを作成するタスクが使用するプロファイルがデフォルト・サイズを指定する場合はデフォルト・サイズを、プロファイルが代替サイズを指定する場合は代替サイズを選択します。

ただし、すべての論理メッセージは同一の REQID 値を使用するため、メッセージをリカバリー可能にするか、または可能にしないかをまだ選択できます。

BMS はまた、メッセージの受信に適格な端末のリストを格納するため、また送達が行われたかどうかを追跡するために一時記憶域を使用します。特定のタイプの適格な端末のすべてがメッセージを受信したとき、BMS は関連した論理メッセージを削除します。すべての宛先が送達を受け取ったとき、またはパージ遅延が終了したとき、BMS はそのメッセージのすべての情報を消去し、宛先ごとに送達不能なメッセージの数をマスター端末オペレーター・メッセージ・キューに報告します。

メッセージの識別

必要な場合は、ルーティング・メッセージに表題を割り当てることができます。表題はメッセージそのものの一部ではありませんが、BMS がメッセージについて保守する他の情報とともに CICS 一時記憶域に組み込まれます。表題は、オペレーターまたは端末に対して多くのメッセージが累積されているような状況で役立ちます。表題を使用すると、オペレーターはメッセージの表示順序を制御できるためです。(「*CICS Supplied Transactions*」のCSPG - ページ検索にある、CSPG コマンドの「照会」オプションを参照してください。)

表題を割り当てるためには、ROUTE コマンドの TITLE オプションを使用して、ハーフワード 2 進数の長さフィールドとその後に続く表題から構成されるデータ域を指すようにします。この長さは最大 64 文字で、長さが 2 バイトのフィールドが含まれています。したがって、表題そのものの長さは 62 文字までです。例:

```
01  MSG-TITLE.
    02  TITLE-LENGTH    PIC S9(4) COMP VALUE +19.
    02  TITLE-TEXT      PIC X(17) VALUE 'MONTHLY INVENTORY'.
...
EXEC CICS ROUTE TITLE(MSG-TITLE)....
```

図 154. 表題の割り当て

ルーティングに関するプログラミングの考慮事項

ルーティング・メッセージは、通常、それ以外のメッセージと同じ方法で作成します。しかし、BMS は宛先の中のそれぞれの端末タイプに対し別々の論理メッセージを作成するため、いくつか違う点もあります。初めにページ・オーバーフローについて説明します。

ルーティングとページ・オーバーフロー

端末のタイプが異なるとページ容量も異なるため、ページ・オーバーフローは、異なる場合に異なるタイプで起こると考えられます。SEND MAP コマンドを使用し、ページ・オーバーフローを代行受信する場合、ROUTE に応じて BMS が作成する各論理メッセージの各ページでページ・オーバーフローが起こると、プログラムは制御を取得します。

ページ・オーバーフローの発生時に、ページに番号を付けたい場合、またはページに依存する処理を行いたい場合は、端末タイプごとの情報を別々に把握していなければならない場合もあります。この目的のために保持されているデータ域を、**ページ・オーバーフロー制御域**といいます。このような区域がいくつ必要か (ROUTE コマンドに異なる端末タイプがいくつ表示されているか) を指示するには、DESTCOUNT オプションを指定した ASSIGN コマンドを、ROUTE の後で、しかもページ・オーバーフローを起こす可能性のある BMS コマンドの前に発行します。このときに出すと、ASSIGN DESTCOUNT は BMS が作成する論理メッセージの数を返します。

ページ・オーバーフローが起こったとき、同じコマンドを使用して、ページ・オーバーフローが起こった論理メッセージを判別できます。このとき ASSIGN DESTCOUNT は、BMS がこの ROUTE コマンドのために作成しているメッセージの中での、そのメッセージの相対番号を返します。ページ・オーバーフロー制御域を使用している場合、この番号は使用している制御域を示します。このときに ASSIGN PAGENUM を使用すると、BMS はページ・オーバーフローが起こったページ数も返します。

異なる端末タイプについて異なる時点で起こる、複雑なページ・オーバーフローを処理するために、ルーティング環境でページ・オーバーフローについて行うべき処理を以下に示します。

- ASSIGN DESTCOUNT を使用してページ・オーバーフローが起こった論理メッセージを判別する (非常に簡単なページ・オーバーフロー処理を行っている場合は除く)。
- 非ルーティング環境で行うように、現行ページ用にトレーラー・マップを送信し、その後で次のページ用のヘッダーを送信する (840 ページの『改ページ:

BMS のオーバーフロー処理』を参照)。OVERFLOW 条件が有効となっている間は、これらの SEND MAP コマンドは、ページ・オーバーフローが起こった論理メッセージにのみ適用されます (ページの途中の論理メッセージにこれらのコマンドを適用したくない場合もあり、BMS は、偶然に同じページ容量を持つ異なる端末タイプについては想定していません)。

- 非ルーティング環境の場合と同様に、オーバーフローの原因となったコマンドを再発行する。ただしこれを行った後に、ページ・オーバーフローが起こらなくなるまで、ページ・オーバーフローの再テストを行い、処理全体を繰り返さなくてはなりません。このプロシージャーにより、ページ・オーバーフローの原因となったトレーラー、ヘッダー、およびマップが、作成する各論理メッセージに確実に組み込まれます。

SET でのルーティング

ルーティング環境で SET の後処置を指定しているときは、経路リストにある宛先にメッセージは送信されません。ページは完了するとプログラムに返されるためです。しかし、ROUTE コマンドは通常の方法で処理すると、これらの宛先および宛先の中の端末タイプを判別します。BMS は、通常どおり各タイプに別々の論理メッセージを作成し、いずれの端末タイプに対しても 1 ページ完了するごとにそのページをプログラムに返します。BMS は、PAGING の後処置の場合と同様 OVERFLOW 条件および RETPAGE 条件を起こします。結果として、SET とともに ROUTING を使用すると、プリンシパル装置のタイプ以外の端末タイプのためにメッセージの形式設定ができます。

メッセージ・ルーティングによる会話のインターリービング

ルーティングするメッセージを作成する場合、プリンシパル装置との通信に、RECEIVE MAP および端末制御コマンドだけでなく BMS SEND コマンドも使用することができます (833 ページの『論理メッセージの作成』 で説明されているように、ルーティングしない場合、BMS SEND は使用できません)。このような SEND コマンドでは、後処置オプションは PAGING または SET ではなく、TERMINAL でなければなりません。また、ACCUM を指定してはいけません。関連した入出力は直接処理され、端末が論理メッセージの宛先の 1 つである場合でも、この論理メッセージに干渉しません。

第 69 章 MAPPINGDEV 装置

最小 BMS 機能では、タスクのプリンシパル装置が**マッピング装置**であることを想定します。この装置は、TCTTE (端末管理表項目) で定義される機能および状況に対する入出力のマッピング操作を実行します。

BMS を使用するトランザクションのプリンシパル装置には、BMS がサポートする装置タイプが必要です。しかし、MAPPINGDEV 機能は最小 BMS の拡張機能で、それによってユーザーはプリンシパル装置では**ない**装置に対するマッピング操作を行うことができます。MAPPINGDEV 要求が完了すると、マップされたデータがアプリケーションに返されます。BMS は、MAPPINGDEV 装置と通信することはできません。

MAPPINGDEV オプションは、RECEIVE MAP コマンドおよび SEND MAP コマンドに指定できますが、それ以外の BMS コマンドには指定できません。

MAPPINGDEV オプションで指定された TERMID は、BMS でサポートされた 3270 ファミリーの装置を表していなければなりません。その装置が区分されている場合は、基本状態にあると想定されます。外部様式設定機能は無視されます。

データは最小 BMS の場合と完全に同じ方法でマップされるので、マップ・セットの定義を変更したり、再生成したりする必要はありません。

MAPPINGDEV オプションを指定した SEND MAP

MAPPINGDEV オプションを指定した SEND MAP コマンドでは、SET オプションも指定する必要があります。 (SET オプションによって、マップ済み出力データ・ストリームを含むストレージ域のアドレスを設定するポインターが BMS に指定されます)。

ストレージ保護をアクティブにすると、トランザクション定義の TASKDATAKEY オプションで指定されたキーのストレージにデータが返されます。ストレージは、ユーザーがトランザクション定義に指定した TASKDATALOC オプションに応じて、その境界線の上または下に配置されます。

ストレージ域はタスク関連ユーザー・ストレージ域内にありますが、その形式は TIOA (端末入出力域) です。アプリケーションは、DFHTIOA コピーブックを使用してストレージ域を参照することができます。オフセット 8 の TIOATDL フィールドには、ストレージ域内のオフセット 12 の TIOADBA から始まるデータ・ストリームの長さが入っています。長さ 4 バイトのページ制御域は、TIOATDL 内に配置された長さの値には含まれません。この制御域には、データ・ストリーム内で使われてきた拡張属性などの情報が含まれており、DFHPGADS コピーブックを使用してそれを参照することができます。

© Copyright IBM Corp. 1989, 2009

ストレージ域は、通常データ・ストリームより長くなります。それは、出力データ・ストリームの正確な長さが決定される前にストレージ域が割り振られているためです。このストレージ域は、SEND TEXT MAPPED コマンドで使用できる形式になっています。

MAPPINGDEV オプションを指定しない SET オプションの使用について (詳しくは 627 ページの『保護』 を参照) 十分理解していればお分かりのように、データ・ストリームがページのリストによって間接的にアプリケーションに返されます。しかし、MAPPINGDEV を指定すると、データ・ストリームが含まれているストレージ域を直接指すポインターがアプリケーションに返されます。

SEND MAP MAPPINGDEV コマンドが処理を完了すると、ストレージ域はアプリケーションで制御されるようになり、アプリケーションがそれを FREEMAIN しない限り、トランザクションが終了するまで割り振られたままになります。長時間実行するトランザクションの場合には、このようなストレージ域を FREEMAIN することが推奨されますが、タスクが終了すると、CICS はこれらの領域を解放します。

MAPPINGDEV オプションを指定した RECEIVE MAP

RECEIVE MAP コマンドで MAPPINGDEV オプションを使用する場合は、FROM オプションを指定する必要があります。BMS では、FROM オプションを使用して、端末管理 RECEIVE コマンドにより返されたデータ・ストリームとの整合性がある、形式設定された 3270 入力データ・ストリーム (すなわち、通常の 3270 入力データ・ストリーム) を提供する必要があります。唯一違う点は、BMS が AID や入力カーソル・アドレスでは開始しないことです。その理由は、この情報が端末制御によって入力データ・ストリームから削除されるからです。しかし、RECEIVE MAP コマンドには、MAPPINGDEV オプションを指定するときに、AID 値や入力カーソル位置を指定できるようなオプションはありません。データ・ストリームに AID 値および入力カーソル・アドレスが含まれている場合、それらは BMS によって無視されます。

どちらのオプションも指定されていない場合、BMS は入力データ操作が Enter (実行) キーで終了したと想定し、EIBAID フィールドからアプリケーションに適切な AID 値を返します。BMS はまた、入力カーソルがホーム・アドレスに位置指定されたと想定し、EIBCPOSN フィールドからアプリケーションに値ゼロを返します。

RECEIVE MAP コマンドの新規 AID オプションによって、アプリケーションは AID 値を指定できます。この値が指定されると、デフォルト値である ENTER が上書きされます。アプリケーションにより指定されている場合でも、BMS によりデフォルト指定されている場合でも、設定した AID 値によって、アプリケーションが発行した直前の HANDLE AID 要求により登録されたルーチンに制御が渡されます (適用可能な場合)。

RECEIVE MAP コマンドの新規 CURSOR オプションによって、アプリケーションは入力カーソル位置を指定することができます。この位置が指定されると、デフォルト値であるゼロが上書きされます。アプリケーションにより指定されている場合でも、BMS によりデフォルト指定されている場合でも、CURSLOC=YES を指定してマップを定義するときのカーソルの位置指定処理には、入力カーソル値が使用されます。

最小 BMS RECEIVE MAP コマンドの場合と同様に、INTO オプションまたは SET オプションによって、アプリケーションにマップされたデータが返されます。どちらのオプションも指定されていない場合、CICS 変換プログラムは、マップ名に 'I' という文字を追加することによって、デフォルトの INTO オプションを適用しようとします。

MAPPINGDEV オプションを持つ SET オプションを使用する場合、それによって、マップされた入力データ・ストリームを含むストレージ域のアドレスを持つ、BMS が設定するポインター変数が指定されなければなりません。データはタスク関連ユーザー・ストレージに返されます。ストレージ保護をアクティブにすると、トランザクション定義の TASKDATAKEY オプションで指定されたキーのストレージにデータが返されます。ストレージは、ユーザーがトランザクション定義に指定した TASKDATALOC オプションに応じて、その境界線の上または下に配置されます。

RECEIVE MAP MAPPINGDEV コマンドが処理を正常に完了すると、ストレージ域が SET オプションによって返され、アプリケーションで制御されるようになります。このストレージ域は、アプリケーションがそれを FREEMAIN しない限り、トランザクションが終了するまで割り振られたままです。長時間実行するトランザクションの場合には、このようなストレージ域を FREEMAIN することが推奨されますが、タスクが終了すると、CICS はこれらの領域を解放します。

MAPPINGDEV アセンブラー・アプリケーションの例

866 ページの図 155 は、FILEA オペレーター命令サンプル・プログラムを修正した例であり、DFH$AGA という名前の同じマップ・セットを使用しています。

このアプリケーションは、MAPPINGDEV 機能に関連するキーワードのコード方法を分かりやすく説明するためだけのものであり、この機能をテストするための手段です。MAPPINGDEV 機能を使用するアプリケーションの設計としてお勧めするものではありません。

```
DFH$AMNX CSECT
*
         DFHREGS
DFHEISTG DSECT
OUTAREA  DS    0CL512
         DS    CL8
OUTLEN   DS    H
         DS    H
OUTDATA  DS    CL500
INLEN    DS    H
INAREA   DS    CL256
PROOF    DS    CL60
         COPY  DFH$AGA
         COPY  DFHBMSCA
DFH$AMNU CSECT
         EXEC CICS HANDLE AID PF3(PF3_ROUTINE)
*
         XC    DFH$AGAS(DFH$AGAL),DFH$AGAS
         MVC   MSGO(L'APPLMSG),APPLMSG
         EXEC  CICS SEND MAP('DFH$$AGA') FROM(DFH$AGAO) ERASE
               MAPPINGDEV(EIBTRMID) SET(R6)
         MVC   OUTAREA(256),0(R6)
         MVC   OUTAREA+256(256),256(R6)
         EXEC  CICS SEND TEXT MAPPED FROM(OUTDATA) LENGTH(OUTLEN)
*
         EXEC CICS RECEIVE INTO(INAREA) LENGTH(INLEN)
              MAXLENGTH(MAXLEN)
*
         EXEC CICS RECEIVE MAP('DFH$AGA') SET(R7) LENGTH(INLEN)
              MAPPINGDEV(EIBTRMID) FROM(INAREA)
              CURSOR(820) AID(=C'3')
*
         XC    PROOF,PROOF
         MVC   PROOF(25),=C'You just keyed in number '
         MVC   PROOF+25(6),KEYI-DFH$$AGAI(R7)

FINISH   DS    0H
         EXEC CICS SEND TEXT FROM(PROOF) LENGTH(60) ERASE FREEKB
         TM    MSGF-DFH$AGAI(R7),X'02'
         BNO   RETURN
         XC    PROOF,PROOF
         MVC   PROOF(33),=C'Input cursor located in MSG field'
         EXEC CICS SEND TEXT FROM(PROOF) LENGTH(60) ERASE FREEKB
*
*        THE RETURN COMMAND ENDS THE PROGRAM.
*
RETURN   DS    0H
         EXEC  CICS RETURN
*
PF3_ROUTINE DS 0H
         XC    PROOF,PROOF
         MVC   PROOF(30),=C'RECEIVE MAP specified AID(PF3)'
         B     FINISH
MAXLEN   DC    H'256'
APPLMSG  DC    C'This is a MAPPINGDEV application'
         END
```

図 155. MAPPINGDEV アプリケーションの ASM 例

第 70 章 区分画面サポート

区分画面は、BMS がサポートするいくつかの特殊ハードウェア機能のうちの最初のものです。区分画面には**標準** BMS が必要です。

IBM ディスプレイの中には、画面を複数の区域に分割して、それらの区域が独立した画面であるかのように個別に書き込んだり読み取ったりできるものがあります。このような区域を区分画面といい、この特殊ハードウェア機能を利用できる BMS の機能を総称して「区分画面サポート」といいます。

IBM 3290 ディスプレイは 3270 ファミリーのメンバーであり、IBM 8775 は区分化をサポートする装置の基本的な例です。区分化された装置のすべての機能を理解するにはその装置のマニュアルを参照する必要がありますが、その基本的な機能は、3290 の場合は「*IBM 3290 Information Display Panel Description and Reference*」に、8775 の場合は「*IBM 8775 Display Terminal Component Description*」に記載されています。

- 物理画面を、1 から 8 までの重なり合わない長方形の区域に分割し、任意の配置にすることができます。オペレーターが個別に消去できるという意味でこれらの区域は互いに独立し、キーボードの状態 (ロックまたはアンロック) はそれぞれ個別に保守され、一度に 1 つの区域に対して書き込みおよび読み取りができます。
- ある時点で**アクティブ**にできる区分画面は 1 つだけです。これは、カーソルを含む区分画面です。オペレーターはこの区分画面にのみ入力でき、カーソルは区分画面の境界で折り返します。データを伝送するキー (ENTER キーまたはプログラム・ファンクション・キーのいずれか 1 つ) を押すと、データはアクティブ区分画面からのみ伝送されます。
- オペレーターは「ジャンプ」キーの使用により、アクティブ区分画面を随時変更できます。プログラムでもこれを実行可能であり、872 ページの『アクティブ区分画面の判別』 に説明されています。
- BMS も指定された SEND 上で 1 つの区分画面にだけ書き込みを行いますが、ユーザーは複数の SEND を出すこともでき、アクティブ区分画面に書き込む必要はありません。
- 区分画面構成はデータ・ストリームとして装置に送信されるため、各新規タスクに対して、またはタスク内でも区分画面を変更することができます。区分画面を定義する BMS 構成を**区分画面セット**といい、869 ページの『区画定義』で説明しています。
- また、端末を**基本状態** (区分画面なし) で使用することもでき、区分画面配置を変更するために使用するのと同一のコマンドで、区分化状態から基本状態に切り替えることもできます。
- 画面区域の区分方法を指定するときは、その画面が実行されるハードウェア・バッファー空間も分割します。区分化した装置では、一般にバッファー容量が画面より大きいため、区分画面の中には余分なバッファー空間を割り当てられるものがあります。区分画面に割り振られた画面区域を、区分画面の**表示窓**といい、バッファー・ストレージを区分画面の**表示スペース**といいます。

© Copyright IBM Corp. 1989, 2009

BMS は表示スペースを区分画面のためのページ・サイズとして使用するため、一度にすべてが表示できない場合でも、適合可能な最大量のデータを送信することができます。装置に備わったキーにより、オペレーターは区分画面の表示窓を垂直にスクロールして、表示スペース全体を見ることができます。スクロールの作動には、ホストからの介入がまったくありません。

- 区分化された装置の中には、異なるサイズの文字セットから選択できるものがあります。これについては 870 ページの『3290 の文字サイズ』で説明します。

区分画面が独立しているにもかかわらず、ディスプレイは依然として CICS に対する単一の端末です。端末では、プリンシパル装置として一度に複数のタスクを実行することはできませんが、複数の疑似会話型トランザクション・シーケンスが同一の区分画面セットを使用する場合は、それらの間で画面空間を共同で使用できます (875 ページの『端末の共用』を参照)。

注: 3290 は、複数の論理装置 (LU) (CICS またはその他のシステムに対して) として動作するように内部的に構成することができます。この定義は、これらの論理端末のいずれの 1 つでも起こり得る区分化とは別のものです。

区分化画面の使用法

区分化画面は、一定のタイプのアプリケーションで特に便利です。例を以下に示します。

スクロール

単一画面におさまる以上の出力を生成するトランザクションの場合、スクロールが BMS 端末ページングの代替になります (814 ページの『出力後処理オプション: TERMINAL、SET、および PAGING』を参照)。例えば、1 つだけの区分から成る区分画面セットを定義できます。この場合、表示窓は画面全体で、表示スペースはバッファー全体です。バッファー全体に単一ページとして書き込むことができ、オペレーターは端末機能を使用してデータ全体をスクロールできます。ホストとの対話がないため、スクロール要求に対する応答時間はごく短くなります。バッファーの容量には当然制約されます。

画面の一部分だけをスクロールしたり、固定データのためにいくつかの区分画面を使用することもできます。

データ入力

区分化画面のもう 1 つの便利な使用方法は「ヘッド・ダウン」データ入力です。この場合オペレーターの生産性は、アプリケーションがどのくらい速く入力を処理し、次の入力のためにキーボードを再オープンできるかによって変わります。区分化画面を使用すると、画面を 2 つの同一入力画面に分割できます。オペレーターが 1 つに入力し、Enter を押してから 2 つ目に入力する一方で、データ入力トランザクションが最初の入力を処理します。入力が正しければ、プログラムは次の入力に備えて、それを消去します。正しくない場合でも、オペレーターが後続の作業を失うことなく訂正できる機会があります。「*CICS 4.1 Sample Applications Guide*」には、このようなデータ入力トランザクションの例が記載されています。

ルックアサイド

多くのオンライン操作では、オペレーターは進行中のトランザクションを終了するために 2 つ目のトランザクションを実行する必要があることがあります。受注入力が 1 つの例です。オペレーターは入力を完了するために、コードまたは価格を検索しなければならないことがあります。多くの問い合わせでも同様です。最初の問い合わせにより、問い合わせ対象の要約リストが返されます。オペレーターが 1 つを選択し、さらに詳細を要求すると、詳細のためにまた別のものを選択する必要があるといった場合があります。このような場合、区分化画面ではオペレーターが 2 つ目のタスクを行う間、後で必要になる最初の出力を画面に残しておくことができます。「*CICS 4.1 Sample Applications Guide*」には、ルックアサイド・トランザクションの例も記載されています。

「ヘルプ」テキストも、「ルックアサイド」の例の 1 つです。画面の 1 区画をこのテキストに割り振ると、オペレーターはメイン画面を消去せずに必要な学習情報を入手することができます。

データの比較

オペレーターが、2 つまたはそれ以上のデータのセットを同時に比較する必要のあるアプリケーションも、区分化画面を使用すると非常に便利な例です。区分化を行うと横並びの比較をすることができ、またスクロール機能により比較的大きな文書またはレコードの比較が可能になります。

エラー・メッセージ

画面を分割して、1 つの区域にエラー・メッセージおよびその他の説明テキストを割り振ると、オペレーターは常に同じ位置でメッセージを見られるため使用可能度が高まります。また、メイン画面の区域がこういった情報により上書きされることもありません。区分画面セットにこのような区分画面を指定すると、CICS は固有のメッセージをこの区分画面に送信します。このことについては、『区画定義』で説明します。

区画定義

画面の区分化はそれぞれ区分画面セットで定義されます。これは、画面上でともに表示するための、画面区域 (区分画面) の集合体です。

マップ・セットの場合と同様、アセンブラー・マクロで区分画面セットを定義します。これには DFHPSD および DFHPDI の 2 つがあります。

区分画面セット定義は DFHPSD (区分画面セット定義) マクロで始まり、次のものを定義します。

- 区分画面セットの名前
- 画面サイズ (BMS では、区分画面の表示窓が使用可能な空間の合計を超えないことが保証されます)
- デフォルト文字セル・サイズ (セル・サイズについては、 870 ページの『3290 の文字サイズ』で説明します)
- 区分画面セット接尾部。区分画面セットと特定の画面サイズを関連付けるために使用 (871 ページの『区分化の設定』を参照)。

開始 DFHPSD マクロの後に、各区分画面 (画面区域) を DFHPDI マクロで定義します。DFHPDI は次のものを指定します。

- 区分画面セット内部の区分画面の ID
- 画面上で区分画面を配置する場所
- 表示窓のサイズ (行および桁で指定)
- 表示窓に関連する表示スペース (すなわち割り振られたバッファー空間の合計) も、行および桁で指定します。スクロールが垂直方向のみなので、BMS では表示スペースの幅が表示窓の幅と一致しなければなりません。
- 使用する文字サイズ
- 区分画面に関連するマップ・セットの接尾部。区分画面サイズに適したマップ・セットを選択するために使用します。
- 区分画面が CICS エラー・メッセージを受信するかどうか (BMS は、生成する一定のエラー・メッセージを、受信するように指定された区分画面があれば、そこに送信します)。

2 つ目の DFHPSD マクロで区分画面セットを終了します。これには TYPE=FINAL オプションのみが含まれます。 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

これらはアセンブラー・マクロであるため、作成するときはアセンブラー形式設定規則に従う必要があります。アセンブラー言語についてよく知らない場合は、 785 ページの『BMS マクロの作成』を参照してください。その後で、区分画面セットのアセンブルおよびリンク・エディットを行う必要があります。その結果生成されるロード・モジュールは、マップ・セットと同じライブラリーに、またはご使用のシステムにとって望ましい場合には別のライブラリーに置くことができます。システム・スタッフが、PARTITION 定義を使用してシステムに各区分画面セットを定義する必要もあります。

3290 の文字サイズ

3290 ハードウェアでは、8 つまでの異なる文字セットを、異なるサイズで使用することができます。ハードウェアには 2 つのセットが付いていますが、他のセットは端末制御 SEND コマンドを使用してロードできます (詳しくは、「*IBM 3290 Information Display Panel Description and Reference*」を参照してください)。

各文字は画面上で長方形の**セル**を占有します。セル・サイズは画面に適合する行数および桁数を決めます。セル・サイズは区分画面ごとにも指定できるため、画面上の特定の区分画面についても同じです。セルはペル (画素) によって、垂直および水平に測ることができます。使用できる最小のセルは垂直が 12 ペルで水平が 6 ペルです。3290 画面は高さが 750 ペルで幅が 960 ペルです。したがって、最小セル・サイズを使用すると、垂直に 62 文字 (62 行)、水平に 160 文字 (160 桁) を指定できます (3290 は常にセル・サイズに最適の文字セットを選択し、文字をセルの最上部左隅に配置します)。

区分画面サイズは、その区分画面に指定するセル・サイズに基づいて行および桁で表し、セル・サイズはペルで表します (このオプションの名前は CHARSZE ですが、これは実際のセル・サイズです)。区分画面が確実に画面に適合するようにするためには、割り振りをペルで行う必要がありますが、BMS は区分画面が重なり合う

かまたは画面に適合しない場合、アセンブルのときに通知します。区分画面の高さは、区分画面の行数と垂直 CHARSZE ディメンションの積です。区分画面の幅は、桁数と水平 CHARSZE の積です。

DFHPDI 区分画面定義に CHARSZE サイズを指定しないと、BMS は DFHPSD 区分画面セット定義に指定されたデフォルトを使用します。DFHPSD にも CHARSZE が指定されない場合、BMS はインストール時に端末に設定されたデフォルトを使用します。区分画面について、セル・サイズを指定するものもあるが、すべての区分画面には指定しない場合、区分画面セットにもデフォルトを指定しなくてはなりません。そうすると、選択したものとインストール・デフォルトとが混同されません。

区分化の設定

TRANSACTION 定義の PARTITIONSET オプションに名前を指定することにより、特定のトランザクションに対してロードする区分画面セットを BMS に指示することができます。これを行った場合、指定された区分画面セットがまだ端末でロードされていない時は、BMS は、区分画面セット定義をタスクの最初の BMS SEND のデータに追加します。

また、BMS が区分画面を現在の状態を変更しないように指示したり (TRANSACTION 定義において PARTITIONSET=KEEP)、自分自身で区分画面をロードすることを指示することもできます (PARTITIONSET=OWN)。PARTITIONSET 値をまったく指定しない場合、BMS はトランザクション開始時に、端末を基本状態 (区分画面なし) に設定します。

トランザクションに関連した PARTITIONSET 値に関係なく、タスクはほとんどいつでも SEND PARTNSET コマンドにより新規の区分画面を設定できますが、例外として、論理メッセージを作成する間にはコマンドを発行できません。

SEND PARTNSET は、端末に即時には何も送信しません。その代わりに BMS は、データまたは制御情報を送信する次の BMS コマンドと一緒に、区分画面情報を送信することを保管します。これは、最初の BMS SEND 上にある TRANSACTION 定義の PARTITIONSET オプションに指定された区分画面セットを送信する場合と同じです。結果として、新規区分画面の影響を受ける RECEIVE または RECEIVE MAP を出す前に、SEND MAP、SEND TEXT または SEND CONTROL コマンドを出さなくてはなりません。

注: 次の状態では区分画面の予期しない変更が起こることがあります。すなわち、CICS がエラー・メッセージを端末に送信する必要があり、現区分画面の設定にエラー区分画面が組み込まれていない場合、CICS は端末を基本状態に返し、画面を消去してメッセージを書き込みます。この理由から、すべての区分画面セットにエラー・メッセージに適した区分画面を 1 つ指定するといいでしょう。

BMS は区分画面セットをロードする時、要求される名前に端末タイプを表す文字の接尾部を付けますが (装置依存のサポートが有効な場合)、これは端末に適したものをロードするためです。接尾部は、端末に関連した TYPETERM 定義の ALTSUFFIX オプション値から取られます。区分画面セットの接尾部付けはマッ

プ・セットの場合と類似しており、正しい接尾部を持つ区分画面セットがない場合は、ステップが同一のシーケンスで行われます (797 ページの『装置依存マップ』を参照)。

BMS SEND コマンドの区分画面オプション

前述したように、区分化された画面に書き込むときは、書き込む区分画面は 1 つだけで、コマンドの影響はその区分画面だけに限定されます。ERASE および ERASEAUP はその区分画面の内部のみを消去し、FREEKB はその区分画面がアクティブになるときのみキーボードをアンロックします。

マップ定義の PARTN オプション、または SEND MAP の OUTPARTN オプションのいずれかにより、送信する区分画面を指定することができます。OUTPARTN は PARTN を指定変更します。いずれも指定しない場合、BMS はセットの最初の区分画面を選択します。

区分画面の使用は、797 ページの『装置依存マップ』で述べたマップ・セット名の接尾部付けに影響します。マップ・セットの接尾部は、そのセクションで述べたように決定される代わりに、区分画面のための MAPSFX 値から取られます。

アクティブ区分画面の判別

区分画面を送信するときは、カーソルをその区分画面か、または別の区分画面に移動できます。マップ定義の PARTN オプションにある ACTIVATE の値によって、書き込む区分画面にカーソルが置かれます。BMS SEND コマンドに ACTPARTN を指定すると、任意の区分画面の名前を指定することができ (書き込んでいるものである必要はありません)、ACTIVATE の指定が上書きされます。ACTIVATE および ACTPARTN はアクティブ区分画面に対し、カーソルを配置するとともにキーボードもアンロックします。いずれも指定されない場合、カーソルは移動せず、キーボードもアンロックされません。

区分画面を送信するとき、そこにカーソルを配置することにより区分画面をアクティブにできますが、これが唯一の方法という訳ではありません。オペレーターが端末でジャンプ・キーを使用して、カーソルを別の区分画面に移動することができるためです。これは端末からのデータの再受信を複雑にすることがありますが、BMS にはこのためのヘルプがあります。以下ではこのヘルプについて説明します。

BMS RECEIVE コマンドの区分画面オプション

RECEIVE MAP コマンドを発行する場合は、マップ定義の PARTN オプション、または RECEIVE MAP の INPARTN オプションのいずれかを使用して、データを入手する区分画面 (すなわちアクティブにする区分画面) を BMS に指示できます。INPARTN は PARTN を指定変更します。この指定を行った場合、指定したものと異なる区分画面からオペレーターが伝送したときは、BMS は、指定した区分画面にカーソルを再配置し、キーボードをアンロックして RECEIVE コマンドを繰り返します。また、エラー区分画面 (ATTRB=ERROR になっているもの) にメッセージを送信し、オペレーターに正しい区分画面を使用するように指示します (エラー区分画面がない場合はメッセージは送信されません)。正しくない区分画面からの入力は

廃棄されますが、後で再読み取りできるため、失われることはありません。BMS ではこれを 3 回まで行います。オペレーターが 4 回目まで続けて行うと、BMS は PARTNFAIL 条件を起こします。

入力区分画面を指定する必要はありません。入力できる区分画面が 1 つだけのこともあり、同一マップがすべてに適用されることもあります。INPARTN なしで RECEIVE MAP を出し、マップに PARTN オプションがない場合、BMS はいずれの区分画面からもデータを受け入れ、それをコマンドで指定した名前のマップを使用してマップします。また、後に必要であれば、その区分画面を INPARTN オプションを含む ASSIGN コマンドで判別することができます。

ただし、INPARTN は最初の BMS 操作の後まで設定されないため、正しいマップを選択するために送信する区分画面を判別する必要がある場合は、別の方法が必要です。この状況では RECEIVE PARTN コマンドを出すと、マップされていないデータを読み取り、それを送信した区分画面を知ることができます。次に、その区分画面と FROM オプションとを突き合わせるマップを使用する RECEIVE MAP コマンドを出し、その区分画面に一致するマップを使用します。FROM を伴う RECEIVE MAP は、既に読み取られたデータをマップします。 830 ページの『その他の入力の形式設定』に説明があります。

区分画面の ASSIGN オプション

今説明した INPARTN オプションに加え、他に 3 つの ASSIGN オプションがあり、区分化された端末のためのプログラミングに便利です。PARTNS オプションはタスクに関連した端末が区分画面をサポートするかどうかを示し、PARTNSET オプションは現区分画面セットの名前 (設定されていなければブランク) を返します。4 つ目の ASSIGN オプションである PARTNPAGE は、論理メッセージにのみ適用されます。これについて 『区分画面と論理メッセージ』で述べます。

区分画面と論理メッセージ

BMS 論理メッセージを、区分画面が設定された端末のために作成する場合、メッセージのページを複数の区分画面に向けることができます。テキスト出力をある区分画面に、マップ出力を他の区分画面に送信することもできますが、これは同一区分画面でそれらを混合しない場合です (このことは、論理メッセージにおけるテキスト出力とマップ出力の混合に対する通常の規則から見ると例外です)。

出力が表示されると、各区分画面の最初のページがまず表示されます。ページは区分画面によって番号が付けられ、オペレーターが特定の区分画面に入力する CSPG コマンドは、その区分画面にのみ適用されます。ただし、ページ除去コマンドは例外です。この除去コマンドは、すべての区分画面からすべての論理メッセージを削除します。

メッセージにかかわる各 BMS SEND に、出力を送信する区分画面を指定します。ACCUM を使用しない場合、BMS はその区分画面のためにページを作成します。ACCUM を使用する場合、BMS はその区分画面のための現行ページに出力を送信します。このため、区分画面ごとにページ・オーバーフローが起こります。ページ・オーバーフローを代行受信しており、ページ・オーバーフローの起こった区分画面がどれかわからない場合、ASSIGN コマンドの PARTNPAGE オプションを使用して検出することができます。

注: BMS は、論理メッセージの作成のときにページ・サイズと区分画面 ID の両方を使用するため、メッセージの途中で区分画面を変更することはできません。

区分画面の中でページを配布するとき、ページ・オーバーフローを処理するために必要な記帳機能は、ルーティング環境において必要なものと類似しています (861 ページの『ルーティングとページ・オーバーフロー』を参照)。特に、ある区分画面のためのページ・オーバーフロー処理は、他でページ・オーバーフローを起こすことが考えられるような作業をする前に、確実に終了させておく必要があります。そのようにしないで障害が起こった場合、誤った出力になるだけでなく、プログラム・ループが起こることもあります。

区分画面とルーティング

複数の区分画面に書き込まれた論理メッセージはルーティングできません。BMS は、ルーティング環境では BMS SEND コマンドの OUTPARTN オプションおよび ACTPARTN オプションを無視します。

通常のメッセージを、区分画面をサポートする端末にルーティングすることはできますが、BMS がこのメッセージを作成し、CSPG トランザクションが基本状態 (未区分の状態) の端末にこのメッセージを表示します。

また、区分画面と論理装置コードを一緒に使用することはできません (LDC については 877 ページの『論理デバイス・コンポーネント』 に説明があります)。さらに、区分画面を GDDM との組み合わせで使用することもできませんが、区分画面を外部形式制御とともに使用することはできます (883 ページの『外部形式制御』を参照)。

アテンション ID および例外条件

区分化された端末には、CLEAR キーが画面全体を消去するのと同じ方法でアクティブ区分画面を消去する CLEAR PARTITION キーがあります (CLEAR キーは、区分化された端末でも画面全体を消去します)。プログラム・ロジックで、この追加のアテンション ID を検査する必要がある場合もあります。CLEAR PARTITION AID 値は DFHAID に組み込まれています (824 ページの『アテンション ID の使用』を参照)。

また、区分画面に関連した新たな例外条件もいくつかあり、以前からの例外条件のいくつかが当てはまる新たな状態もあります。新たな例外条件には、INVPARTN (区分画面セットにない区分画面を指定する)、INVPARTNSET (区分画面セットではないモジュールを指定する)、および PARTNFAIL (オペレーターが伝送に使用したもの以外の区分画面から受信する) があります。これらはすべて、適用されるコマンドとともに「*CICS アプリケーション・プログラミング・リファレンス*」に説明されています。

端末の共用

適切な計画を行えば、それぞれに別々の区分画面を割り当てることにより、複数の処理の間で 1 つの端末を共用することができます。1 つの端末で一度に複数のタスクを進行することは当然できませんが、区分化された端末で複数の疑似会話型トランザクション・シーケンスのコンポーネント・タスクをインターリーブすることはできます。

非常に簡単な例を示すため、オペレーターに 2 つの区分画面にデータを入力させることにより、既存の疑似会話型データ入力トランザクションの応答時間を向上する場合を想定します。(868 ページの『データ入力』 を参照)。この場合一度に 2 つのレコードで作業するようにアプリケーションを修正できます。または、入力を入手したのと同じ区分画面に送信するように修正することもできます。次に、それぞれの区分画面から独立して実行できます。

TRANSACTION 定義の PARTITIONSET オプションを使用して区分画面を設定することができます (シーケンスの中に複数のトランザクションがある場合、そのすべてが関連します)。前述したように、BMS は各トランザクションが同一の PARTITIONSET 値を持つ限り、区分画面を再ロードしません。代わりに、予備トランザクション (例えば、両方の区分画面の最初の項目画面を表示したトランザクション) を用いて区分画面を設定し、データ項目トランザクションに対して KEEP の PARTITIONSET 値を使用することができます。類似のトランザクション同士であっても異なるものであっても、区分化された画面を共用するときは必ず、誰かが、他で必要とされている区分画面セットを破棄しないようにする必要があります。また、2 つの異なる CICS システムが同一画面を共用するような場合は、区分画面セットに共通の名前を付ける必要があります。そうすると BMS は、不要な場合には区分画面を再ロードしません。

仮のデータ入力トランザクション・シーケンスが RETURN コマンドの TRANSID オプションを使用して次のトランザクション ID を指定する場合、さらに変更を少し加える必要があります。このオプションはその区分画面ではなく、端末全体に適用されるためです。1 つの解決策として、次のトランザクション ID を画面上の最初のフィールドに配置し (フィールド定義の修正データ・タグがオンになります)、RETURN から TRANSID を除去します。すると CICS は、244 ページの『タスクの開始方法』 の説明にあるように入力から次のトランザクションを判別します。

第 71 章 特殊ハードウェアのサポート

区分画面に加え、BMS では次に挙げる他の特殊ハードウェア機能もサポートします。

- 論理デバイス・コンポーネント
- 10/63 磁気スロット読み取り装置
- フィールド選択機能 (カーソル選択、ライト・ペン、トリガー・フィールド)
- 外部形式制御

磁気スロット読み取り装置および外部形式制御は、ともに**標準** BMS が必須です。カーソル選択キー、ライト・ペンおよびトリガー・フィールドのサポートは、**最小** BMS に組み込まれています。

論理デバイス・コンポーネント

論理デバイス・コンポーネント (LDC) は、BMS がサポートする別の特殊ハードウェア機能です。区分画面と同様、LDC にも**標準** BMS が必須です。

LDC をサポートする端末は、単一のポイント (論理装置 (LU)) を介して制御される複数の機能コンポーネント (論理デバイス) で構成されています。このコンポーネントは、リモート・ワークステーションの代表であるプリンター、読み取り装置、キーボードおよびディスプレイであることもあり、またワード処理端末や通帳プリンターといったデバイスのように複数であることもあります。IBM 3601 論理装置、3770 バッチ論理装置、3770、および 3790 バッチ・データ交換論理装置、および LU タイプ 4 論理装置はすべて、論理デバイス・コンポーネントをサポートします。

論理装置 (LU) は、CICS にとって単一のエンティティーではあっても、独立して読み書きできるコンポーネントで構成されているため、LDC 端末用の CICS アプリケーション・プログラム・インターフェースは、区分化された端末用のものと同じように見えます。各 LDC は区分画面セットの 1 つの区分画面に対応しています。異なる点も当然多くあり、特定の端末タイプについては CICS サポートについて解説してある CICS をお読みください。次のセクションでは、プログラミングに影響する主な違いについて説明します。それらは、以下のとおりです。

- LDC 定義
- SEND コマンド・オプション
- 論理メッセージ
- ルーティング

論理デバイス・コンポーネントの定義

端末のための論理デバイス・コンポーネントは、LDC 表と呼ばれるリストによって定義されます。TERMINAL 定義の TYPETERM コンポーネントがこの表を示します。論理装置に対して個別である場合も、同一のコンポーネントを持つ複数の論理装置が共有する場合もあります。

© Copyright IBM Corp. 1989, 2009

表そのものは DFHTCT TYPE=LDC (端末制御) マクロによって定義されます両方のマクロについての説明は、「 *CICS Resource Definition Guide*」の を参照してください。

LDC 表は、論理装置 (LU) の各論理デバイス・コンポーネントに関する次の情報を含みます。

- 2 文字の論理デバイス ID。これらの ID は通常、CO がコンソールの略、MS が磁気ストライプ・エンコーダーの略といったように、標準の省略語ですが、そうである必要はありません。
- 1 文字の装置コード。装置タイプ (コンソール、カード読み取り装置、ワード・プロセッシング・ステーション) を示します。コードは CICS によって装置タイプから割り当てられ、その他の情報はマクロで提供されます。
- BMS ページ・サイズ。BMS は、論理装置に関連したサイズではなく、このサイズを使用します。異なる論理デバイスでは、ページ・サイズも異なるためです。
- BMS ページ状況 (AUTOPAGE または NOAUTOPAGE)。 836 ページの『AUTOPAGE オプション』を参照してください。

論理デバイス・コンポーネントへのデータの送信

BMS 出力を、SEND MAP、SEND TEXT、または SEND CONTROL コマンドの LDC オプション、またはマップ・セットの LDC オプションに指定することにより、端末の特定の論理デバイス・コンポーネントに送信します。このコマンドの値は、マップ・セットの値を指定変更します。LDC がどこにも指定されていない場合、BMS は端末タイプによって異なるデフォルトを使用します。

LDC および論理メッセージ

自分自身の端末用に BMS 論理メッセージを作成する場合、単一の論理メッセージのページを異なる区分画面に送信するのと同じ方法で、メッセージのページを異なる論理デバイス・コンポーネント間で配布することができます。BMS は、区分画面の場合と同じ方法で、各論理デバイス・コンポーネントに対して個別にページを累積します (873 ページの『区分画面と論理メッセージ』を参照)。メッセージにはテキストとマップ出力の両方を組み込むことができますが、これはその両方を 1 つの LDC に送信するのではない場合です。 LDC によってページ・オーバーフローが起こり、端末オペレーター・ページング・コマンドは論理デバイス・コンポーネントをベースとして動作します。

ページを取り出すときは、そのオペレーター (またはデバイス制御プログラムのユーザー・コード) が、要求が適用される LDC を指示しなくてはなりません。キーボードがない装置もあるためです。区分画面の場合と同様、メッセージ除去要求はメッセージ全体を、すべての LDC から除去します。論理デバイスに対するページの取得について詳しくは、「*CICS Supplied Transactions*」のCSPG - ページの取得を参照してください。

ページ・オーバーフローを代行受信する場合は、LDCMNEM オプションまたは LDCNUM オプションのいずれかを指定して ASSIGN コマンドを出すことにより、ページ・オーバーフローが起こった LDC を判別することができます。どちらも、オーバーフローを起こした装置を識別しますが、LDCMNEM は 2 文字の名前によ

って、LDCNUM は 1 バイトの数字 ID によって識別します。区分化された装置の場合と同じく、ASSIGN PAGENUM によってページ・オーバーフローが起こった装置のページ番号を判別できます。

LDC およびページ・オーバーフローに関連した制約事項が 1 つあり、これは LDC に固有のものです。ページ・オーバーフローが起こった後で、現行ページのトレーラー・マップと、次ページのヘッダーの両方を、ページ・オーバーフローが起こった LDC に送信しなくてはなりません。これに失敗すると、BMS は INVREQ (無効要求) 条件を起こします。

LDC とルーティング

LDC 環境ではルーティングがサポートされていますが、この場合メッセージは、LDC をサポートするすべての宛先に対し、同一のコンポーネント・タイプに送信されます (複数の LDC メッセージはルーティングできません)。

LDC 値はいくつかの方法で指定できます。

- ROUTE コマンドの LDC オプションを使用する場合、指定する値が他のすべてのソースを指定変更し、LDC が適用されるすべての適格な宛先に対して使用されます。
- LDC を (ROUTE コマンドには指定せず) 経路リスト項目に指定する場合、その値は関連した宛先に対して使用されます (両方を指定してそれらが一致しない場合、「ROUTE」リストの値が使用され、この矛盾が項目の状況フラグに示されます)。
- どちらも指定しない場合、BMS SEND コマンドから LDC を省略すると、端末とシステム LDC 表から、非ルーティング環境の場合と同じ方法で値が判別されます (SEND コマンドの値はルーティングが有効な場合無視されます)。

10/63 磁気スロット読み取り装置

IBM ディスプレイ端末の中には、オプション機能として磁気スロット読み取り装置 (MSR) をサポートするものがあります。MRS とは、小さい磁気カードからデータを読み取る装置です。MSR には、オペレーターの処置を促すための表示ライトと音響アラームがあります。端末が MSR を制御するものもありますが、IBM 8775 や IBM 3643 などでは、プログラムが読み取り装置の機能を制御します。

CICS は MSR という ASSIGN コマンド・オプションを提供し、タスクのプリンシパル装置に MSR があるかないかを示します。

BMS では、BMS SEND コマンドの MSR オプションを使用することにより、このような MSR の状態を制御できます。このオプションにより、端末に送信された表示データに加え、4 バイトの制御データが、接続された MSR に伝送されます。BMS には DFHMSRCA というコピーブックがあり、これには必要なほとんどの制御シーケンスが含まれます。「*CICS アプリケーション・プログラミング・リファレンス*」には、提供されている定数が示され、制御データの構造が説明されているため、必要に応じてそのリストを拡張できます。

MSR に送信する制御シーケンスは、装置からの次の入力に影響します。したがって、RECEIVE コマンドが出されるまでは有効になりません。MSR からの入力は、キーボード入力と同じ方法で装置バッファーに配置され、伝送されます。MSR 入力

の伝送が行われると、EIBAID のアテンション ID を調べることによりこれを検出できます。 X'E6' の値は MSR からの入力を示し、X'E7' の値は拡張 MSR (使用可能である場合の 2 番目の MSR) からの入力を示します。MSR 入力の画面の形式設定方法に関する情報、および装置のその他の詳細については、「*IBM 3270 Information Display System Data Stream Programmer's Reference*」を参照してください。

フィールド選択の機能

以下のような BMS がサポートするいくつかの特殊ハードウェア機能により、オペレーターは、画面でフィールドを選択することによって情報の入力および伝送ができます。

- トリガー・フィールド
- カーソル選択可能フィールド
- ライト・ペン検出

トリガー・フィールドのサポート

トリガー・フィールドは、8775 など一定のタイプの端末の特殊ハードウェア機能です。トリガー・フィールドとして定義されたフィールドにより、オペレーターが**プライム状態**のときのフィールドから外にカーソルを移動すると、端末はフィールドの内容を伝送します。フィールドは、オペレーターがそのフィールドにカーソルを移動し、データを入力するか、DELETE キーまたは ERASE EOF キーのいずれかを使用するときプライム状態になります。フィールドによって伝送が行われた後、またはオペレーターが ERASE INPUT キーを使用した場合、または端末への送信の後に、そのフィールドはプライム状態ではなくなります (区分画面を使用する場合は、これが有効であるために、送信はトリガー・フィールドを含む区分画面に送られなくてはなりません)。

VALIDN 拡張属性を TRIGGER の値に設定することにより、フィールドをトリガー・フィールドとして定義します。マップで、またはプログラムの指定変更のいずれかにより、これを行います。

トリガー・フィールドが伝送を行うときは、フィールドそのものだけが送信されます。他のフィールドは、修正されていた場合でも送信されません。トリガー・フィールドによって行われた伝送は、アテンション ID を検査することにより検出できます。この ID の値は X'7F' です。

妥当性検査機能をサポートする端末はキーボードのバッファーとして動作するため、オペレーターはホストが先に行われた伝送を処理する間データ入力を継続することができます。オペレーターがバッファー容量を超えず、また先に起こったエラーが診断される前に大量のデータを入力しないよう、このような入力を処理するプログラムでは迅速な応答を必要とします。

通常のプロシージャーは、入力を受信するプログラムが、トリガー・フィールドの内容を即時に検査します。正しければ、プログラムはオペレーターが継続できるようにキーボードをアンロックします (FREEKB オプションを含む BMS SEND コマ

ンドがこれを行います)。フィールドが誤っている場合は、診断メッセージを送信するとともに、保管されたキー・ストロークを廃棄することができます。このためには、次の処置のいずれかを行います。

- ERASE、ERASEAUP、または ACTPARTN を含む BMS SEND コマンドか、または FREEKB を含まない BMS SEND コマンド
- トリガー・フィールドを含むもの (使用中の区分画面) 以外の区分画面に送信される BMS SEND
- RECEIVE MAP、RECEIVE PARTITION または端末制御 RECEIVE コマンド
- タスクの終了

トリガー・フィールドについて詳しくは、「*IBM 3270 Information Display System Data Stream Programmer's Reference*」を参照してください。

カーソルおよびペンで検出可能なフィールド

BMS は**検出可能**フィールドもサポートします。これは、いくつかの端末で利用可能な別の特殊ハードウェア機能です。検出可能フィールドには、「カーソル選択」キーおよびライト・ペンの、2 つのハードウェア機構があります。端末にはキーまたはペンのいずれかがあり、両方はありません。両方とも作用は同じですが、キーはペンの後継なので、キーについて説明します。

フィールドを検出可能にするためには、一定のフィールド属性がなくてはならず、**指定機能文字**と呼ばれるデータの先頭文字が、5 つの特定の値のいずれか 1 つを含んでいなくてはなりません。必要な場合は、指定機能文字の後に他の表示データを加えることができます。

検出可能性を管理するフィールド属性バイトのビットは、輝度も制御します。指定機能文字が検出可能な値のいずれか 1 つである場合、高輝度 (ATTRB=BRT) フィールドは検出可能です。通常輝度フィールドは検出可能にも、検出不能にもできます。検出可能にするためには、ATTRB=DET と指定する必要があります。非表示 (ATTRB=DRK) フィールドは検出可能にはできません。

この場合も、属性および指定機能文字をマップ定義に、またはプログラムによる指定変更のいずれかにより指定することができます。ただし、DET は入力専用マップに使われると特殊な効果があります。これについてはすぐに説明します。

高輝度フィールドは定義によって、検出可能性に対して適切なフィールド属性を持っているため、端末オペレーターは指定機能文字をフィールドの最初の位置に入力することにより、*無保護* 高輝度フィールドを検出可能にできることに注意してください。

選択フィールド

検出可能フィールドには、**選択**フィールドおよび**アテンション**・フィールドの 2 つのタイプがあります。このタイプは指定機能文字により管理されます。選択フィールドは、疑問符 (?) またはより大記号 (>) の、いずれかの指定機能文字によって定義されます。規則では、(?) はオペレーターがフィールドが何を表すかに関係なく選択しなかったことを意味し、(>) は選択したことを意味します。ハードウェアはこの規則に基づいて設計されていますが、強制ではないので適したものであれば別の規

則を適用できます。指定機能文字をいずれの値にも初期設定できます。また、いずれの値も、変更データ・タグをオフまたはオンに初期設定するのに使用できます。

カーソルが選択フィールド内にあるときに、オペレーターがカーソル選択キーを押すたびに、指定機能文字がある値から他の値に切り替わります (? から > に、> から ? に変わります)。直前の状態とはかかわりなく、指定機能文字が ? から > に変わると MDT はオン になり、指定機能文字が > から ? に変わるとオフ になります。これにより、オペレーターは既に選択したフィールドを変更することができ (選択したフィールドの下で再度カーソルを押す)、また MDT の状況を最大限に制御できます。他のフィールドの場合と同様、MDT は、伝送が行われるときこのフィールドが組み込まれるかどうかを管理します。ただし、この時は伝送は行われません。選択フィールドそのものは伝送を行わないためです。この伝送を行うのが、アテンション・フィールドの目的です。

アテンション・フィールド

アテンション・フィールドは、ブランク、ヌル、または & 記号の指定文字によって定義されます。データ・ストリーム内でヌルを使用すると、この機能でブランクを使った場合と同じ結果になります。ただし、場合によっては BMS がヌルを伝送せず、フィールドの最初の位置をヌルで上書きできないため、BMS ではブランクを使用する必要があります (811 ページの『値の取得元』を参照)。選択フィールドと違い、アテンション・フィールドのカーソルでカーソル選択キーを押すと、伝送が行われます。

指定機能文字がアンパーサンドの場合、カーソル選択キーを押した結果は ENTER キーを押した結果と同じです。ただし、指定機能文字がブランクまたはヌルの場合、伝送されるものは MDT がオンであるすべてのフィールドのアドレス、カーソルの位置、および X'7E' のアテンション ID です。これらのフィールドの内容 は伝送されませんが、ENTER キー (またはアンパーサンドの指定機能文字によるカーソル選択) の場合は伝送されます。いずれの場合も、MDT ビットがオンのフィールドは、オペレーターが変更したかまたは MDT がオンの状態で送信された、選択フィールドまたは通常のフィールドであると考えられます。

検出可能フィールドからの BMS 入力

指定機能文字がブランクまたはヌルのカーソル選択アテンション・フィールドにより伝送が行われると、BMS は対応する入力 (I) サブフィールドの最初の位置を X'FF' に設定することにより、どのフィールドが伝送されたか (どのフィールドで MDT がオンになったか) を示します。それ以外の場合、最初の位置は X'00' に設定されます。伝送が行われたアテンション・フィールドが 1 つだけの場合は、この値からそのアテンション・フィールドを知ることができます。それ以外の場合はカーソルの位置から知ることができます。

アンパーサンド指定子付きのカーソル選択アテンション・フィールドにより (または ENTER キーか PF キーにより) 伝送が行われると、MDT がオンで L サブフィールドがその長さを反映していれば、通常、I サブフィールドにはそのフィールドの内容が収容されます。ただし、入力マップ内のフィールドに DET 属性が指定されている場合 (すなわち、MODE=IN、MODE=INOUT、または DATA=FIELD の場合) は除きます。RECEIVE MAP がこのようなマップに名前を指定した後、この I サブフィールドはそのフィールドが選択される場合 (その MDT がオンであった場

合) は長さが 1 の X'FF' を、選択されない場合はヌル (X'00') を含みます。BMS は、何かが伝送された場合でも、フィールドに対するその他の入力を提供しません。

そのため、検出可能フィールドが選択されたかどうか確認するだけでなく、そこからデータを受信する必要がある場合、入力マップでは DET の使用を避ける必要があります。入力と出力で別々のマップを使用し、出力マップにのみ DET 属性を指定するか、またはマップ内ではなく、プログラムによって送信されるデータ・ストリーム内に DET 属性を設定することができます。高輝度フィールドの場合は、BRT が DET を暗黙指定するため DET を指定する必要はありません。BMS は、入力マップの BRT に指定された各フィールドのデータを返します。

データが伝送されることを確認する必要もあります。伝送が ENTER キー、PF キー、または指定機能文字がアンパーサンドのアテンション・フィールドによって行われる場合は、フィールド・データは伝送されます。指定機能文字がブランクまたはヌルのアテンション・フィールドによって行われる場合は、伝送されません。

検出可能フィールドについて詳しくは、「*IBM 3270 Information Display System Data Stream Programmer's Reference*」マニュアルを参照してください。

外部形式制御

外部形式制御は、ホスト・プロセッサーと接続されたサブシステムの間の回線通信量を削減する技法です。この削減は、ネットワーク経由で変数データしか送信しないことによって行います。このデータは、サブシステム内のプログラムによって、物理マップなどの定数データと結合されます。その後で、形式設定データを表示することができます。

外部形式制御は、3650 ホスト通信論理装置、DPPX および DPS バージョン 2 を持つ 8100 シリーズ・プロセッサー、または 3174 制御装置を介して接続された端末とともに使用することができます。3650 によって使用されるマップは、使用する前に、3650 変換定義言語を使用して再定義しなければなりません。8100 によって使用されるマップは、SDF II のユーティリティー、または DPS バージョン 2 の対話式マップ定義コンポーネントのいずれかを使用して、8100 上に生成しなくてはなりません。

ホスト・プロセッサーのプログラムが大量のマップされたデータをサブシステムに送信した場合には、マップ内の変数データしか伝送しないことを BMS に指示することによって、回線通信量が削減されることがあります。データを受け取った時に、サブシステムはマッピング操作を実行しなければなりません。BMS は、データを形式設定するために使用するサブシステム・マップを識別する情報を使用して、変数データに接頭部を付加します。

外部形式制御をサポートする端末は、TYPETERM 定義に OBFORMAT(YES) が指定されています。プログラムがこのような端末に対して SEND MAP コマンドを出し、指定されたマップ定義に OBFMT=YES が含まれている場合、BMS はサブシステムがデータを形式設定しようとしていると想定し、適切なデータ・ストリームを生成します。OBFMT=YES の指定があるマップを、外部形式制御をサポートしない端末に送信した場合、BMS はその OBFMT オペランドを無視します。

外部形式制御をサポートするいくつかの装置のプログラミングの詳細は、 617 ページの『第 42 章 バッチ・データ交換の使用』を参照してください。

第 72 章 BMS: パフォーマンスの設計

基本マッピング・サポート (BMS) によって定様式データ・ストリームを作成する場合には、ここで説明する事項を念頭に置く必要があります。

変更データ・タグ (MDT) の不要な調整の回避

MDT は、READ MODIFIED コマンド (CICS がコピー操作以外のすべてに使用するコマンド) で、フィールドを伝送するかどうかを判別する属性バイトのビットです。

フィールドの MDT は、通常、ユーザーがデータをフィールドに入力したときに、3270 ハードウェアによってオンにされます。しかし、マップを画面に送るときに、マップに FSET を指定するか、あるいはオンのタグを持つ上書き属性バイトを送ることによって、オンにすることもできます。マップの中の定数のフィールド、またはラベルを持たない (マップを受け取るプログラムに送られない) フィールドには、この方法でタグをオンに設定してはいけません。

また、通常、入力フィールドに FSET を指定する必要はありません。この理由は、既に言及しているとおり、ユーザーがデータを入力するすべてのフィールドの MDT は自動的にオンになるためです。これは、次の RECEIVE コマンド内に組み込まれます。たとえ何回画面を送っても、プログラムで (FRSET、ERASEAUP、または ERASE オプションによって、あるいはオフのタグで属性を上書き変更することによって) 明示的に**オフ** にするまで、これらのタグはオンのままです。

入力から入力までの間の、ユーザーが画面入力しなかった情報を保管することができます。プログラムで MDT をオンにするのはこのためです。しかし、伝送オーバーヘッドがともなうので、このストレージ技法が適切なのは、少量データの場合だけで、リモート端末の場合よりローカル端末の場合に適切です。例えば、この技法が特に有用なのは、入力フィールドにデフォルト値を保管する場合です。一部のアプリケーションでは、ユーザーは、一部のフィールドに既にデフォルト値が入っている画面を完成する必要があります。デフォルトを変更したくないユーザーは、そのフィールドをスキップするだけです。入力を処理するプログラムには、これらのデフォルトの内容を通知する必要があります。デフォルトが常時同じ場合には、プログラムで定数として指定することができます。しかし、デフォルトが可変で、前の入力によって異なる場合には、画面を書き込むマップで FSET を使用して MDT をオンにすることによって、前の値を画面に保管できます。その後で、画面を読み取るプログラムは、ユーザーが変更しないフィールドからデフォルト値を、ユーザーが変更したフィールドから新規の値を受け取ります。

注: CLEAR、PA1、PA2、または PA3 キーを押した場合、保管された値は画面には返されません。

FRSET を使用したインバウンド・トラフィックの削減

複数回読み取る必要がある、入力フィールドの多い画面がある場合には、次の読み取りの準備で、画面を書き戻すとき、FRSET を指定することによって、入力デー

© Copyright IBM Corp. 1989, 2009

タ・ストリームの長さを削減することができます。FRSET は MDT をオフにするので、その書き込みの前に入力されるフィールドは、次回ユーザーがフィールドに再入力しない限り存在していません。比較的いっぱいの画面、およびいくつかエラーが出る (あるいはその他の理由で伝送を繰り返す) 可能性がある処理を取り扱う場合には、これは大幅な節約になります。しかし、後続する読み取りでは***変更済みの*** フィールドしか送信されないので、プログラムで各サイクルごとに入力を保管し、新規データを古いデータと組み合わせなければなりません。FRSET を使用していない場合には、MDT はオンになったままで、入力時点とは無関係にすべてのフィールドが送信されるので、この処理は不要です。

画面へのブランクのフィールドの送信の禁止

全体がブランクになっているフィールドか、あるいは右側に後書きブランクを埋めるフィールドを画面に送ると、通常、回線容量の無駄になります。BMS がこれを行うようにユーザーに要求するのは、画面の他の部分を変更しないで、現在データを含んでいる画面上のフィールドを消去する必要があるか、あるいはそのフィールドを、現在の画面上のデータより短いデータで置き換える必要がある場合だけです。

この理由は、BMS がユーザーのマップを表すデータ・ストリームを作成するときに、ヌル (X'00') は省略するが、ブランク (X'40') は組み込むためです。このことにより、出力データ・ストリームがさらに短くなります。BMS は、フィールドの中の先頭文字がヌルであれば、2 桁目以降の文字とは無関係にそのフィールドを省略します。

BMS では、ユーザーがマップを作成するために使用するすべての区域を、ヌルで初期設定する必要があります。これは、シンボリック・マップ構造体の mapnameO フィールドに、ヌル (X'00') を移動させることによって行われます。詳しくは、803 ページの『出力マップの初期化』を参照してください。BMS は、属性位置およびデータの先頭位置でヌルを使用して、マップ内の値が変更されていないことを指示します。プログラムまたは TIOA でマップ域を再利用している場合には、この方法でマップをクリアする際には特に注意する必要があります。

CICS 領域の適切なアドレッシング

CICS 域が正しくアドレッシングされていることを検査する方法はいくつかあります。次の事項を確認してください。

- 4KB を超える LINKAGE SECTION 構造体を持つ各 COBOL プログラムは、必須の定義および 1 つ以上の隣接する BLL セルの指定を持っています。
- すべての BLL ポインターは、01 レベル項目である区域をポイントします。
- 呼び出しレベル DL/I は、正しくアドレスされた PSB とともにしか、使用されません。

可能な場合の MAPONLY オプションの使用

MAPONLY オプションは、マップ内の***定数*** データのみを送信し、プログラムのどの変数データとも組み合わせません。結果のデータ・ストリームは常に短くなるわけではありませんが、BMS での操作のパス長は短くなります。データ入力に使用するスケルトン画面を送信する場合には、しばしば MAPONLY を使用することになります。

既存の画面への変更フィールドのみの送信

変更済みフィールドのみを送信することが重要なのは、例えば、メッセージを追加するか、あるいは入力画面上の 1 つ以上のフィールドがエラーであることを示すために強調表示する場合などです。これらの状態では、DATAONLY オプションを使用して、変更済みフィールドを除いたヌルから成るマップを送信する必要があります。属性バイトのみを変更したフィールドの場合には、そのバイトだけを送信すればよく、残りのフィールドはヌルとして送信します。BMS は、この入力を使用して、問題のあるフィールドのみで構成されるデータ・ストリームを作成し、画面上のその他のすべてのフィールドは未変更のままです。

ここでの助言を無視して、不必要に長いデータ・ストリームを送信しがちです。例えば、入力画面のエラーを検査しているプログラムがエラーを検出した場合には、選択肢が 2 つあります。

- プログラムは、入力マップにエラー情報 (強調表示属性、エラー・メッセージなど) を追加し、再送することができます。
- プログラムは、エラー・フィールドおよびメッセージ・フィールドだけで構成された、まったく新規の画面を作成することができます。

前者はわずかにコーディングが容易になります (2 つのマップ域を持つことも、フィールドを転送する必要もありません) が、出力データ・ストリームには、エラー・フィールドおよびメッセージ・フィールドの他に正しいフィールドも含まれているので、伝送ははるかに長くなる可能性があります。実際、入力に空のフィールドまたは短いフィールドがあった場合には、BMS は脱落文字をブランクまたはゼロによって置き換えるので、元の入力ストリームより長くなることすらあります。

3270 ハードウェアに関して、端末の入力ストリームが 256 バイトを超える場合には、端末制御装置は、入力ストリームを最大 256 バイトの個別の伝送に自動的に分割します。これは、長い入力ストリームが、複数の物理 I/O 操作を必要とする場合があることを意味しています。この点はアプリケーション・プログラムにとって透過的ですが、回線およびプロセッサーのオーバーヘッドが増える原因になります。一般に、***出力*** ストリームは単一の伝送で送信されます。

回線トラフィックを削減するためのデータ入力操作の設計

しばしば、ユーザーは同一画面を複数回にわたり完成する必要があります。各サイクルではデータだけが変更され、表題、フィールド・ラベル、指示などは未変更のままです。この状態では、項目を受け入れ、処理する場合に、SEND CONTROL ERASEAUP コマンド (または短い確認メッセージのみを含み、ERASEAUP オプションを指定したマップ) で応答することができます。これにより、画面上のすべての**無保護**フィールド (つまり、最後の入力でのすべての入力データ) が消去され、その MDT がリセットされます。保護フィールドに入っているラベルおよび他のテキストは未変更で、画面は次のデータ入力サイクルのために使用可能になり、さらに必要なデータしか送信されません。

画面に送信されるデータの圧縮

不定様式データを画面に送信するか、あるいは定様式画面を BMS の外側に作成する場合には、バッファー・アドレス設定 (SBA) 命令およびアドレス反復 (RA) 命令をデータ・ストリームに挿入することによって、データをさらに圧縮することがで

きます。SBA によって、画面上のデータを位置指定し、RA によって、それに続く文字がバッファー内の現在地点から指定の終点アドレスまで生成されます。SBA は、画面上に未使用区域がかなりあって、その後にデータが続いている場合はいつも有用です。RA が有用なのは、画面上にブランクまたはダッシュなど、同一文字の長いシーケンスがある場合です。しかし、RA が処理する速度は 3270 制御装置のすべてのモデルを通じて一様ではないことに、注意する必要があります。RA を使用する前に、RA がユーザーの構成にどのように適用されるかを検査する必要があります。

CICS は、出力が端末に送信される (XTC OUT) 直前に駆動される出口ルーチンを提供します。SBA 置換および RA 置換をこの出口ルーチンに追加して、一般サブルーチンを使用してデータ・ストリームを圧縮したくなる場合があります。これには、ユーザー・アプリケーション・プログラムから圧縮論理を除去することと、出力データ・ストリームが BMS によって作成されたか、あるいは BMS 以外によって作成されたかにかかわらず、すべてに適用されることの二重の利点があります。

ブランクの代わりのヌルの使用

BMS の外側では、***出力*** データ・ストリーム中のヌルは、特別の意味を持たないことに注意する必要があります。画面上にブランクの区域が必要な場合には、ブランクまたはヌルのいずれかを画面に送信することができます。これらは出力ストリームで同じスペースを占めます。しかし、ユーザーおよび後続の読み取りがブランク・フィールドを変更しそうな場合には、ヌルは返送されないので、ヌルを使用します。

ヌルまたはブランクの必要を回避する方式の使用

ブランクにする必要がある画面の**大きな** 区域の場合には、ブランクまたはヌルの伝送以外の方式を考慮する必要があります。例えば、BMS を使用するか、SBA 命令および RA 命令を直接データ・ストリームに入れるか、あるいは ERASE オプションと ERASEAUP オプションを使用します。

ページの作成およびルーティングの操作

BMS ページ作成操作機能は、長いメッセージの作成と表示、複数の宛先へのメッセージの送信、および異なる物理的特性を持つ複数の装置についての単一メッセージの形式制御のために、強力で柔軟性のあるツールを提供します。しかし、高機能ツールの場合と同様、 586 ページの『効果的なブラウズ (非 RLS モード)』で言及したようにかなりのオーバーヘッドを必要とします。次の場合には、ページ作成オプション (ACCUM) が必要となります。

- 出力装置の容量を超える長さのメッセージの送出 (複数ページ出力)
- 入力端末以外の宛先の使用
- 複数のマップから作成されたページの送出
- BMS ページ・コピー機能の使用

複数ページ出力の送信

画面サイズの多くのページ数からなる、非常に大きな出力メッセージを生成するトランザクションは、システム・リソースの負担になりがちです。最初に、すべての

ページを作成する必要があり、これにはプロセッサー・アクティビティー、CSPG トランザクションの実行、およびデータ・セットの I/O アクティビティーがともないます。そのあと、このページは一時記憶域に保管しなければなりません。端末ユーザーがメッセージをページごとに見る場合には、ページ送り要求を処理するために多くのトランザクションが実行され、そのそれぞれに回線オーバーヘッドとプロセッサー・オーバーヘッドが必要になります。明らかにオーバーヘッドの一部は、トランザクションのサイズと複雑さがもたらすもので、不可避な場合があります。確かに、複数のユーザーがページ出力を同時に急速にスクロールしている場合には、必要なトランザクションがシステムを独占することがあります。

ユーザーが本当にすべてのページを見る必要があり、前後に頻繁にスクロールする必要がある場合には、すべてのページを同時に生成し、「従来の」CICS ページ送りサービスを使用してそれを提供すると、もっと効率的になる可能性があります。しかし、ユーザーが数ページしか必要としないか、あるいはメッセージを前後にどのくらいスクロールしたいかを容易に指定することができる場合には、選択肢が次のように 2 つあります。

1. 最初、疑似会話型トランザクションを構成して、出力を 1 画面だけ作成します。最初にこのトランザクションを実行するときに、複数ページ出力の最初のページを生成します。出力画面にはユーザーが次に必要なページを指示するためのスペースが含まれています。トランザクションは、次に実行するときに要求されたページを表示できるように、常に次のトランザクション ID を設定してそれ自体を指すようにします。

   ユーザーに CICS 提供のオプションの一部 (1 ページ下方、1 ページ上方、および選択したページへスキップなど)、および次の出力ページを始めるデータ・セット・キーなどの、アプリケーションに関係のある一部のオプションを与えたくなる場合があります。
2. もう 1 つは、ACCUM オプションを使用して複数ページ出力メッセージのページを作成するものの、メッセージのページ数 (例えば、5 ページなど) を制限することです。ユーザーは通常の CICS ページ・コマンドを使用してサブセット・ページを指定します。出力の最終画面で、もっと出力があることを指示し、次のセグメントを見たいかどうかを指示します。最初の例と同様に、CICS がページ送りコマンドを受け取らない場合に、トランザクションを呼び出すように、次のトランザクション ID を元のトランザクション ID に設定します。

入力端末以外の宛先へのメッセージの送信

タスクと関連した入力端末以外の端末にメッセージを送信したい場合には、BMS ルーティングがそれを行うのに最も効率的な方法である可能性があります。これが特に効率的なのは、メッセージを複数の宛先に送信する必要がある場合、あるいはメッセージに複数のページがある場合です。推奨する方式がルーティングとなるのは、メッセージの受信側がメッセージをアクセスするために CICS ページ送りコマンドを必要とする場合です。

ただし、上記のどちらの条件にも該当しない場合は、トランザクションに関連しない端末に出力を配布する方法として、この他にも次の 2 とおりの方法があります。

1. TERMID オプションを指定して START コマンドを使用して、書き込み先の端末を指定し、FROM オプションを指定して送信したいデータを指定することができます。ユーザー独自のトランザクションは開始済みトランザクションです。

このトランザクションがメッセージに関する RETRIEVE コマンドを発行してから、それを自分の端末に送信します。START コマンドに関するプログラミング情報については、 「*CICS アプリケーション・プログラミング・リファレンス*」を参照してください。

2. 同様に、特定の端末に予定されているメッセージを、区画内一時データ・キューに入れることができます。一時データ・キューに対する定義には、次の事項を指定しなければなりません。
   - TERMINAL としての宛先
   - 端末 ID
   - トリガー・レベル
   - トランザクション名

   ユーザー独自のトランザクションは一時データ・キューを読み取り、メッセージをその端末に送ります。このトランザクションは、この処理をキューが空になるまで繰り返してから終了します。指定したトリガー・レベルは、指定された数のメッセージがキューに置かれるたびに、呼び出されることを意味します。
   「*CICS/ESA Sample Applications Guide*」には、この機能を実行する DFHœTDWT サンプル・プログラムについての説明があります。

注: メッセージのルーティング (どのような手段によるものでも) には、オーバーヘッドが生ずるので、ROUTE=ALL などの機能を使用する場合は注意してください。

複数のマップから作成されたページの送出

異なるマップを段階的に使用して、1 つの画面を作成することは容易にできますが、特に出力の画面が 1 つしかなく、ページ送りの必要がない場合に、ページ作成操作を使用しないことによって、オーバーヘッドを避けられることがあります。この例は、出力が、ヘッダー・マップ、それに続く 2 番目のマップとともに送信される数が可変の明細セグメント、そして最後に明細に続くトレーラー・マップからなるアプリケーションです。このようなアプリケーションの平均出力画面には、8 つの明細セグメント (2 行) にヘッダーとトレーラーが加わったものが含まれ、このすべてが単一画面に収まるものとします。プログラムが出力画面を内部的に作成する場合は、BMS 呼び出しは 1 回しか必要ないのに対して、ページ作成によるこの画面の書き込みには、11 回の BMS 呼び出し (ヘッダー、明細、トレーラー、およびページアウト) が必要です。

BMS ページ・コピー機能の使用

累積される BMS メッセージを構成する個別のページは一時記憶域に保管されるので、BMS によって端末ユーザーは個別ページを他の端末にコピーすることができます。しかし、コピーする機能がページ作成を使用する唯一の理由である場合には、代わりに、3274 制御装置コピー機能または CICS **コピー・キー**機能のいずれかを使用することを、考慮する必要があります。

3274 コピー機能は、CICS との関連および伝送を必要とせず、格段に効率的な方式です。BMS コピー機能とは別のタイプですが、CICS コピー・キー機能はオーバーヘッドをともないます (742 ページの『印刷出力の要求』を参照してください)。また、これには、BMS のコピーには適用されない宛先の制約事項があります。

第 16 部 付録

© Copyright IBM Corp. 1989, 2009

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あり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、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。

本書には、技術的に正確でない記述や誤植がある場合があります。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。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN 本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

© Copyright IBM Corp. 1989, 2009

商標

IBM、IBM ロゴ、および ibm.com は、International Business Machines Corporation 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。これらおよび他の IBM 商標に、この情報の最初に現れる個所で商標表示 (® または TM) が付されている場合、これらの表示は、この情報が公開された時点で、米国において、IBM が所有する登録商標またはコモン・ロー上の商標であることを示しています。このような商標は、その他の国においても登録商標またはコモン・ロー上の商標である可能性があります。現時点での IBM の商標リストについては、 www.ibm.com/legal/copytrade.shtml の「Copyright and trademark information」をご覧ください。

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。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc.の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

参考文献

CICS Transaction Server for z/OS の CICS ブック

一般

CICS Transaction Server for z/OS Program Directory、GI13-0536
CICS Transaction Server for z/OS リリース・ガイド、GC88-5845
CICS Transaction Server for z/OS V2.3 からのアップグレード、GC88-5847
CICS Transaction Server for z/OS V3.1 からのアップグレード、GC88-5848
CICS Transaction Server for z/OS V3.2 からのアップグレード、GC88-5849
CICS Transaction Server for z/OS インストール・ガイド、GC88-5846

CICS へのアクセス

CICS インターネット・ガイド、SC88-5853
CICS Web サービス・ガイド、SC88-5852

管理

CICS System Definition Guide、SC34-6999
CICS Customization Guide、SC34-7001
CICS Resource Definition Guide、SC34-7000
CICS Operations and Utilities Guide、SC34-7002
CICS RACF Security Guide、SC34-7003
CICS Supplied Transactions、SC34-7004

プログラミング

CICS アプリケーション・プログラミング・ガイド、SC88-5850
CICS アプリケーション・プログラミング・リファレンス、SC88-5851
CICS System Programming Reference、SC34-7024
CICS Front End Programming Interface User's Guide、SC34-7027
CICS C++ OO Class Libraries、SC34-7026
CICS Distributed Transaction Programming Guide、SC34-7028
CICS Business Transaction Services、SC34-7029
Java Applications in CICS、SC34-7025

診断

CICS Problem Determination Guide、SC34-7034
CICS パフォーマンス・ガイド、SC88-5854
CICS Messages and Codes、SC34-7035
CICS Diagnosis Reference、GC34-7038
CICS Recovery and Restart Guide、SC34-7012
CICS Data Areas、GC34-7014
CICS Trace Entries、SC34-7013
CICS Supplementary Data Areas、GC34-7015
CICS Debugging Tools Interfaces Reference、GC34-7039

© Copyright IBM Corp. 1989, 2009

通信

CICS 相互通信ガイド、SC88-5857
CICS External Interfaces Guide、SC34-7019

データベース

CICS DB2 Guide、SC34-7011
CICS IMS Database Control Guide、SC34-7016
CICS Shared Data Tables Guide、SC34-7017

CICS Transaction Server for z/OS の CICSPlex SM ブック

一般

CICSPlex SM 概念および計画、SC88-5855
CICSPlex SM WUI ガイド、SC88-5856

管理

CICSPlex SM Administration、SC34-7005
CICSPlex SM Operations Views Reference、SC34-7006
CICSPlex SM Monitor Views Reference、SC34-7007
CICSPlex SM Managing Workloads、SC34-7008
CICSPlex SM Managing Resource Usage、SC34-7009
CICSPlex SM Managing Business Applications、SC34-7010

プログラミング

CICSPlex SM Application Programming Guide、SC34-7030
CICSPlex SM Application Programming Reference、SC34-7031

診断

CICSPlex SM Resource Tables Reference、SC34-7032
CICSPlex SM Messages and Codes、GC34-7035
CICSPlex SM Problem Determination、GC34-7037

他の CICS 資料

以下の資料には CICS に関する詳しい情報が含まれますが、これらの資料は CICS Transaction Server for z/OS バージョン 4 リリース 1 の一部としては提供されません。

Designing and Programming CICS Applications、SR23-9692
CICS Application Migration Aid Guide、SC33-0768
CICS ファミリー: API の構成、SC88-7261
CICS ファミリー クライアント・サーバー プログラミングの手引き、SC88-7429
CICS Family: Interproduct Communication、SC34-6853
CICS Family: Communicating from CICS on System/390、 SC34-6854
CICS Transaction Gateway (OS/390 版) 管理の手引き、SD88-7246
CICS Family: General Information、GC33-0155
CICS 4.1 Sample Applications Guide、SC33-1173
CICS/ESA 3.3 XRF Guide、SC33-0661

その他の IBM 資料

以下の資料には、関連した IBM 製品についての情報が含まれています。

MVS

「*z/OS MVS 初期設定およびチューニング・ガイド (SA88-8563)*」
「*z/OS MVS 初期設定およびチューニング解説書 (SA88-8564)*」
「*z/OS MVS JCL 解説書 (SA88-8569)*」
「*OS/390 V2R8.0 MVS システム・コマンド (GC88-6592)*」

SNA

「*Sessions between Logical Units (GC20-1868)*」

SQL

「*DB2 ユニバーサル・データベース (OS/390 版および z/OS 版) アプリケーション・プログラミングおよび SQL ガイド (SC88-8763)*」
「*DB2 Universal Database (OS/390 および z/OS 版) SQL 解説書 (SC88-8772)*」

その他の関連資料

「*An Introduction to the IBM 3270 Information Display System (GA27-2739)*」
「*3274 Control Unit Reference Summary (GX20-1878)*」
「*Component Description: IBM 2721 Portable Audio Terminal (GA27-3029)*」
「*IBM 2780 Data Transmission Terminal Component Description (GA27-3035)*」
「*CICS/ESE 3.3 IBM 3270 Data Stream Device Guide (SC33-0232)*」
「*IBM 3270 Data Stream Programmer's Reference (GA23-0059)*」
「*IBM 4700/3600/3630 Guide (SC33-0233)*」

アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア・プロダクトを快適に使用できるようにサポートします。

CICS システムのセットアップ、実行、および保守に必要なほとんどの作業は、以下のいずれかの方法で行うことができます。

- CICS にログオンした 3270 エミュレーターを使用する
- TSO にログオンした 3270 エミュレーターを使用する
- 3270 エミュレーターを MVS システム・コンソールとして使用する

IBM パーソナル・コミュニケーションズは、身体障害のある方々のためのアクセシビリティー機能を持つ 3270 エミュレーションを提供します。 CICS システムで必要なアクセシビリティー機能を提供するためにこの製品を使用することができます。

© Copyright IBM Corp. 1989, 2009

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]

アクセス、システム情報への
  EXEC インターフェース・ブロック (EIB) 6
アクティブ区分画面 872
アセンブラー言語 71, 109
  アプリケーション 274
  言語環境プログラムの要件 73
  作業用ストレージ 71
  準拠していないルーチン 73
  制約事項 71
  他の言語との混合 16, 76
  プログラミング手法 71, 73, 76
  31 ビット・アドレッシング 71
  64 ビット・レジスター 71
  CALL ステートメント 76
  DFHECALL マクロ 109
アセンブル 81
アセンブル、TYPE=DSECT 788
宛先の識別 618
アテンション ID 824
アテンション・フィールド 882
アドレッシング、CICS 域の 886
アドレッシング・モード (AMODE)
  オプション、CICS アプリケーションの 131
アプリケーション
  XML への変換 429
アプリケーション・データへの XML スキーマのマッピング 502
アプリケーション・パフォーマンスのモニター 403
アプリケーション・プログラム
  インストール 119
  機能シップ 409
  作成 4
  使用、BMS マップ・セットの 137
  設計 241
  相互通信に関する考慮事項 407
  テスト 172
  トランザクション・ルーティング 408
  パフォーマンスの設計 271
  非同期処理 423
  分散トランザクション処理 423
アプリケーション・プログラム (続き)
  分散プログラム・リンク 409
  論理レベル 326
アプリケーション・プログラムの論理レベル 34
アンエスケープ 684, 704
域、動的ストレージ 272
異常終了 388
異常終了、ユーザー・タスク、EDF 195
異常終了出口機能 397
異常終了出口プログラム 398
異常終了出口ルーチン 399
異常終了のリカバリー 397
一時記憶域
  キュー 677
  データ 677
  ブラウズ・トランザクション、CEBR 203
  補助 263, 678
  メイン 263, 678
一時記憶キュー
  文書テンプレートとして 692
一時ストレージ、データを渡すために使用 323
一時データ 677, 753
  キュー 265, 360
  区画外 266, 279
  区画内 265
一時データ管理
  キュー 673, 674
  自動トランザクション開始 (ATI) 675
一時データ・キュー
  文書テンプレートとして 692
イベント
  モニター点 403
入り口点、トレース 402
印刷 741
  一時データ 753
  行の長さ 745
  BMS ルーティングによる 755
  CICS API に関する考慮事項 756
  START コマンド 753
印刷キー 596, 758
印刷キー、ハードウェア 760
印刷形式設定 612
印刷出力の要求 742
印刷制御ビット 744
インストール、アセンブラー・アプリケーション・プログラムの
  例、ジョブ・ストリームの 141
インストール、アプリケーション・プログラムの
  アセンブラー言語 141
  使用、ユーザー独自のジョブ・ストリームの 153
  C 149
  COBOL 143
  PL/I 147
インストール、HTML テンプレート 166
インターバル制御 643
  インターバル制御コマンドの取り消し 643
  タスクの開始 643
  遅延、タスク処理の 643
  満了時刻 644
  要求 ID の指定 645
  DELAY コマンド 644
  POST コマンド 644
  START コマンド 644
インターフェース・スタブ、EXEC 107
インターフェース・モジュール
  使用 140
  プログラム、EXEC CICS または EXEC DLI コマンドを使用する 154
  CPI コミュニケーション 106
  EXEC 106
  SAA リソース・リカバリー 106
インターリーブ、メッセージ・ルーティングによる会話の 862
インバウンド
  データ・ストリーム 885
エスケープ・シーケンス 684, 704
エンキュー
  VSAM 内部 277
  VSAM ファイル内の 277
オーバーレイ 276
オープン・トランザクション環境 (OTE) 251
大きな COMMAREA 289, 292, 311, 319
大文字変換、BMS での 823
オプション
  ファンクション・キーの、EDF 195
  HANDLE CONDITION コマンド 389
オプション・リスト、変換プログラムの動的起動における 87
オペレーティング・システム待機 278

© Copyright IBM Corp. 1989, 2009

[カ行]

カーソル位置 813
カーソル位置決め、記号 813
カーソル検出可能フィールド 881
カーソルの検出 825
概説
  チャネルを使用した動的ルーティング 312, 323
回線通信量の削減 887
外部 CICS インターフェース (EXCI) 424
外部形式制御 883
外部コントローラー 617
外部呼び出しインターフェース (EXCI) 407
改ページ 840
会話型プログラミング 241, 621
会話パートナー 591
カウンター名
  名前付きカウンター 718
書き込み、レコードの 576, 581
拡張相対バイト・アドレス (XRBA) 546
拡張相対バイト・アドレッシング (XRBA)
  アップグレード 547
拡張読み取り専用 DSA (ERDSA) 134
確定応答プロトコル
  端末管理 608
重ね書き、EDF 画面の 193
仮想記憶域 272
仮想記憶域環境 271
仮想ルックアサイド機能 (VLF) 133
カップリング・ファシリティー・データ・テーブル 558
カップリング・ファシリティー・リスト構造
  現行値 718
可変配列 479
画面からの読み取り、形式設定 822
画面コピー、BMS 760
画面フィールド、3270 823
空のデータ・セット 543
間接キュー 675
キー
  総称 568
  代替 (2 次) 543
  ハードウェア印刷 760
  物理 555
キー順データ・セット (KSDS) 541
記号 684, 698
  値の定義 703, 704
  出力マップ 819
  入力マップ 819
  文書テンプレート内での位置指定 698
記号カーソル位置決め 813
記号参照 684, 698
記号置換 684
記号テーブル 684
記号リスト 684, 703, 704
疑似会話型プログラミング 241
機能 (PF) キー、CEBR トランザクション 206
機能、EDF 175
機能管理ヘッダー
  説明 609
機能シップ 407, 409
機能伝送 578
基本マッピング・サポート
  アセンブル、TYPE=DSECT 788
  アセンブルおよびリンク・エディット、物理マップ・セットの 162
  アプリケーション・プログラムにおける BMS マップ・セットの使用 137
  インストール、区分セットの 169
  インストール、シンボリック記述マップ・セットの 163
  インストール、物理マップ・セットの 161
  インストール、マップ・セットの 158, 159
  大文字変換 823
  カーソル位置 813
  カーソルの検出 825
  画面コピー 760
  完全 777
  グループ・フィールド 791
  コピー機能 890
  最小 776
  サポートされる端末 776
  修正データ・タグ (MDT) 885
  出力マップの初期化 803
  出力例 777
  準備、マップの 157
  シンボリック記述マップ・セット、BMS 用 163
  シンボリック・マップ 159, 788
  タイプ、マップ・セットの 159
  データのマップへの移動 803
  データ・ストリーム 885
  ディスプレイからのデータの受信 817
  入力データのマップ 821
  パス長の最小化 844
  パフォーマンスの考慮事項 844
  反復フィールド 792
  標準 776
  フィールド 779
  複合フィールド 791
  複数マップ画面 890
  物理マップ 159, 788
  プラットフォーム間のサポート 777
  プログラムにおける使用、シンボリック・マップ・セットの 163
基本マッピング・サポート *(続き)*
  ページ作成操作 888
  ページ・ルーティング操作 888
  マクロ 781
  マクロを書くための規則 785
  マップ 777, 885, 886, 890
  マップのアセンブル 788
  マップの作成 781
  マップのためのストレージ 801
  マップ・セット 789
  無効データ 814
  メッセージ長の削減 845
  リンク・エディット 788
  BMS サポート・レベル 775
  DFHASMVS プロシージャー 162
  DFHLNKVS プロシージャー 162
  DFHMAPS、マップのインストール用プロシージャー 167
  DFHMDF マクロ 781
  DFHMDI マクロ 781, 784
  DFHMSD マクロ 781, 784
  DFHPSD、区分セット定義の 169
  EOC 状態 830
  GRPNAME オプション 791
  MAPSET リソース定義 788
  MDT 823
  OCCURS オプション 792
  PROGRAM リソース定義 788
  SEND MAP コマンド 801
  TYPE=DSECT アセンブル 788
逆割り込み 593
キュー
  一時記憶域 677
  一時データ 673
  区画外 674
  区画内 673
競合、端末の 592
競合、リソースに対する 242
共通作業域 (CWA) 281
  保護 282
行の長さ、印刷時の 745
共用、トランザクション間のデータの 281
共用ストレージ 266
共用制御、レコードの
  VSAM RLS 550
共用データ・テーブル 557
局所性、参照の 273
空間、表示 867
区画外一時データ 266, 279
区画外キュー 674
区画内一時データ 265
区画内キュー 673
区分画面 867
区分画面、アクティブ 872
区分画面の定義 869

区分セット
インストール 168
ロード、16MB 境界より上のアドレスに 158
区分データ・セット (PDS)
文書テンプレートとして 691
クライアント領域 410
クライアント・コード・ページ 689
グループ・フィールド 791
グローバル・ユーザー出口 658
形式設定画面からの読み取り 822
経路リスト 854
セグメント・チェーン項目形式 857
標準入力形式 856
LIST オプション 856, 857
現行チャネル
概説 296, 300
例、LINK コマンドを使用した 296
例、XCTL コマンドを使用した 299
言語環境プログラム 11
アセンブラー言語 73
異常終了処理 14
PL/I 67
言語環境プログラム の条件処理ルーチン 14
言語の混合 16
サポートされるコンパイラー 11
サポート・レベル 11
条件処理 14
条件ハンドラー、ユーザー作成 14
ストレージ 15
ダイナミック・リンク・ライブラリー 18
ダンプ宛先 12
メッセージ宛先 12
呼び出し可能サービス 12
ランタイム・オプション 19, 22
および CICS LINK 19
子エンクレーブにおける 19
CBLPSHPOP 29
CEEBXITA ユーザー出口 21
CEECSTX ユーザー出口 21
AID 処理 14
CEEBINT 23
CICSVAR 環境変数 22
DLL 18
HANDLE AID コマンド 14
HANDLE CONDITION コマンド 14
HLL ユーザー出口 23
PL/I 67
VS COBOL II 31
言語環境プログラムの ランタイム・オプション 19
STORAGE 26
言語環境プログラムの CBLPSHPOP ランタイム・オプション 29
言語構造
WSDL への変換 431
言語の混合 16
検査、プログラムの 175
検出可能フィールド 881
コールバック・コンテナー
DFHRL-CONTROL 533
DFHRL-DATA 534
DFHRL-NAME 535
DFHRL-ROOT 535
DFHRL-TYPE 531, 535
コールバック・プログラム 531
高輝度 881
更新、レコードの 576
更新セット 543
更新操作、BDAM 579
肯定 593
子エンクレーブ 19
コピー機能
BMS 890
コピーブックの変換 105
コマンド、SYNCPOINT 380
コマンド言語変換プログラム 84, 88
オプション 88, 90
行番号 86
APOST オプション 92
CBLCARD オプション 92
CICS オプション 92
COBOL2 オプション 92
COBOL3 オプション 93
CPSM オプション 93
DBCS オプション 93
DEBUG オプション 93
DLI オプション 94
EDF オプション 94
EPILOG オプション 94
EXCI オプション 94
FEPI オプション 95
FLAG オプション 95
GDS オプション 95
GRAPHIC オプション 95
LENASM オプション 96
LENGTH オプション 96
LINECOUNT オプション 96
LINKAGE オプション 96
MARGINS オプション 97
NATLANG オプション 98
NOCBLCARD オプション 98
NOCPSM オプション 98
NODEBUG オプション 98
NOEDF オプション 98
NOEPILOG オプション 98
NOFEPI オプション 99
NOLENGTH オプション 99
NOLINKAGE オプション 99
NONUM オプション 99
コマンド言語変換プログラム *(続き)*
NOOPSEQUENCE オプション 99
NOOPTIONS オプション 99
NOPROLOG オプション 100
NOSEQ オプション 100
NOSEQUENCE オプション 100
NOSOURCE オプション 86, 100
NOSPIE オプション 100
NOVBREF オプション 100
NOXREF オプション 100
NUM オプション 101
OPMARGINS オプション 101
OPSEQUENCE オプション 101
OPTIONS オプション 101
PROLOG オプション 101
QUOTE オプション 102
SEQ オプション 102
SEQUENCE オプション 102
SOURCE オプション 86, 102
SP オプション 103
SPACE オプション 103
SPIE オプション 103
SYSEIB オプション 103
VBREF オプション 86, 103
XOPTS キーワード 89
XREF オプション 103
コマンド・レベル・インタープリター
セキュリティーの考慮事項 211
メッセージ表示画面 221
呼び出し 217
EIB 221
ENTER キー 216
混合アドレッシング・モード・トランザクション 332
コンテキスト
コンテナー、BTS またはチャネルの 310
コンテナー
概説 289
基本的な例 290
コンテキスト、BTS またはチャネル 310
チャネルの設計 306
プログラムに渡されたコンテナーの検出 305
読み取り専用 306
リンクから返されたものの検出 305
JCICS からの使用 311
コンパイル 81
コンポーネント
複数、対話式 294
複数のチャネルに 1 つのコンポーネント 294
1 つのチャネル - 複数のプログラム 293
コンポーネント・アーキテクチャー 515

コンポジット 517
コンポジットのワイヤリング 517

[サ行]

サーバー
  プログラム 412
  領域 410, 413
サーバー・サイド・インクルード・コマンド 699
サービス 515
サービスの処理 524
サイズ、プログラムの 271
再入可能性 250
作業セット 273
先読みキューイング 593
索引、代替 543
削減、回線通信量の 887
削除、レコードの 579
サブスペース 668
サブルーチン 274
サポートされるコンパイラー
  言語環境プログラム 11
参照修正 41
参照セット 275
シーケンスの制御、リソースへのアクセス 648
支援機能、 XML 430
時間フィールド、EIB の 6
磁気スロット読み取り装置、10/63 879
識別
  BDAM レコード 555
  VSAM レコード 545
システム情報へのアクセス 5
システム・トレース入り口点 402
事前印刷用紙 679
事前変換済みコード 153
実行診断機能 86, 93, 175
実行単位 34
指定機能文字 881
自動タスク開始 244, 592
自動トランザクション開始 675
シナリオ
  複数の対話式コンポーネント 294
  複数のチャネルに 1 つのコンポーネント 294
  1 つのチャネル - 1 つのプログラム 293
  1 つのチャネル - 複数のプログラム 293
ジャーナル
  レコード 267, 377
ジャーナル ID 380
ジャーナル管理
  出力の同期化 378
ジャーナル処理 280, 377
ジャーナル・タイプ ID 380
修正、実行の、EDF 193
主記憶装置 272
  一時データ 678
出力データのチェーニング 607
出力マップ、記号 819
出力マップの初期化 803
手法、プログラミングの 271
順次端末サポート 171, 611
照会トランザクション 247
状況フラグ・バイト、経路リスト 857
条件、例外 383
常駐モード (RMODE)
  オプション、CICS アプリケーションの 131
商標 894
ジョブ入力サブシステム・コンポーネント、MVS の 761
シンボリック記述マップ・セット
  使用、プログラムにおける 137
垂直タブ 748
水平タブ 748
据え置きジャーナル出力 379
スキーマ
  バンドル・マニフェスト 535
スキーマ、SCA コンポジット 537
スタブ、EXEC インターフェース 107
ストレージ
  一時 263
  共用可能 654
  静的 275
  プログラム 263
  メイン 272
  ユーザー 261
  ユーザー・キー 655
  CICS キー 655
ストレージ域、動的 272
ストレージ制御 653
ストレージ保護 653
スプール
  コマンド 6
  ファイル 761
スレッド・セーフ・プログラム 251
  効果
    ADDRESS CWA 255
    EXTRACT EXIT 255
    GETMAIN SHARED 255
制御
  BDAM の排他 579
  VSAM ブロックの 549
制御の渡し、戻りを前提とした (LINK) 326
静的 LIBRARY (DFHRPL) 120
静的ストレージ 275
制約事項
  アセンブラー言語 71
制約事項 *(続き)*
  31 ビット・アドレッシング 71
  64 ビット・レジスター 71
  C および C++ 54
  COBOL 26, 36, 41
  PL/I 66
セキュリティー
  プログラミングのヒント 406
  レコード・レベルまたはフィールド・レベル 405
  CICS 定義のリソース ID 406
  EDF 179
  SEC システム初期設定オプション 406
  SPCOMMAND リソース・タイプ 406
設計上の考慮事項、アプリケーションの
  リソースの排他制御 276
選択フィールド 881
相互通信 407
総称キー 568
総称削除 579
相対バイト・アドレス (RBA) 542, 546
相対バイト・アドレッシング (RBA)
  アップグレード 547
相対レコード番号 (RRN) 542, 546
相対レコード・データ・セット (RRDS) 542
送達不能なメッセージ 859
装置依存サポート 798
装置依存マップ 797

[タ行]

待機、オペレーティング・システム 278
代替
  キー 543
  索引 543
ダイナミック・リンク・ライブラリー 18
大容量の COMMAREA 289, 292, 311, 319
対話式デバッグ
  CECI トランザクション 211
  CECS トランザクション 217
  CEDF トランザクション 175
対話式問題制御システム 403
タスク関連ユーザー出口 658
タスク制御 647
  リソースへのアクセス・シーケンス 648
妥当性、参照の 274
タブ、垂直 748
タブ、水平 748
単位、作業の 380
単一画面モード、EDF 188
単一スレッド化 249
単一スレッド・テスト 171

端末
オプション 807
競合 592
共用 224
サポート、順次 611
待ち 594
端末、拡張データ・ストリーム 161
端末管理
印刷形式設定 612
会話パートナー 591
確定応答 608
機能管理ヘッダー (FMH) 609
コマンド 591
先読みキューイング 593
出力データのチェーニング 607
中断プロトコル 593
テーブル・ユーザー域 (TCTUA) 285
入力データのチェーニング 607
パートナー、会話 591
半二重モード 591
ブラケット・プロトコル、LAST オプション 610
フリップフロップ・モード 591
プロトコル、中断 593
マップ入力データ 821
論理装置の機能 607
論理レコードの提示 608
割り込み 593
FMH、アウトバウンド 609
FMH、インバウンド 609
端末制御コマンド 587
端末を使用しないトランザクション
EDF 191
チェーニング 595
チェーニング、データの 607
チャネル
概説 289
基本的な例 290
現行、例、LINK コマンドを使用 296
現行、例、XCTL コマンドを使用 299
現行の 296, 300
構成 308
作成 295
設計 306
大容量の COMMAREA として 289
データ変換 315
典型的なシナリオ
複数の対話式コンポーネント 294
複数のチャネルに 1 つのコンポーネント 294
1 つのチャネル - 1 つのプログラム 293
1 つのチャネル - 複数のプログラム 293
チャネル *(続き)*
動的および分散ルーティング 312, 323
プログラムに渡されたコンテナーの検出 305
有効範囲 301, 303
読み取り専用コンテナー 306
利点 319
リンクから返されたコンテナーの検出 305
BTS アクティビティーとの比較 309
JCICS からの使用 311
LINK および XCTL コマンドの 263
LINK コマンド 333
RETURN コマンド 335
RETURN コマンドにおける 286
チャネル、大きな COMMAREA としての 289, 292, 311, 319
チャネルの構成 308
チャネルの作成 295
チャネルの設計 306
チャネルの有効範囲
概説 301
例、LINK コマンドを使用した 301
例、XCTL コマンドを使用した 303
チャネルを使用した動的ルーティング 312, 323
チャネルを使用する START データのマイグレーション 323
チャネル・ベース・サービス 520, 524
中断データ・セット 678
直接端末 862
追加、レコードの 581
データ
初期設定 274
他のプログラムへの受け渡し 327
チェーニング 607
定義 274
トランザクション内の格納 260
レコード 267
データ、ディスプレイからの読み取り 821
データ交換ブロック 86
データ終了標識文字 612
データの受け渡し、他のプログラムへの 327
データの格納、トランザクション内の 260
データのマップへの移動 803
データ変換 313
およびチャネル 313
SOAP の例 318
チャネルを使用した 315
必要な理由 313
データを渡すために一時ストレージを使用するプログラムのマイグレーション 323
データ・ストリーム
圧縮 887
アドレス反復命令 (SBA) 887
インバウンド 885
バッファー・アドレス設定命令 887
RA 命令 887
SBA オーダー 887
データ・セット 584
空 543
順次 279
バッチ・データ交換 617
ブロック化 554
ユーザー 266
BDAM 553, 555
CICS アプリケーション・プログラムからのアクセス 567
データ・テーブル
カップリング・ファシリティー 558
共用 557
ディスプレイからの読み取り 821
出口プログラム
文書テンプレートとして 696
連絡域 696
テスト、アプリケーションの
アプリケーション・テーブル項目の準備 172
システムの準備 173
システム・テーブル項目の準備 172
順次装置の使用 171, 611
順次端末サポート 171
単一スレッド・テスト 171
マルチスレッド・テスト 171
レグレッション・テスト 171
デッドロック 277, 565
予防 549
デバッグ 175
デバッグ・モード 26
デフォルト
条件の処理 383
同期化処置
ジャーナル出力 378
同期点処理 380, 381
同期点追跡、DPL 415
統合 CICS 変換プログラム 81
統合変換プログラム 6, 81
同時ブラウズ 572
動的
ストレージ域 272
トランザクション・バックアウト・プログラム 398
トランザクション・ルーティング 332
プログラム 276
動的 LIBRARY 120

動的 LIBRARY リソースの使用 120
動的起動、変換プログラムの 87
登録コンテナー
DFHRL-ACTION 529
DFHRL-ERROR 529
DFHRL-PROGRAM 531
DFHRL-TYPE 531, 535
トランザクション
会話型 241
疑似会話型 241
デッドロック 563
非会話型 241
ルーティング 407, 408
ルーティング、動的 332
類縁性 326, 408, 643, 648, 654, 665
トランザクション ID
CEBR 203
CECI 217
CEDF トランザクション 175
トランザクション間の類縁性
アプリケーション生成プログラムが原因となる 369
安全なプログラミング手法 343
BTS コンテナーの使用 348
COMMAREA 343
ENQMODEL での DEQ の使用 346
ENQMODEL での ENQ の使用 346
TCTUA 345
一時記憶域データ共用
一時記憶域 357
関係および存続期間 370
グローバル関係 370
端末関係 372
ユーザー ID 関係 373
危険性のあるプログラミング手法
一時データ 360
グローバル・ユーザー出口 341
DELAY および CANCEL REQID コマンド 365
INQUIRE および SET コマンド 341
POST コマンド 367
RETRIEVE WAIT および START コマンド 361
START および CANCEL REQID コマンド 363
危険なプログラミング手法 349
共用ストレージの使用 350
タスク存続期間ストレージの共用 353
CWA 349
DEQ の使用 355
ENQ の使用 355
トランザクション間の類縁性 *(続き)*
危険なプログラミング手法 *(続き)*
LOAD PROGRAM HOLD の使用 351
WAIT EVENT の使用 354
検出 368
推奨 343
プログラミング手法 341
類縁性存続期間 369
類縁性トランザクション・グループ 370
トランザクション作業域 260
トランザクションとシステム間の類縁性 340
トランザクションの類縁性
トランザクション間の類縁性 340
トランザクションとシステム間の類縁性 340
トランザクション分離 653
トリガー・フィールド 880
トレース
説明 401
トレース入り口点 402

[ナ行]

名前付きカウンター 717
オプション・テーブル 719
概説 717
カウンター名 718
カップリング・ファシリティー・リスト構造 718
現行値 718
最小値 718
最大値 718
名前付きカウンター・フィールド 718
プール 718
CICS API 721
DFHNC001 720
DFHNCO マクロ 719
入力順データ・セット (ESDS) 542
入力データ
チェーニング 607
入力データのマップ 821
入力マップ、記号 819
ヌル値の使用 888
ヌル・パラメーターを持つ DFHNCTR CALL の例 732
ネイティブ・ランタイム・ライブラリー 11

[ハ行]

パージ遅延 859
パートナー、会話 591
パートナーの範囲 423
排他制御、レコードの
BDAM 579
VSAM 549
VSAM RLS 550
排他的リソース 276
バイナリー・データの XML への変換 504
バイナリー・データへの XML の変換 506
バインディング 520, 522
バインディング・ファイル
XML バインディング 500
バックアウト、リソースの 380
バッチ・データ交換 617
宛先の識別 618
確定応答 618
DEFRESP オプション 618
ISSUE WAIT コマンド 619
NOWAIT オプション 619
バッチ・ユーティリティー
XML 支援機能 430
パラメーター
null 732
範囲、パートナーの 423
バンドル
登録インターフェース 527
登録プログラム 528
マニフェスト・スキーマ 535
ユーザー・リソース 527
半二重モード 591
反復内容 494
反復フィールド 792
非 CICS プリンター 741, 755
非会話型プログラミング 241
ビジネス・サービス 515
日付フィールド、EIB の 6
非同期ジャーナル出力 378
非同期処理 407, 423
非ブロック化引数 556
表示
画面 287
レジスター、EDF の 197
表示スペース 867
表示特性 804
表示窓 867
表題、メッセージの 860
ファイル制御
概説 541
BDAM データ・セット 555
フィールド
グループ 791
反復 792
複合 791
ブランク 886
BMS 779

複合フィールド 791
複数ページ出力 888
複数マップ画面 890
ブックマーク 707, 709
物理キー 555
物理マップ・セット
インストール 161
浮動マップ 839
ブラウズ 586
レコード 571
DELAY 586
SUSPEND 586
ブラウズ操作
BDAM 575
フラグ・バイト、経路リスト 857
ブラケット・プロトコル、LAST オプション 610
ブランク・フィールド 886
ブリッジ (3270)
ADS 記述子 790
フリップフロップ・モード 591
プリンシパル装置 244, 245
プリンター
非 CICS 741, 755
3270 743
オプション 744
CICS 741
特性の判別 751
SCS 746
プログラミング上の制約事項
アセンブラー言語 71
C および C++ 54
COBOL 26, 36, 41
PL/I 66
VS COBOL II 31
プログラミング手法
アセンブラー言語 71, 73, 76
一般 271
C および C++ 54
COBOL 26, 33, 36
PL/I 65, 66, 67, 68
プログラミング・モデル 241
プログラム
サイズ 271
テスト 175
文書テンプレートとして 693
プログラム制御
他のプログラムへのデータの受け渡し 327
プログラムの論理レベル 326
別のプログラムへのリンク 326
プログラム設計
会話型 241, 621
疑似会話型 241
非会話型 241
プログラムに渡されたコンテナーの検出 305
プログラムへのリンク、戻りを前提とした 326
プログラム・ストレージ 263
プログラム・ラベル、EDF の 193
ブロック化データ・セット 554
ブロック参照 555
フロントエンド・プログラミング・インターフェース (FEPI) 408
分散アプリケーションの設計 248
分散プログラム・リンク 325
オプション 411
クライアント領域 410
サーバー領域 410, 413
サーバー・プログラム 412
独立同期点 415
プログラミング上の考慮事項 417
例外条件 420
COMMAREA オプション 413
DPL API サブセット 419
REMOTENAME オプション 412
REMOTESYSTEM オプション 413
SYSID オプション 413
TRANSID オプション 414
文書 683
および CICS Web サポート 683
コード・ページ変換 689
再利用 710
削除 712
作成 701
取得 710
データの置換 709
データの追加 707
ブックマーク 707, 709
文書テンプレート 683
一時記憶キュー 692
一時データ・キュー 692
埋め込みコマンド
#echo 698, 699
#include 699
#set 684, 698, 699
および CICS Web サポート 683, 689, 692
記号 683, 684, 698
記号リスト 684
キャッシング 687
出口プログラムでの指定 696
区分データ・セット (PDS) 691
設定 691
出口プログラム 696
文書内での位置指定 701, 707, 709
連絡域 696
CICS ファイル 692
CICS プログラム 693, 694
DFHDHTL マクロ 693, 694
文書テンプレート *(続き)*
HFS ファイル 692
文書テンプレートの記号 683
文書テンプレートのキャッシング 687
文書テンプレートの連絡域 696
ページ作成操作 888
ページ不在 273
ページング
影響の軽減 273
ページ・オーバーフロー 861
ページ・ルーティング操作 888
変換 6, 81
COBOL 41
変換プログラム
言語環境プログラム準拠のコンパイラーとの統合 81
動的起動 87
変換プログラム・データ・セット 84, 88
ペン検出可能フィールド 881
変更データ・タグ 823, 885
変数、CECI/CECS 219
補助一時記憶域 263
補助記憶装置
一時データ 678
補助トレース 278
ホスト・コード・ページ 689

[マ行]

マイグレーション 323
チャネルを使用する LINK コマンドの 321
チャネルを使用する RETURN コマンドの 322
チャネルを使用する START データの 323
チャネルを使用する XCTL コマンドの 322
COMMAREA からチャネルへ 321
待ち、端末 594
マップ
記号出力 819
記号入力 819
作成 781
出力の初期化 803
セット 275, 789
装置依存 797
データの移動 803
浮動 839
リンク・エディット 788
BMS 777, 886, 890
マップのリンク・エディット 788
マップ・セット
追加、CSECT 167
プログラムにおける使用、シンボリック記述マップ・セットの 137

マップ・セット *(続き)*
ロード、16MB 境界より上のアドレスに 158
マニフェスト
スキーマ 535
マルチスレッド化 249
マルチスレッド・テスト 171
満了時刻
指定 644
メイン一時記憶域 263
メッセージ、送達不能な 859
メッセージの表題 860
メッセージ・ルーティング 853
文字セット 689
モジュラー・プログラム 275

[ヤ行]

ユーザー
ストレージ 261
データ・セット 266
トレース入り口点 402
ユーザー置換可能モジュール 658
ユーザー保守テーブル 557
ユーザー・キー・ストレージ 655
ユーザー・リソース 527
ユーティリティー・プログラム
XML 支援機能 430
要求応答単位 (RU) 607
呼び出し、EDF の 176
呼び出し可能サービス 12
読み取り、形式設定画面からの 822
読み取り、ディスプレイからのデータの 821
読み取り、レコードの 567
読み取り専用 DSA (RDSA) 134
読み取り専用コンテナー 306

[ラ行]

ライト・ペン検出可能フィールド 881
ライブラリー・ルックアサイド機能 (LLA) 133
ランタイム・ライブラリー
言語環境プログラム 11
ネイティブ 11
ランナウェイ・タスク 277
リカバリー
順次端末サポート 611
同期点 380
問題回避 268
リソースの 276
リカバリー可能リソース 242
排他使用 242
リソース
アクセス・シーケンスの制御 648
競合 242
制御 242
排他使用 242
排他制御 276
リカバリー可能 242, 276
リソース・タイプの登録 527
リモート・トランザクション、EDF の 191
リモート・リンクされているプログラム
DPL 192
EDF 191
領域間通信 424
リンクから返されたコンテナーの検出 305
リンク・エディット 81, 87
リンク・パック域 (LPA) 133
ルーティング、トランザクション 407
ルーティング端末 862
類縁性 339
ルックアサイド・トランザクション 869
例
コンテナー、基本的な 290
チャネル、基本的な 290
チャネルを構成するクライアント・プログラム 308
チャネルを使用する CICS サーバー・プログラム 309
複数の対話式コンポーネント 294
複数のチャネルに 1 つのコンポーネント 294
1 つのチャネル - 1 つのプログラム 293
1 つのチャネル - 複数のプログラム 293
BTS アクティビティーと比較して単純なクライアント・プログラム 309
例外条件
説明 383
HANDLE CONDITION コマンド 388, 389
IGNORE CONDITION コマンド 393
例外トレース入り口点 402
レグレッション・テスト 171
レコード
更新 576
削除 579
作成 576, 581
識別 545, 555
ジャーナル 377
追加 581
長さ 267
ブラウズ 567
読み取り 567
ロック 549
レコード *(続き)*
ロック (RLS) 550
BDAM データ・セットへの追加 583
レコード記述フィールド 584
レコード・レベル共用 (RLS)
RLS モードでのファイルのアクセス 544
レコード・ロック 550
レベル、アプリケーション・プログラム論理の 326
ローカル・コピー・キー 758
ロード・ライブラリー
サポート、2 次エクステントの 140
ロギング 280
論理作業単位 (LUW)
使用する同期点 380
説明 243
リカバリー可能リソース 243
論理装置 (LU)
機能 607
論理デバイス・コンポーネント 877
論理メッセージの規則 833
論理レコードの提示 608
論理レベル、アプリケーション・プログラムの 34, 326

[ワ行]

割り込み 593

[数字]

10/63 磁気スロット読み取り装置 879
2 次エクステント、CICS ロード・ライブラリー 140
31 ビット・アドレッシング
アセンブラー言語 71
COBOL 26
31 ビット・モード・トランザクション 332
3262 プリンター 741
3270 画面フィールド 823
3270 情報表示システム 161
3270 ディスプレイ 595
3270 ファミリー
アウトバウンド・データ・ストリーム 633
アテンション・キー 636
インバウンド・フィールド形式 638
書き込み制御文字 625
拡張属性 628
カラー、基本 628
輝度 628
基本カラー 628
属性、拡張 628

3270 ファミリー *(続き)*
端末への書き込み 624
データ・ストリーム 623
データ・ストリーム、アウトバウンド 633
データ・ストリーム中のオーダー 629
データ・ストリーム・オーダー 629
入力 635
バッファー 623
表示特性 626
フィールド 625
フィールド形式、インバウンド 638
フィールド属性 626
不定形式モード 639
変更データ・タグ 627
保護 627
読み取り 637
MDT 627
3270 ブリッジ
ADS 記述子 790
3270 プリンター 743
オプション 744
3289 プリンター 741
3290 ディスプレイ 867
文字サイズ 870
32K 以上の COMMAREA 289, 311, 319
3601 論理装置 877
3770 バッチ論理装置 877
3770 バッチ・データ交換論理装置 877
3790 バッチ・データ交換論理装置 877

A

ABEND コマンド 398
ACCEPT ステートメント、COBOL 26
ACCUM オプション 745, 807, 833
ACK 593
ACTPARTN オプション 833, 872
ADDRESS COMMAREA コマンド 328
ADDRESS コマンド 5
ADDRESS 特殊レジスター 33
ADS 記述子 790
AFTER オプション 644
ALARM オプション 808
ALLOCATE コマンド 278
待機禁止、NOSUSPEND オプション 278
ALLOCERR 状態 763
ALTPAGE 値 839
AMODE (アドレッシング・モード)
オプション、CICS アプリケーションの 131
APAK トランザクション 749
APCG 272
API
DPL のサブセット 419
APOST オプション 92
application/x-www-form-urlencoded 684
argc 54
argv 54
ASIS オプション 823
ASKTIME コマンド 643
ASSIGN コマンド 5, 603, 751
オプション 603
DESTCOUNT オプション 861
MAPCOLUMN オプション 843
MAPHEIGHT オプション 843
MAPLINE オプション 843
MAPWIDTH オプション 843
MSR オプション 879
PAGENUM オプション 861
AT オプション 644
ATI 244, 592, 675
ATNI 409
Atom フィード
XML バインディング
作成 500
ATTENTION キー 593
AUTOPAGE オプション 836
AZI6 409

B

BAKR (分岐およびスタック) アセンブラー命令 71
BASE オプション 802
BDAM 586
更新操作 579
データ・セット 553, 555
排他制御 579
ブラウズ操作 575
BDI 741
BGAM 589
BMS 742
ルーティング 755
BMS コマンド 587
BOTTOM コマンド、CEBR トランザクション 207
BRACKET オプション 610
BROWSE TEMP STORAGE オプション、CEDF 195
BTS アクティビティー 309
BUILDCHAIN 607

C

C および C++ 53
作業用ストレージ 53
サポート 53
制約事項 54
他の言語との混合 16
C および C++ *(続き)*
地域サポート 60
引数 58
プログラミング手法 54
EIB、アクセス 60
XML スキーマへのマッピング 463, 465
XPLink 61, 62, 63
C および C++ での地域サポート 60
C および C++ へのマッピング 463, 465
C 言語に関する考慮事項
LENGTH オプションのデフォルト 267
struct、シンボリック記述マップ・セット 159
CALL DL/I インターフェース、COBOL 30
CALL ステートメント
アセンブラー言語 76
CANCEL コマンド 643
CBLCARD オプション 92
CDUMP 54
CEBR トランザクション 203
一時記憶域のブラウズ 203
一時データ 210
開始 203
コマンド域 206
セキュリティーの考慮事項 203
表示 205
ブラウズ・トランザクション 203
ヘッダー 205
本体 206
メッセージ行 206
BOTTOM コマンド 207
CEBR の開始 203
COLUMN コマンド 207
FIND コマンド 207
GET コマンド 208
LINE コマンド 208
PURGE コマンド 208
PUT コマンド 209
QUEUE コマンド 209
SYSID コマンド 209
TERMINAL コマンド 209
TOP コマンド 209
CECI トランザクション
アンパーサンド (&) 220
概要 211
拡張域 219
画面レイアウト 211
共用ストレージ 225
コマンド行 211
コマンド構文検査 213
コマンド実行開始 214
コマンド実行完了 214
コマンド入力 211

CECI トランザクション *(続き)*
コマンド入力行 211
状況域 212
情報域 215
端末の共用 224
プログラム制御 224
変更方法 218
変数 219
本体 215
メッセージ行 216
ENQ コマンド 225
PF キー値域 216
CECS トランザクション 217
CEDF トランザクション 175, 176
一時記憶域のブラウズ 195
画面の重ね書き 193
疑似会話型プログラム 190
機能 175
機能 (PF) キーによるオプション 195
実行の修正 193
セキュリティー 179
単一画面モード 188
端末を使用しないトランザクション 191
二重画面モード 190
表示 179
表示レジスター 197
プログラム・ラベル 193
ヘッダー 180
本体 180
ユーザー・タスクの異常終了 195
呼び出し 176
リモート・トランザクション 191
リモート・リンクされているプログラム 191
CECI の呼び出し 196
DPL 192
EDF トランザクション 176
PF キー 179
CEEBINT、言語環境プログラム HLL ユーザー出口 23
CEEBXITA ユーザー出口 21
CEECSTX ユーザー出口 21
CEEDOPT CSECT 19
CEEENTRY マクロ 73
CEEHDLR サービス 14
CEEROPT CSECT 19
CEEUOPT CSECT 19
CEEWUCHA サンプル・ユーザー条件処理ルーチン 31, 67
CESE 一時データ宛先 148
CESF、GOODNIGHT トランザクション 613
CHANNEL
オプション 286, 325, 330
LINK コマンド 325
CHANNEL *(続き)*
RETURN コマンド 325
XCTL コマンド 325
CICS
テスト環境 7
CICS 値データ域 267
CICS 域のアドレッシング 886
CICS オプション 92
CICS キー・ストレージ 655
CICS ダンプ・ユーティリティー・プログラム 403
CICS ファイル
文書テンプレートとして 692
CICS プリンター 741
特性の判別 751
CICS プログラム
文書テンプレートとして 693
DFHDHTL マクロ 693, 694
CICS 文書のコード・ページ 689
CICS 保守テーブル 557
CICSDATAKEY オプション 261, 659
CICSVAR 環境変数 22
CLASS オプション 767
CLEAR
キー 874
PARTITION AID 値 874
PARTITION キー 874
CLEAR キー 287
CLOCK 54
CMT 557
CNOTCOMPL オプション 607
COBOL 25
アドレッシング、CICS データ域の 33
可変反復内容 494
グローバル変数 41
コンパイラー・オプション 26
作業用ストレージ 25
サブプログラムの呼び出し 34, 36
サポート 25
参照修正 41
実行単位 34
制約事項 26, 36, 41, 274
他の言語との混合 16
デバッグ・モード 26
ネストされたプログラム 45
バッチ・コンパイル 43
ブランク行 41
プログラミング上の制約事項 VS COBOL II 31
プログラミング手法 26, 33, 36
変換 41
予約語テーブル 26
31 ビット・アドレッシング 26
ADDRESS 特殊レジスター 33
CALL DL/I インターフェース 30
COBOL *(続き)*
CBLPSHPOP ランタイム・オプション 29
DFHNCTR 呼び出しの例 732
OS/VS COBOL 50
REPLACE ステートメント 41
XML スキーマへのマッピング 453, 458
COBOL のグローバル変数 41
COBOL のネストされたプログラム 45
COBOL プログラムのバッチ・コンパイル 43
COBOL へのマッピング 453, 458
COBOL2 オプション 92
COBOL3 オプション 93
COLUMN
コマンド、CEBR トランザクション 207
COM アセンブラー命令 71
COMMAREA 242, 261, 262, 273
オプション 286, 325, 327, 329
LINK コマンド 325
COMMAREA の現代化 289
COMMAREA を渡す LINK コマンドのマイグレーション 321
COMMAREA を渡す XCTL コマンドのマイグレーション 322
COMMAREA > 32K 289, 311, 319
CONNECT PROCESS コマンド 422
CONSISTENT オプション
READ コマンド 568
CONVERSE コマンド 595, 611, 621
COPY ステートメント 105
CPI
参照 5
CPI コミュニケーション・インターフェース・モジュール、DFHCPLC 106
CPI コミュニケーション・スタブ 424
CPI-C 407, 423
CPSM オプション 93
CQRY トランザクション 247
CSECT、マップのアセンブルへの追加 167
CSNAP 54
CSPG トランザクション 755, 760, 835, 836
CSPP トランザクション 247
CTDLI 54
CTEST 54
CTLCHAR オプション 745
CTRACE 54
CURSOR オプション 807, 813, 833
CVDA 267, 406
CWA 281
CWAKEY パラメーター 282

D

DATAONLY オプション 807, 809, 887
DBCS オプション 93
DCB インターフェース、TCAM の 615
DD 名リスト、変換プログラムの動的起動における 88
DDS 798
DEBKEY オプション 575
DEBREC オプション 556, 575
DEBUG オプション 93
DEFRESP オプション 618, 621
  端末管理 608
DELAY コマンド 643, 644
DELETE ステートメント、COBOL 26
DELETEQ TD コマンド 673
DELETEQ TS コマンド 677
DEQ コマンド 647
DEQUEUE コマンド 754
DESTCOUNT オプション 861
DESTID オプション 618
DESTIDLENG オプション 618
DFH3QSS 7
DFHAID 54
DFHAPXPO 19
DFHASMVS プロシージャー 162, 164, 168
DFHBMSCA 54, 806
DFHBMSCA 定義 825
DFHBMSUP 791
DFHCOMMAREA 26, 328
DFHCPLC 424
DFHCPLC、CPI コミュニケーション・インターフェース・モジュール 106
DFHCPLRR、SAA リソース・リカバリー・インターフェース・モジュール 106
DFHDHTL マクロ 693, 694
DFHDHTXD 696
DFHDHTXH 696
DFHDHTXL 696
DFHDHTXO 696
DFHEAI インターフェース・モジュール、アセンブラーの 107
DFHEAI0 インターフェース・モジュール、アセンブラーの 107
DFHEAP1$ 変換プログラム、アセンブラーの 87
DFHECP1$ 変換プログラム、COBOL の 87
DFHEDF グループ 173
DFHEDP1$、変換プログラム、C の 87
DFHEIBLK 26
DFHEIEND マクロ 100, 102
DFHEIENT マクロ 76, 100, 102
DFHEIRET マクロ 76, 94, 98
DFHEISTG マクロ 100, 102
DFHEITAL プロシージャー 138, 141
DFHEIVAR 26
DFHELII 109
DFHELII、言語環境プログラム準拠コンパイラー用のインターフェース・モジュール 107
DFHEPP1$ 変換プログラム、PL/I の 87
DFHEXEC 109
DFHEXTAL プロシージャー 138
DFHFCT マクロ 557
DFHLNKVS プロシージャー 162, 168
DFHLS2SC
  カタログ式プロシージャー 431
DFHLS2SC、CICS XML 支援機能プログラム 500
DFHMAPS プロシージャー 138, 165
DFHMAPT
  インストール用プロシージャー、HTML テンプレートの 166
DFHMDF マクロ 781
  表示特性 804
  DSATTS オプション 804
  MAPATTS オプション 804
DFHMDI マクロ 781, 784
DFHMIRS プログラム 415
DFHMSCAN ユーティリティー・プログラム 120
DFHMSD マクロ 781, 784
  BASE オプション 802
  STORAGE オプション 802
DFHMSD、マップ・セットのアセンブル用マクロ 160
DFHMSRCA 54, 879
DFHNC001
  デフォルトの名前付きカウンター・プール 720
DFHNCO マクロ
  名前付きカウンター・オプション・テーブル 719
DFHNCOPT
  名前付きカウンター・オプション・テーブル 719
DFHNCTR
  ヌル・ポインターを持つ COBOL 呼び出しの例 732
DFHPDI マクロ 870
DFHPEP プログラム 398
DFHPSD マクロ 169, 869
DFHRESP 変換プログラム機能 86, 384
DFHRL-ACTION コンテナー 529
DFHRL-CONTROL コンテナー 533
DFHRL-DATA コンテナー 534
DFHRL-ERROR コンテナー 529
DFHRL-NAME コンテナー 535
DFHRL-PROGRAM コンテナー 531
DFHRL-ROOT コンテナー 535
DFHRL-TYPE コンテナー 531, 535
DFHSC2LS
  カタログ式プロシージャー 437
DFHURLDS 858
DFHVALUE 86
DFHYITDL プロシージャー 138, 151
DFHYITEL プロシージャー 138, 151
DFHYITPL プロシージャー 138, 147
DFHYITVL プロシージャー 138, 144
DFHYXTDL プロシージャー 138, 151
DFHYXTEL プロシージャー 138, 151
DFHYXTPL プロシージャー 138, 147
DFHYXTVL プロシージャー 138, 144
DFHZITCL プロシージャー 138, 145
DFHZITPL プロシージャー 138, 149
DIB 86
DISPLAY ステートメント、COBOL 26
DLI 94
DLI オプション 94
DLL 18
DL/I
  参照 5
  同期点 381
DOCTEMPLATE リソース定義 683, 691
  EXITPGM 属性 696
  FILE 属性 692
  HFSFILE 属性 692
  MEMBERNAME 属性 691
  PROGRAM 属性 693
  TDQUEUE 属性 692
  TSQUEUE 属性 692
DOCTOKEN 701, 712
DOCUMENT CREATE コマンド 683, 701
  DELIMITER オプション 703, 704
  DOCSIZE オプション 710
  DOCTOKEN 701
  LISTLENGTH オプション 703
  SYMBOLLIST オプション 683, 684, 701, 703, 704
DOCUMENT DELETE コマンド 712
  DOCTOKEN 712
DOCUMENT INSERT コマンド 683, 707, 709, 710
DOCUMENT RETRIEVE コマンド 683, 710
  CLNTCODEPAGE オプション 689, 710
  HOSTCODEPAGE オプション 689
DOCUMENT SET コマンド 683
  DELIMITER オプション 703, 704
  LISTLENGTH オプション 703
  SYMBOL オプション 683, 684, 703, 704

DOCUMENT SET コマンド *(続き)*
SYMBOLLIST オプション 683, 684, 703, 704
DPL 192, 325, 381, 407, 409
DSA 272
DSATTS オプション 804
DTP 407, 423
DUMP TRANSACTION コマンド 403
DUPKEY 状態 572

E

ECBLIST 648
EDF 86, 93, 175
EDF オプション 94
EIB 86, 383, 592
説明 6
端末管理フィードバック 604
EIBCALEN フィールド 328
EIBCOMPL フィールド 596
EIBFN フィールド 329
ENDBR コマンド 571
ENQ コマンド 278, 647
ENQBUSY 状態 278
ENQUEUE コマンド 754
EOC 状態 607, 830
EODI 文字 612
EODS 状態 607
EPILOG オプション 94
EQUAL オプション 568
ERASE オプション 745, 807, 833
ERASEAUP オプション 807, 833, 847
ERDSA 134
ESDS
拡張アドレス方式へのアップグレード 547
ESDS (入力順データ・セット) 542
EXCI
オプション 94
通信 424
CALL 424
EXCI - 外部呼び出しインターフェース 407
EXEC インターフェース・スタブ 107
EXEC インターフェース・ブロック 86
EXEC インターフェース・モジュール 106, 107, 154
EXECKEY 262, 282
EXECKEY パラメーター 656
EYUVALUE 86

F

FEPI
参照 5
FEPI オプション 95
FEPI - フロントエンド・プログラミング・インターフェース 408
FETCH 54
FETCHABLE オプション 68
FIND コマンド、CEBR トランザクション 207
FLAG オプション 95
FLOAT コンパイラー・オプション 65
FMH 609
アウトバウンド 609
インバウンド 609
オプション 609
FMHPARM オプション 833
FOR オプション 644
FORMATTIME コマンド 643
FORMFEED オプション 747, 833
FREE コマンド 610
FREEKB オプション 808
FREEMAIN コマンド 654
FROM オプション 807
FRSET オプション 807

G

GDDM 813
GDS オプション 95
GENERIC オプション 545, 568
GET コマンド、CEBR トランザクション 208
GETMAIN コマンド 261
CICSDATAKEY オプション 261, 659
INITIMG オプション 274, 654
NOSUSPEND オプション 654
SHARED オプション 260, 266, 654
TASKDATAKEY オプション 261
USERDATAKEY オプション 261, 659
GOODNIGHT トランザクション、CESF 613
GRAPHIC オプション 95
GRPNAME オプション 791
GTEQ オプション 545, 568

H

HANDLE ABEND LABEL の制約事項、アセンブラー言語での 71
HANDLE ABEND コマンド 384, 388, 397
HANDLE AID コマンド 824
HANDLE CONDITION ERROR コマンド 392
HANDLE CONDITION コマンド 384, 388, 395
HFS ファイル
文書テンプレートとして 692
HOLD オプション 325
HONEOM オプション 745
HTML テンプレート
インストール 166

I

IBM 表示画面定義機能 II (SDF II) 158
IBMWRLKC リンケージ・エディターの入力 67
ICTL (入力形式制御) アセンブラー命令 71
ICVR パラメーター 173
IGNORE CONDITION コマンド 384, 388, 392, 393
IGREQID 状態 833
IGYCCICS 26
IGZWRLKA 31
IMMEDIATE オプション 329, 592, 610
IMS.RESLIB (IMS ライブラリー) 156
INBFMH 状態 607
INITIMG オプション 274, 654
INPUTMSG オプション 325, 330, 332
INQUIRE TERMINAL コマンド 604, 751
INQUIRE コマンド 6
INRTN オプション 872
INTERVAL オプション 644
INVITE オプション 591
INVOKE SERVICE 520, 522, 524
INVPARTN 状態 874
INVPARTNSET 状態 874
INVREQ 状態 833
IPCS 403
IRC 424
iscics 7
ISCINVREQ 409
ISSUE ABORT コマンド 618
ISSUE ADD コマンド 617
ISSUE COPY コマンド 594, 759
ISSUE DISCONNECT コマンド 593
ISSUE END コマンド 618
ISSUE ERASE コマンド 594, 618
ISSUE NOTE コマンド 618
ISSUE PRINT コマンド 759
ISSUE QUERY コマンド 617
ISSUE RECEIVE コマンド 617
ISSUE REPLACE コマンド 617
ISSUE SEND コマンド 617
ISSUE WAIT コマンド 618, 619

J

JCICS
  およびチャネル 311
JCICS からのチャネルの使用 311
JES 6, 761
JES (ジョブ入力サブシステム)
  スプーラー・コマンド 763
  出口 766
  入力 766
  JES スプールからのデータの検索 761, 762
  RESP および RESP2 オプション 763
JOURNALNAME 380
JOURNALNUM 380
JTYPEID 380
JUSFIRST オプション 848
JUSLAST オプション 848
JUSTIFY オプション 848

K

KSDS (キー順データ・セット) 541

L

LAST オプション 610, 808
  ブラケット・プロトコル 610
LDC 877
LDCMNEM オプション 878
LDCNUM オプション 878
LEASM オプション 96
LENGERR 状態 596
LENGTH オプション 96, 266, 596
LENGTHLIST オプション
  タスク関連ストレージ域の複数のダンプ 404
LIBRARY 120
LINE コマンド
  CEBR トランザクション 208
LINECOUNT オプション 96
LINK PROGRAM 424
LINK コマンド 261, 271, 272, 326
  CHANNEL オプション 325, 330
  COMMAREA オプション 325, 327, 329
  IMMEDIATE オプション 329
  INPUTMSG オプション 325, 330
  TRANSID オプション 329
LINK コマンド、チャネルを使用するマイグレーション 321
LINKAGE オプション 96
LIST オプション 854
LLA (ライブラリー・ルックアサイド機能) 133
LOAD コマンド
  HOLD オプション 325
LPA 133
LU タイプ 4
  装置 593
  バッチ・データ交換 618
  論理レコードの提示 608
LU (論理装置)
  機能 607

M

MAPATTS オプション 804
MAPCOLUMN オプション 843
MAPFAIL 状態 822, 829
MAPHEIGHT オプション 843
MAPLINE オプション 843
MAPONLY オプション 807, 809, 886
MAPPED オプション 851
MAPSET オプション 807
MAPSET リソース定義 788
MAPWIDTH オプション 843
MARGINS オプション 97
MASSINSERT オプション 581, 585
maxOccurs
  XML スキーマ内 479
MDT 823, 885
MERGE ステートメント、COBOL 26
minOccurs
  XML スキーマ内 479
MONITOR POINT コマンド 403
MONITOR コマンド 403
MSGINTEG オプション 621
MSR 879
MSR オプション 833, 879
MVS サブスペース 668
MVS トランザクション 332
MXT パラメーター 277

N

NATLANG オプション 98
NLEOM オプション 743, 745, 746, 833
NOAUTOPAGE オプション 836
NOCBLCARD オプション 98
NOCPSM オプション 98
NODE オプション 764
NODEBUG オプション 98
NOEDF オプション 98
NOEDIT オプション 851
NOEPILOG オプション 98
NOFEPI オプション 99
NOFLUSH オプション 840, 847
NOHANDLE オプション 384, 390
NOJBUFSP 状態 278
NOLENGTH オプション 99
NOLINKAGE オプション 99
NONUM オプション 99
NOOPSEQUENCE オプション 99
NOOPTIONS オプション 99
NOPROLOG オプション 100
NOQUEUE オプション 278
NOSEQ オプション 100
NOSEQUENCE オプション 90, 100
NOSOURCE オプション 100
NOSPACE 状態 391
NOSPIE オプション 100
NOSUSPEND オプション 278
  GETMAIN コマンド 654
  READ コマンド 568
  READNEXT コマンド 579
  READPREV コマンド 579
  WRITE コマンド 581
NOTRUNCATE オプション 596
NOVBREF オプション 100
NOWAIT オプション 619
NOXREF オプション 100
NUM オプション 101
NUMREC オプション 579
NUMSEGMENTS オプション
  タスク関連ストレージ域の複数のダンプ 404

O

OCCURS オプション 792
OO COBOL
  サポート 25
OPCLASS オプション 854
OPEN ステートメント、COBOL 26
OPID オプション 854
OPIDENT 値 854
OPMARGINS オプション 101
OPSEQUENCE オプション 101
OPSYN (命令コード等価) アセンブラー命令 71
OPTIONS オプション 101
OPTIONS(MAIN) の指定 65
OS/VS COBOL 50
  サポート 25
OTE、オープン・トランザクション環境 251
OVERFLOW 状態 840

P

PA キー 749
PAGENUM オプション 861
PAGESIZE 値 839
PAGING オプション 745, 807, 814

PARTITIONSET オプション 871
PARTN オプション 872
PARTNFAIL 状態 874
PARTNPAGE オプション 873
PERFORM DUMP コマンド 403
PERFORM コマンド 6
PF (プログラム機能) キー 179, 205, 216
PLT プログラム 658
PL/I 65
  言語環境プログラムの要件 67
  制約事項 66
  他の言語との混合 16
  取り出したプロシージャー 68
  プログラミング手法 65, 66, 67, 68
  FLOAT コンパイラー・オプション 65
  OPTIONS(MAIN) の指定 65
  XML スキーマへのマッピング 469, 473
PL/I へのマッピング 469, 473
POP HANDLE コマンド 388, 398
POST コマンド 643, 644
PRGDLY オプション 859
PRINTERCOMP オプション 747
PROGRAM オプション 397
PROGRAM リソース定義 788
PROLOG オプション 101
PROTECT オプション 621
PURGE MESSAGE コマンド 834, 853
PURGE コマンド、CEBR トランザクション 208
PUSH HANDLE コマンド 388, 398
PUT コマンド、CEBR トランザクション 209

Q

QBUSY 状態 278
QUERY SECURITY コマンド 405
  NOLOG オプション 406
  RESCLASS オプション 406
  RESID オプション 406
  RESTYPE オプション 406
QUEUE コマンド、CEBR トランザクション 209
QUOTE オプション 102
QZERO 状態 676

R

RACF 405
RBA (相対バイト・アドレス) 542, 546
RDF 584
READ コマンド 576
  CONSISTENT オプション 568
READ コマンド *(続き)*
  NOSUSPEND オプション 568
  REPEATABLE オプション 568
  UNCOMMITTED オプション 568
READ ステートメント、COBOL 26
READNEXT コマンド 571
  NOSUSPEND オプション 579
READPREV コマンド 571
  NOSUSPEND オプション 579
READQ TD コマンド 278, 673
READQ TS コマンド 677
  ITEM オプション 678
RECEIVE MAP コマンド 821
  ASIS オプション 823
RECEIVE PARTN コマンド 873
RECEIVE コマンド 592, 593, 594, 595, 611, 621
  MAPFAIL 状態 829
RELEASE 54
RELEASE オプション 835
RELEASE コマンド
  HOLD オプション 325
RELTYPE キーワード 554
REMOTENAME オプション 412
REMOTESYSTEM オプション 413
REPEATABLE オプション
  READ コマンド 568
REPLACE ステートメント 41
REQID オプション 572, 645, 833, 860
RESCLASS オプション 406
RESETBR コマンド 571
RESID オプション 406
RESP 値 383
RESP オプション 384, 385, 390
  NOHANDLE の非活動化 390
RESP および RESP2 オプション
  JES に対するインターフェース 763
RESP2 値 383, 384
RESP2 オプション 384
RESTYPE オプション 406
RETPAGE 状態 816, 848
RETRIEVE コマンド 643, 649
RETURN コマンド 263, 329
  CHANNEL オプション 286, 325
  COMMAREA オプション 286, 325
  IMMEDIATE オプション 592, 610
  INPUTMSG オプション 325, 330, 332
  TRANSID オプション 244
RETURN コマンドでの COMMAREA のマイグレーション 322
REWRITE コマンド 576
REWRITE ステートメント、COBOL 26
RIDFLD オプション 554, 571
RMODE コンパイラー・オプション 26
RMODE (常駐モード)
  オプション、CICS アプリケーションの 131
ROUTE コマンド 853
  ページ・オーバーフロー 861
  LIST オプション 854
  TITLE オプション 860
ROUTEDMSGS オプション 855
RPTOPTS 19
RRDS (相対レコード・データ・セット) 542
RRN (相対レコード番号) 542, 546
RTEFAIL 状態 857
RTESOME 状態 857
RU (要求応答単位) 607
RUWAPOOL 15
RVI 593

S

SAA リソース・リカバリー 380
SAA リソース・リカバリー・インターフェース・モジュール、DFHCPLRR 106
SAM 589
SCA 515
  概説 515
  コンポジット 517
  チャネル・ベース・サービス 520
  ベスト・プラクティス 519
  ワイヤリング 517
  SCA コンポジットの作成
    ベスト・プラクティス 519
  XML ベース・サービス 522
SCA スキーマ 537
SCA の問題の診断 525
SCA の問題のトラブルシューティング 525
SCS
  プリンター 746
SCS 入力 749
SDF II 781, 789
SDF II (IBM 表示画面定義機能 II) 158
SEC システム初期設定オプション 406
SEGMENTLIST オプション
  タスク関連ストレージ域の複数のダンプ 404
SEND CONTROL コマンド 741, 809
SEND MAP コマンド 741, 801
  ACCUM オプション 807, 833
  ALARM オプション 808
  CURSOR オプション 807, 813
  DATAONLY オプション 807
  ERASE オプション 807
  ERASEAUP オプション 807, 847
  FREEKB オプション 808
  FROM オプション 807

SEND MAP コマンド *(続き)*
LAST オプション 808
MAPONLY オプション 807
MAPSET オプション 807
NOFLUSH オプション 840, 847
PAGING オプション 807
SET オプション 807
TERMINAL オプション 807, 814
WAIT オプション 808
SEND PAGE コマンド 381, 834, 853
AUTOPAGE オプション 836
NOAUTOPAGE オプション 836
RELEASE オプション 835
SEND PARTNSET コマンド 871
SEND TEXT コマンド 741, 847
MAPPED オプション 851
NOEDIT オプション 851
SEND コマンド 593, 595, 611, 621
CNOTCOMPL オプション 607
CTLCHAR オプション 745
FMH オプション 609
INVITE オプション 591
LAST オプション 610
MSR オプション 879
SEQ オプション 102
SEQUENCE オプション 102
Service Component Architecture
概説 515
診断 525
Service-Oriented Architecture 515
SESSBUSY 状態 278
SET
コマンド 6, 814
SET DOCTEMPLATE NEWCOPY コマンド 687
SET オプション 807, 814
SETLOCALE 54
SHARED オプション 260, 266
GETMAIN コマンド 654
SHARED オプション 262
SIGNAL 条件 593
SOA 515
SORT ステートメント、COBOL 26
SOURCE オプション 102
SP オプション 103
SPACE オプション 103
SPCOMMAND リソース・タイプ 406
SPIE オプション 103
SPOLBUSY 状態 766
SPOOLCLOSE コマンド 761
SPOOLOPEN
例 769
SPOOLOPEN コマンド 756, 761
NODE オプション 764
TOKEN オプション 764
USERID オプション 764
SPOOLREAD コマンド 761
SPOOLWRITE コマンド 761
SQL
参照 5
START コマンド 643, 649, 753
START ステートメント、COBOL 26
START データ、チャネルを使用するマイグレーション 323
STARTBR コマンド 571
STOP ステートメント、COBOL 26
STORAGE オプション 802
STORAGE ランタイム・オプション、言語環境プログラム 26
struct、C/370 シンボリック記述マップ・セット 159
SUSPEND コマンド 647
SVC99 54
SYNCONRETURN オプション 415, 422
SYNCPOINT コマンド 380, 381, 834
ROLLBACK オプション 398
SYSEIB オプション 103
SYSID オプション 413
SYSID コマンド、CEBR トランザクション 209
SYSIDERR 409
SYSIN 85
SYSPARM、マップ・セットのアセンブル用オペランド 160
SYSPRINT 86
SYSPUNCH 85
SYSTEM 54

T

TASKDATAKEY オプション 261, 658
TASKDATALOC オプション 103, 261
TCAM 615
DCB インターフェース 615
TCTUA 285, 658
TCTUAKEY 285, 658
TCTUALOC 285
TERM オプション 798
TERMID 値 854
TERMINAL
オプション 814
TERMINAL オプション 814
TERMINAL コマンド、CEBR トランザクション 209
TEST コンパイラー・オプション 26
TIOATDL 値 852
TITLE オプション 860
TOKEN オプション 577, 764
TOP コマンド、CEBR トランザクション 209
TRANISO 666
TRANSFORM DATATOXML 504
TRANSFORM XMLTODATA 506
TRANSID オプション 329, 414
TRUNC コンパイラー・オプション 26
TS キュー 357
TST TYPE=SHARED 359
TWA 260, 261
TWASIZE オプション 261
TYPE=DSECT アセンブル 788

U

UMT 557
UNCOMMITTED オプション
READ コマンド 568
UNTIL オプション 644
UOW 380
UPDATE オプション 576
USERDATAKEY オプション 261, 659
USERID オプション 764, 767

V

VBREF オプション 103
VLF (仮想ルックアサイド機能) 133
VOLUME オプション 618
VOLUMELENG オプション 618
VS COBOL II
言語環境プログラム 31
言語環境プログラムの呼び出し可能サービス 12
サポート 25
プログラミング 31
WORKING-STORAGE の制限 26
VSAM
エンキュー 277
データ・セット 584
プロセッサー・オーバーヘッド 585
MASSINSERT オプション 585
VTAM 589, 591

W

WAIT EVENT コマンド 354, 643
WAIT EXTERNAL コマンド 355, 648
WAIT JOURNALNUM コマンド
ジャーナル出力との同期化 377
WAIT TERMINAL コマンド 594
WAIT オプション 378, 379, 594, 808
WAITCICS コマンド 355, 648
WRITE JOURNALNAME コマンド 278, 378
WRITE JOURNALNUM コマンド 278, 378
ジャーナル・レコードの作成 377
WRITE コマンド 581

WRITE コマンド *(続き)*
NOSUSPEND オプション 581
WRITE ステートメント、COBOL 26
WRITEQ TD コマンド 673
WRITEQ TS コマンド 677
WRKAREA パラメーター 281
WSDL
言語構造への変換 437
WSDL 文書
可変長 497
空白 497

X

X8 および X9 TCB 62
XCTL コマンド 261, 263, 271, 272
CHANNEL オプション 325, 330
COMMAREA オプション 325, 327
INPUTMSG オプション 325, 330
XCTL コマンド、チャネルを使用するマイグレーション 322
XML
解析 508
支援機能 429
照会 507
データ型 508
データへの変換 429
変換 508
xsd:any データ型 509
XML 支援機能 430
DFHLS2SC 500
XML スキーマ 453, 458, 463, 465, 469, 473
可変長 497
空白 497
WSDL への変換 431
XML の解析 508
XML の照会 507
XML の変換 508
XML バインディング 500
XML ベース・サービス 522, 524
XML への言語構造のマッピング 500
XOPTS キーワード 89
XPCFTCH 63
XPCREQ グローバル・ユーザー出口 410, 414
XPCTA 63
XPLink 61
グローバル・ユーザー出口 63
非 XPLink オブジェクト 62
TCBs 62
XRBA (拡張相対バイト・アドレス) 546
XREF オプション 103
xsd:any データ型 509
XTC OUT 出口ルーチン 888
XTSEREQ、グローバル・ユーザー出口 359

[特殊文字]

#echo コマンド 698, 699
#include コマンド 699
#set コマンド 684, 698, 699

SC88-5850-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12